

Preface

This volume contains the papers presented at the 23^{rd} Canadian Conference on Computational Geometry (CCCG'11), held in Toronto on August 10-12, 2011, in a special collaboration with the Fields Institute. These papers are also available electronically at http://www.cccg.ca and at http://2011.cccg.ca.

We are grateful to the Program Committee for agreeing to a rigorous review process. They, and other reviewers, thoroughly examined all submissions and provided excellent feedback.

Out of 105 papers submitted, 83 were accepted. We thank the authors of all submitted papers, all those who have registered, and in particular Noga Alon, William Steiger and Emo Welzl for presenting plenary lectures.

We also thank those who provided valuable information that helped with this year's organization. Organizers from the past ten years, and in particular Stephane Durocher and Jason Morrison from last year, answered many questions. The same can be said for several friends and colleagues who happened to be located near the organizers during critical decision times. Included in this group are Sébastien Collette, who also prepared these proceedings, as well as Perouz Taslakian and Narbeh Bedrossian who designed the cover and conference logo.

Last but not least, we are grateful for sponsorship from the *Fields Institute*, *AARMS*, and the *Mprime Network* (formerly MITACS). Their financial support has helped us to cover many costs and waive the registration fees for over 90 students and postdocs. Fields also permitted us to rely on their ever helpful staff (in particular Alison Conway, Natalie Dytyniak, and Andrea Yeomans) for planning, registration and on-site assistance.

Greg Aloupis and David Bremner (Conference Organizers)



Copyrights of the articles in these proceedings are maintained by their respective authors. More information about this conference and about previous and future editions is available online at

http://cccg.ca

Invited Speakers

- Noga Alon (Tel Aviv U.) Erdős memorial lecture
- William Steiger (Rutgers U.)
- Emo Welzl (ETH Zurich)

Organizing Committee

- Greg Aloupis (U. Libre de Bruxelles)
- David Bremner (U. New Brunswick)

Program Committee

- Greg Aloupis (U. Libre de Bruxelles)
- Binay Bhattacharya (Simon Fraser U.)
- Therese Biedl (U. Waterloo)
- Prosenjit Bose (Carleton U.)
- David Bremner (U. New Brunswick)
- Paz Carmi (Ben-Gurion University of the Negev)
- Timothy Chan (U. Waterloo)
- Sébastien Collette (U. Libre de Bruxelles)
- Mirela Damian (Villanova U.)
- Erik D. Demaine (MIT)

- Antoine Deza (McMaster U.)
- Vida Dujmović (Carleton U.)
- Adrian Dumitrescu (U. Wisconsin-Milwaukee)
- Stephane Durocher (U. Manitoba)
- Will Evans (U. British Columbia)
- Joachim Gudmundsson (National ICT, Australia)
- Maarten Löffler (U. California, Irvine)
- Anna Lubiw (U. Waterloo)
- Henk Meijer (Roosevelt Academy)
- Jason Morrison (U. Manitoba)
- Asish Mukhopadhyay (U. Windsor)
- Sheung-Hung Poon (National Tsing Hua U.)
- Michiel Smid (Carleton U.)
- Jack Snoeyink (UNC Chapel Hill)
- Diane Souvaine (Tufts U.)
- Godfried Toussaint (McGill U.)
- Jorge Urrutia (U. Nacional Autónoma de México)
- Marc van Kreveld (Utrecht U.)
- Caoan Wang (Memorial U. Newfoundland)

Other Reviewers

- Md. Shafiul Alam
- Olivier Devillers
- Anne Driemel
- Muriel Dulieu
- Fabrizio Frati
- Minghui Jiang
- Marcin Kaminski
- Matya Katz
- Matias Korman
- Stefan Langerman
- Chung-Shou Liao
- Dave Millman
- Pat Morin
- Satish Chandra Panigrahi
- Suneeta Ramaswami

- $\bullet\,$ Noushin Saeedi
- Maria Saumell
- Marc Scherfenberg
- Rodrigo Silveira
- Matthew Skala
- Darren Strash
- Shin-Ichi Tanigawa
- Perouz Taslakian
- Lowell Trott
- Mikael Vejdemo-Johansson
- Antoine Vigneron
- David R. Wood
- Stefanie Wuhrer
- $\bullet\,$ Ke Yi
- Norbert Zeh

Contents

Invited speaker 1 - Wednesday Aug.10 9:40-10:40	13
Geometric Partitioning William Steiger, Rutgers University	13
Session 1 α - Wednesday Aug.10 11:00-12:00	14
Convex blocking and partial orders on the plane Canek Peláez, José Miguel Díaz-Báñez, Marco A. Heredia, J. Antoni Sellarès, Jorge Urrutia and Inmaculada Ventura	15
On k-Gons and k-Holes in Point Sets Birgit Vogtenhuber, Oswin Aichholzer, Ruy Fabila-Monroy, Clemens Huemer, Jorge Urrutia, Marco A. Heredia, Hernan Gonzalez-Aguilar, Thomas Hackl and Pavel Valtr	21
Hardness Results for Two-Dimensional Curvature-Constrained Motion Planning David Kirkpatrick, Irina Kostitsyna and Valentin Polishchuk	27
Session 1 eta - Wednesday Aug.10 11:00-12:00	33
Optimizing Budget Allocation in Graphs Boaz Benmoshe, Eran Omri and Michael Elkin	33
Bottleneck Steiner Tree with Bounded Number of Steiner Vertices A. Karim Abu-Affash, Paz Carmi and Matthew Katz	39
Connecting Two Trees with Optimal Routing Cost Mong-Jen Kao, Bastian Katz, Marcus Krug, Der-Tsai Lee, Martin Nöllenburg, Ignaz Rutter and Dorothea Wagner	43
Session 1 γ - Wednesday Aug.10 11:00-12:00	48
Minimum Many to Many Matchings for Computing the Distance Between Two Sequences David Rappaport, Godfried Toussaint and Mustafa Mohamad	49
Staying Close to a Curve Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz and Hamid Zarrabi-Zadeh	55
Isotopic Frchet Distance Erin Chambers, David Letscher, Tao Ju and Lu Liu	59
Session 2 α - Wednesday Aug.10 13:30-15:10	65
Edge Unfoldings of Platonic Solids Never Overlap Takashi Horiyama and Wataru Shoji	65
Development of Curves on Polyhedra via Conical Existence Joseph O'Rourke and Costin Vilcu	71
Common Developments of Several Different Orthogonal Boxes Zachary Abel, Erik Demaine, Martin Demaine, Hiroaki Matsui, Günter Rote and Ryuhei Uehara	77
Edge-Unfolding Orthogonal Polyhedra is Strongly NP-Complete Zachary Abel and Erik D. Demaine	83

A Topologically Convex Vertex-Ununfoldable Polyhedron Zachary Abel, Erik D. Demaine and Martin L. Demaine	89
Session 2 β - Wednesday Aug.10 13:30-15:10	92
Isoperimetric Triangular Enclosure with a Fixed Angle Prosenjit Bose and Jean-Lou De Carufel	93
Robust approximate assembly partitioning Elisha Sacks, Victor Milenkovic and Yujun Wu	99
Approximation Algorithms for a Triangle Enclosure Problem Karim Douieb, Matthew Eastman, Anil Maheshwari and Michiel Smid	105
Finding the Maximum Area Parallelogram in a Convex Polygon Kai Jin and Kevin Matulef	111
Illumination problems on translation surfaces with planar infinities Nikolay Dimitrov	117
Session 2 γ - Wednesday Aug.10 13:30-15:10	123
Detecting VLSI Layout and Connectivity Errors in a Query Window Ananda Swarup Das, Prosenjit Gupta and Kannan Srinathan	123
Finding Maximum Density Axes Parallel Regions for Weighted Point Sets Ananda Swarup Das, Prosenjit Gupta, Kannan Srinathan and Kishore Kothapalli	129
Bichromatic Line Segment Intersection Counting in $O(n \text{ sqrt}(\log n))$ Time Timothy M. Chan and Bryan T. Wilkinson	135
Sequential Dependency Computation via Geometric Data Structures Gruia Calinescu and Howard Karloff	141
Point Location in Well-Shaped Meshes Using Jump-and-Walk Jean-Lou De Carufel, Craig Dillabaugh and Anil Maheshwari	147
Open Problems Session - Wednesday Aug.10 16:00-17:00	153
Open Problems from CCCG 2010 Erik D. Demaine and Joseph O'Rourke	153
Session 3 $lpha$ - Thursday Aug.11 9:00-10:20	157
Where and How Chew's Second Delaunay Refinement Algorithm Works Alexander Rand	157
Probabilistic Bounds on the Length of a Longest Edge in Delaunay Graphs of Random Points in d-Dimensions	
Esther M. Arkin, Antonio Fernandez Anta, Joseph S. B. Mitchell and Miguel A. Mosteiro	163
Outerplanar graphs and Delaunay triangulations Md. Ashraful Alam, Igor Rivin and Ileana Streinu	169
Toward the Tight Bound of the Stretch Factor of Delaunay Triangulations Ge Xia and Liang Zhang	175

Session 3 eta - Thursday Aug.11 9:00-10:20	181
Rigid components in fixed-lattice and cone frameworks Matthew Berardi, Brent Heeringa, Justin Malestein and Louis Theran	181
Orientations of Simplices Determined by Orderings on the Coordinates of their Vertices Emeric Gioan, Kevin Sol and Gérard Subsol	187
Pushing the boundaries of polytopal realizability David Bremner, Antoine Deza, William Hua and Lars Schewe	193
On the generation of topological (n_k) -configurations Jürgen Bokowski and Vincent Pilaud	199
Session 3 γ - Thursday Aug.11 9:00-10:20	205
Sliding labels for dynamic point labeling Andreas Gemsa, Martin Nöllenburg and Ignaz Rutter	205
A Discrete and Dynamic Version of Klee's Measure Problem Hakan Yildiz, John Hershberger and Subhash Suri	211
Kinetically-aware Conformational Distances in Molecular Dynamics Chen Gu, Xiaoye Jiang and Leonidas Guibas	217
Collinearities in Kinetic Point Sets Benjamin Lund, George Purdy, Justin Smith and Csaba Toth	223
Session 4 $lpha$ - Thursday Aug.11 10:50-11:50	228
Convexifying Polygons Without Losing Visibilities Oswin Aichholzer, Greg Aloupis, Erik D. Demaine, Martin L. Demaine, Vida Dujmovic, Ferran Hurtado, Anna Lubiw, Günter Rote, André Schulz, Diane L. Souvaine and Andrew Winslow	229
Expansive Motions for d-Dimensional Open Chains Sarah Eisenstat and Erik D. Demaine	235
Making triangulations 4-connected using flips Prosenjit Bose, Dana Jansens, André Van Renssen, Maria Saumell and Sander Verdonschot	241
Session 4 eta - Thursday Aug.11 10:50-11:50	248
Approximating the Medial Axis by Shooting Rays: 3D Case Svetlana Stolpner, Kaleem Siddiqi and Sue Whitesides	249
An Incremental Algorithm for High Order Maximum Voronoi Diagram Construction Khuong Vu and Rong Zheng	255
Approximating a Motorcycle Graph by a Straight Skeleton Stefan Huber and Martin Held	261
Session 4 γ - Thursday Aug.11 10:50-11:50	267
Small Octahedral Systems Grant Custard, Antoine Deza, Tamon Stephen and Feng Xie	267
Combinatorics of Minkowski decomposition of associahedra Carsten Lange	273

A Fourier-Theoretic Approach for Inferring Symmetries Xiaoye Jiang, Jian Sun and Leonidas Guibas	279
Invited speaker 2 - Thursday Aug.11 13:30-14:30	285
List coloring and Euclidean Ramsey Theory Noga Alon, Tel Aviv University	285
Session 5 α - Thursday Aug.11 16:05-17:25	287
Rigidity-Theoretic Constructions of Integral Fary Embeddings <i>Timothy Sun</i>	287
Drawing some planar graphs with integer edge-lengths Therese Biedl	291
Approximating the Obstacle Number for a Graph Drawing Efficiently Deniz Sarioz	297
A Note on Minimum-Segment Drawings of Planar Graphs Stephane Durocher, Debajyoti Mondal, Rahnuma Islam Nishat and Sue Whitesides	303
Session 5 β - Thursday Aug.11 16:05-17:25	309
Characterization of Shortest Paths on Directional Frictional Polyhedral Surfaces Gutemberg Guerra Filho and Pedro J. De Rezende	309
Memory-Constrained Algorithms for Shortest Path Problem Tetsuo Asano and Benjamin Doerr	315
Finding Optimal Geodesic Bridges Between Two Simple Polygons Amit Bhosle and Teofilo Gonzalez	319
Approximating Geodesic Distances on 2-Manifolds in R^3 Christian Scheffer and Jan Vahrenhold	325
Session 5 γ - Thursday Aug.11 16:05-17:25	331
An In-Place Priority Search Tree Minati De, Anil Maheshwari, Subhas Nandy and Michiel Smid	331
Orthogonal Range Search using a Distributed Computing Model Pouya Bisadi and Bradford Nickerson	337
On Finding Skyline Points for Range Queries in Plane Anil Kishore Kalavagattu, Ananda Swarup Das, Kishore Kothapalli and Kannan Srinathan	343
Space-efficient Algorithms for Empty Space Recognition among a Point Set in 2D and 3D <i>Minati De and Subhas Nandy</i>	347
Session 6 α - Friday Aug.12 9:00-10:20	354
Realizing Site Permutations Stephane Durocher, Saeed Mehrabi, Debajyoti Mondal and Matthew Skala	355
Establishing Strong Connectivity using Optimal Radius Half-Disk Antennas Greg Aloupis, Mirela Damian, Robin Flatland, Matias Korman, Ozgur Ozkan, David Rappaport and Stefanie Wuhrer	361

Euclidean Movement Minimization Mohammadamin Fazli, Mohammadali Safari, Nima Anari, Pooya Jalaly Khalilabadi and Mohammad Ghodsi	367
A Randomly Embedded Random Graph is Not a Spanner Abbas Mehrabian	373
Session 6 β - Friday Aug.12 9:00-10:20	375
Approximation Algorithms for the Discrete Piercing Set Problem for Unit Disks <i>Minati De, Gautam Das and Subhas Nandy</i>	375
New Lower Bounds for the Three-dimensional Orthogonal Bin Packing Problem Chia-Hong Hsu and Chung-Shou Liao	381
The 22 Simple Packing Problem André Van Renssen and Bettina Speckmann	387
On covering of any point configuration by disjoint unit disks Yosuke Okayama, Masashi Kiyomi and Ryuhei Uehara	393
Session 6 γ - Friday Aug.12 9:00-10:20	398
Improving Accuracy of GNSS Devices in Urban Canyons Boaz Ben-Moshe, Elazar Elkin, Harel Levi and Ayal Weissman	399
Geometry-Free Polygon Splitting Sherif Ghali	405
Robustness of topology of digital images and point clouds <i>Peter Saveliev</i>	411
Planar Pixelations and Shape Reconstruction Brandon Rowekamp	417
Invited speaker 3 - Friday Aug.12 11:00-12:00	423
Counting Simple Polygonizations of Planar Point Sets <i>Emo Welzl, ETH Zurich</i>	423
Session 7 α - Friday Aug.12 13:30-15:10	424
Algorithms for Bivariate Majority Depth Dan Chen and Pat Morin	425
Exact Algorithms and APX-Hardness Results for Geometric Set Cover Elyot Grant and Timothy Chan	431
Enumerating Minimal Transversals of Geometric Hypergraphs Khaled Elbassioni, Imran Rauf and Saurabh Ray	437
Helly Numbers of Polyominoes Jean Cardinal, Hiro Ito, Matias Korman and Stefan Langerman	443
Session 7 β - Friday Aug.12 13:30-15:10	449
Open Guard Edges and Edge Guards in Simple Polygons Csaba Toth, Godfried Toussaint and Andrew Winslow	449

Computing k-Link Visibility Polygons in Environments with a Reflective Edge Salma Sadat Mahdavi, Ali Mohades and Bahram Kouhestani	455
Edge-guarding Orthogonal Polyhedra Giovanni Viglietta, Nadia M. Benbernou, Erik D. Demaine, Martin L. Demaine, Anastasia Kurd Joseph O'Rourke, Godfried Toussaint and Jorge Urrutia	lia, 461
Wireless Localization within Orthogonal Polyhedra Tobias Christ and Michael Hoffmann	467
Weak Visibility Queries in Simple Polygons Mojtaba Nouri Bygi and Mohammad Ghodsi	473
Session 7 γ - Friday Aug.12 13:30-15:10	479
The Possible Hull of Imprecise Points Jeff Sember and William Evans	479
A Slow Algorithm for Computing the Gabriel Graph with Double Precision David L. Millman and Vishal Verma	485
An Experimental Analysis of Floating-Point Versus Exact Arithmetic Martin Held and Willi Mann	489
On Inducing n-gons Marjan Abedin, Ali Mohades and Marzieh Eskandari	495
Weak Matching Points with Triangles Fatemeh Panahi, Ali Mohades, Mansoor Davoodi and Marzieh Eskandari	501

Geometric Partitioning

William Steiger *

The well-known Ham Sandwich Theorem says that given d nice sets in \mathbb{R}^d , there exists a hyperplane that splits each of them into two parts of equal measure. When the sets are finite, there is the computational problem of finding such a plane.

This is a starting point for other facts about when, and how various sets can or cannot be split in various ways. Several old and new geometric partitioning results of this kind will be discussed.

^{*}Department of Computer Science, Rutgers University, NJ, USA. steiger@cs.rutgers.edu

Convex blocking and partial orders on the plane¹

José Miguel Díaz-Báñez*

Marco A. Heredia[†]

leredia[†] Canek Peláez[†] Inmaculada Ventura^{*} J. Antoni Sellarès[‡]

Jorge Urrutia[§]

Abstract

Let $C = \{c_1, \ldots, c_n\}$ be a collection of disjoint closed convex sets in the plane. Suppose that one of them, say c_1 , represents a valuable object we want to uncover, and we are allowed to pick a direction $\alpha \in [0, 2\pi)$ along which we can translate (remove) the elements of C one at a time while avoiding collisions. In this paper we find an $O(n^2 \log n)$ time algorithm that finds a direction α that minimizes the number of elements of C that have to be removed before we can reach c_1 .

1 Introduction

Consider a set $C = \{c_1, \ldots, c_n\}$ of pairwise disjoint closed bounded convex sets, and a direction $\alpha \in [0, 2\pi)$; e.g., the vertical upwards direction. It is well known that the elements of C can be translated (removed) one at a time by moving them upwards while avoiding collisions with other elements of C [7, 10]. Suppose that c_1 is a special object that we want to uncover, and that we are allowed to choose a direction α along which we can remove the elements of C one at a time while avoiding collisions.

We want to find the direction α that minimizes the number of elements we need to remove before we reach c_1 . In Figure 1, it is easy to see that if we remove the elements of C in the direction α_2 , four elements of Chave to be removed before we reach c_1 , while for α_1 we only need to remove two.

This problem can be seen as a variant of the problem known in computational geometry as the *separability problem* [2, 5, 9]. It is also related to *spherical orders* determined by light obstructions [6].

In this paper we present an $O(n^2 \log n)$ time algorithm to solve this problem, assuming that for every pair of elements of C we can compute a tangent line to



Figure 1: A set C of convex sets.

both of them in constant time.

2 Preliminaries

Let X be a finite set, and < a relation on the elements of X that satisfies the following conditions: (a) If x < yand y < z then x < z (transitivity), and (b) $x \not < x$ (antireflexivity). The set X together with < is called a partial order, and it is usually denoted as P(X, <).

Given $x, y \in X$, we say that y covers x if x < y and there is no element $w \in X$ such that x < w < y. The diagram of P(X, <) is the directed graph whose vertices are the elements of X and there is an oriented edge from x to y if y covers x. We say that the diagram of P(X, <)is planar if it can be drawn on the plane in such a way that the elements of X are represented by points on the plane, no edges of G intersect, except perhaps at a common endpoint, and if y is a cover of x, then they are joined by a monotonically increasing oriented edge from x to y (in the vertical direction).

Given two elements $x, y \in X$, a supremum of them is an element $w \in X$ such that x < w, y < w, and for any other element $z \in X$ such that x < z and y < z we have that w < z. An *infimum* is defined in a similar way, except that we require w to be w < x and w < y. An ordered set is called a *lattice* if any two elements have a unique supremum and infimum. A lattice is called a *planar lattice* if its diagram is planar. Finally, a finite order P(<, X) is called a *truncated planar lattice* if by

^{*}Departamento Matemática Aplicada II, Universidad de Sevilla, Spain.

[†]Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Mexico.

[‡]Institut d'Informàtica i Matemàtica Aplicada, Universitat de Girona, Spain.

[§]Instituto de Matemáticas, Universidad Nacional Autónoma de México, Mexico.

¹A shorter version of this paper appeared in the XIV Spanish Meeting on Computational Geometry held June 27-30, 2011. Corresponding author: canek@ciencias.unam.mx

adding to $P(\langle X)$ both a least and a greatest element the resulting order is a planar lattice.

Let $C = \{c_1, \ldots, c_n\}$ be a set of disjoint closed convex sets on the plane. Given two convex sets c_i and c_j in C, we say that c_j is an *upper cover* of c_i in the direction α (for short, an α -cover) if the following conditions are satisfied:

- 1. There is at least one directed line segment with direction α starting at a point in c_i and ending at a point in c_j .
- 2. Any directed line segment with direction α starting at a point in c_i and ending at a point in c_j does not intersect any other element of C.

Observe that if c_j is an α -cover of c_i , then c_i is an $(\alpha + \pi)$ -cover of c_j . We say that c_j blocks c_i in the direction α , written as $c_i \prec_{\alpha} c_j$, if there is a sequence $c_i = c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(k)} = c_j$ of elements of C such that $c_{\sigma(r+1)}$ is an α -cover of $c_{\sigma(r)}, r = 1, \ldots, k-1$ (Figure 2).



Figure 2: c_j is an α -cover of $c_{\sigma(3)}$ and $c_i \prec_{\alpha} c_j$.

Clearly if $c_i \prec_{\alpha} c_j$ and $c_j \prec_{\alpha} c_k$, then $c_i \prec_{\alpha} c_k$, and thus C together with the blocking relation \prec_{α} is a partial order on C, which we will denote as $P(\prec_{\alpha}, C)$. It is known that $P(\prec_{\alpha}, C)$ is a truncated planar lattice [10].

The diagram of such truncated lattice has the elements of C as vertices and there is an oriented edge from c_i to c_j if c_j is an α -cover of c_i (Figure 3). The elements of C that we need to remove in the α direction before an element c_i of C is reached are those convex sets c_j such that $c_i \prec_{\alpha} c_j$, the set containing these elements will be called the α -upper set of c_i , or for short, the α -up-set of c_i in α . Thus our problem reduces to that of finding the direction α such that the cardinality of the α -up-set of c_1 is minimized.

Observe that as α changes, so does $P(\prec_{\alpha}, C)$. In fact, it is easy to find families of convex sets for which $P(\prec_{\alpha}, C)$ changes a quadratic number of times. We proceed now to prove some properties of $P(\prec_{\alpha}, C)$, $0 \leq \alpha < 2\pi$ which will allow us to find an α such that the α -up-set of c_1 has minimum cardinality in $O(n^2 \log n)$ time.

Given a convex set c, a line ℓ is called a supporting line of c if it intersects c, and c is contained in one of the closed half planes determined by ℓ . Given two convex sets c_i and c_j , a line ℓ is called an internal tangent of them if ℓ supports them, and c_i is contained in one of the closed half planes determined by ℓ , and c_j in the other. A set of directions I is called an interval, if there are $\alpha, \beta \leq 2\pi$ such that the elements of I are angles of the form $\alpha + \delta$, $0 \leq \delta \leq \beta$, addition taken mod 2π .



Figure 3: Diagram of $P(\prec_{\alpha}, C)$ for $\alpha = \pi/2$.

Lemma 1 Let c_i and c_j be two convex sets in C. The set of directions in which c_j blocks c_i is a non-empty interval $\mathcal{I}_{i,j}$.

Proof. Clearly a direction in which c_j does not block c_i always exists. Without loss of generality we will assume that such direction is 0.

Let θ_1 be the first direction greater than 0 where $c_i \prec_{\theta_1} c_j$: Such θ_1 exists because c_j always blocks c_i in a set of directions enclosed by the two internal tangents defined by c_i and c_j .

Let θ_2 be the *last* direction greater than θ_1 such that for any $\gamma \in [\theta_1, \theta_2] c_i \prec_{\gamma} c_j$. If there is no other direction $\gamma \in [\theta_2, 2\pi]$ where $c_i \prec_{\gamma} c_j$ then our result holds. Suppose then that there are θ_3 and θ_4 such that i) $\theta_2 < \theta_3$, ii) $\theta_3 < \theta_4 < 2\pi$, and for $\gamma \in [\theta_3, \theta_4]$, $c_i \prec_{\gamma} c_j$, and iii) for any $\gamma \in [\theta_2, \theta_3]$, $c_i \not\prec_{\gamma} c_j$, (Figure 4).

Clearly $\theta_3 - \theta_2 < \pi$, or $\theta_1 - \theta_4 < \pi$, where the second angle is taken modulo 2π . Assume without loss of generality that $\theta_3 - \theta_2 < \pi$, and that $0 < \theta_2 < \frac{\pi}{2} < \theta_3$, for otherwise we can rotate *C* appropriately until this condition holds.

Let $\gamma = \frac{\pi}{2}$, then $c_i \not\prec_{\gamma} c_j$. Since $c_i \prec_{\theta_2} c_j$, we know that there is a sequence $c_i = c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(k)} = c_j$ such that $c_{\sigma(r+1)}$ is a θ_2 -cover of $c_{\sigma(r)}$ for $r = 1, \ldots, k-1$. For the same reason, there is a sequence $c_i = c_{\omega(1)}, c_{\omega(2)}, \ldots, c_{\omega(m)} = c_j$ such that $c_{\omega(r+1)}$ is a θ_3 cover of $c_{\omega(r)}$ for $r = 1, \ldots, m-1$. The two sequences differ in at least one element, otherwise our gap would not exist.

Let ℓ_1 and ℓ_2 be the supporting lines of c_i in the γ direction: Since $c_i \not\prec_{\gamma} c_j$, c_j cannot intersect the interior



Figure 4: We can assume that $\theta_3 - \theta_2 < \pi$.

of the strip bounded by ℓ_1 and ℓ_2 . Suppose first that c_j is to the left of this strip (Figure 5).



Figure 5: c_i to the left of ℓ_1 and ℓ_2 .

Since $c_{\sigma(2)}$ is a θ_2 -cover of $c_i = c_{\sigma(1)}$, there is a line segment parallel to the direction θ_2 with endpoints in $c_i = c_{\sigma(1)}$ and $c_{\sigma(2)}$. Similarly for $c_{\sigma(r)}$ and $c_{\sigma(r+1)}$, there is a line segment parallel to the direction θ_2 with endpoints in $c_{\sigma(r)}$ and $c_{\sigma(r+1)}$, $r \in \{2, \ldots, k-1\}$. Each $c_{\sigma(r)}$, $r \in \{2, \ldots, k-1\}$, contains two endpoints from two of this segments, and this endpoints can be joined with a line segment totally contained in $c_{\sigma(r)}$.

This forms a connected curve that starts in c_i and ends in c_j , passing through all the elements of the sequence. This curve consist of two types of line segments: Those parallel to the θ_2 direction, and those completely contained in the elements of the sequences. But $\theta_2 < \gamma$, so the first type always goes to the right. And the second type may go to the left, but contained in an element of the sequence (Figure 6).

The only way such a curve exists, is if at least one element in $\{c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(k)}\}$ intersects the strip



Figure 6: A sequence of θ_2 -covers from c_j to c_i , and the curve that passes through the elements of it.

bounded by ℓ_1 and ℓ_2 , which implies that $c_i \prec_{\gamma} c_j$, a contradiction!

If we suppose that c_j is to the right of ℓ_2 , a contradiction arises, but using the sequence $c_i = c_{\omega(1)}, c_{\omega(2)}, \ldots, c_{\omega(m)} = c_j$ in the θ_3 direction. Therefore, the directions where c_j blocks c_i form a non-empty interval.

It follows from the proof of Lemma 1 that the endpoints of the intervals $\mathcal{I}_{i,j}$ are defined by the internal tangents of pairs of elements in C. The next observation follows:

Observation 1 There are at most $4\binom{n}{2}$ combinatorially distinct values of α where $P(\prec_{\alpha}, C)$ may change; these changes occur in slopes generated by internal tangents between pairs of elements of C.

We can then reduce the search space for α_0 to the set $\mathcal{D} = \{\gamma_1, \ldots, \gamma_{4\binom{n}{2}}\}$ containing these directions. For the sake of clarity, we are supposing that no two internal tangents are parallel and that the elements of \mathcal{D} are ordered as $\gamma_i < \gamma_j$ if i < j.

We observe that \mathcal{D} can be calculated in $O(n^2 \log n)$ if we suppose that the internal tangents between any two convex sets in C can be determined in constant time. For each γ_k we can store the indexes i, j of the convex sets that define the internal tangent.

3 α -triangulations

Our problem can be solved by calculating the truncated lattices $P(\prec_{\gamma_i}, C)$ for every direction $\gamma_i \in \mathcal{D}$, and then obtaining the γ_i -up-set of c_1 in each one of them. Selecting a $\gamma_i \in \mathcal{D}$ which produces a smallest γ_i -up-set yields an optimal solution. Since calculating the truncated lattice has a cost of $O(n \log n)$ time for each of the $4\binom{n}{2}$ directions in \mathcal{D} [10], this results in an $O(n^3 \log n)$ -time algorithm to solve our problem.

To improve this complexity, we will show that we need to calculate from scratch only one truncated lattice. For the remaining directions of \mathcal{D} the corresponding truncated lattice (more precisely, the α -triangulation, to be described shortly) can be updated in constant time.

For each direction $\alpha \in [0, 2\pi)$, we extend $P(\prec_{\alpha}, C)$ to a planar lattice $P'(\prec_{\alpha}, C)$ by adding two special vertices, a source s and a sink t, i.e. for each $c_i \in C$, $s \prec_{\alpha} c_i \prec_{\alpha} t$. For a fixed direction we can picture t as a very large convex set standing above all of C, and s as a very large convex set standing below all of C (Figure 7).



Figure 7: The lattice $P'(\prec_{\alpha}, C)$ for $\alpha = \pi/2$.

For each α , we now extend $P'(\prec_{\alpha}, C)$ to a triangulation \mathcal{T}_{α} , that is, a planar graph where every internal face is a triangle, which we will call an α -triangulation, by adding oriented edges avoiding creating oriented cycles (Figure 8).

By Observation 1 there are at most $4\binom{n}{2}$ triangulations, and we want to know how \mathcal{T}_{α} changes as α goes from γ_k to γ_{k+1} . We remark that there are cases when the triangulations \mathcal{T}_{γ_k} and $\mathcal{T}_{\gamma_{k+1}}$ are the same (Figure 9).

The next observation will be used:

Observation 2 Let α, β be such that $P(\prec_{\alpha}, C) \neq P(\prec_{\beta}, C)$, then there is at least one pair of elements $c_i, c_j \in C$ such that c_j is an α -cover of c_i in $P(\prec_{\alpha}, C)$, and it is not a β -cover of c_i in $P(\prec_{\beta}, C)$, or vice versa; that is, the set of edges of the diagram of $P(\prec_{\alpha}, C)$ is different from the set of edges of the diagram of $P(\prec_{\beta}, C)$. Moreover, if $\alpha, \beta \in D$, then c_i and c_j define α or β .



Figure 8: The triangulation \mathcal{T}_{α} for $\alpha = \frac{\pi}{2}$.



Figure 9: The triangulations \mathcal{T}_{γ_k} and $\mathcal{T}_{\gamma_{k+1}}$ are the same, since the partial order does not change.

It turns out that the difference between the \mathcal{T}_{γ_k} and $\mathcal{T}_{\gamma_{k+1}}$ triangulations is an arc flip, as defined in [8]:

Lemma 2 Given the triangulation \mathcal{T}_{γ_k} , the triangulation $\mathcal{T}_{\gamma_{k+1}}$ can be obtained from \mathcal{T}_{γ_k} (if they are different) by flipping an arc in \mathcal{T}_{γ_k} . Such an arc flip either adds or removes an arc between the convex sets c_i and c_i that define γ_{k+1} .

Proof. Suppose that $P(\prec_{\gamma_k}, C)$ and $P(\prec_{\gamma_{k+1}}, C)$ are different. By Observation 2 two cases arise:

- 1. There are two elements c_i and c_j of C that generate γ_k such that c_j is a γ_k -cover of c_i , but it is not a γ_{k+1} -cover of c_i .
- 2. The elements c_i and c_j that generate γ_{k+1} become comparable in $P(\prec_{\gamma_{k+1}}, C)$, and one of them, say c_j is a γ_{k+1} -cover of c_i .

In case 1 when we flip the edge connecting c_i to c_j in \mathcal{T}_{γ_k} they become not comparable in $\mathcal{T}_{\gamma_{k+1}}$. Furthermore

it is easy to see that there is a line parallel to the γ_{k+1} direction that separates c_i from c_j , and that this line intersects two elements of $C \cup \{s,t\} - \{c_i,c_j\}$. This are the two vertices that become adjacent as we flip the edge connecting c_i and c_j . Thus $\mathcal{T}_{\gamma_{k+1}}$ can be obtained from \mathcal{T}_{γ_k} in constant time.

In the second case the inverse occurs.

In Figure 10 and Figure 11 we can see an example of the arc flip performed in the proof of Lemma 2.



Figure 10: The arc $c_i \rightarrow c_j$ before flipping.



Figure 11: The arc $c_a \rightarrow c_b$ after flipping.

4 An $O(n^2 \log n)$ algorithm to find α_0

Theorem 3 Finding α_0 can be done in $O(n^2 \log n)$.

To prove Theorem 3 we need the following result:

Lemma 4 For any element c_i , as we go from γ_1 to $\gamma_{4\binom{n}{2}}$, the up-set of c_i changes O(n) times.

Proof. By Lemma 1, the set of directions for which c_j blocks c_i is an interval $\mathcal{I}_{i,j}$. This means that for each $c_j \neq c_i$ in C, as we go from γ_1 to $\gamma_{4\binom{n}{2}}$, c_j starts to block c_i once and stops blocking it once. Therefore the up-set of c_i changes a linear number of times, that is any $c_j \in C$ enters and exits it once.

We proceed now with the proof of Theorem 3.

Proof. We first generate the set \mathcal{D} of critical directions in $O(n^2)$ time. Observe that this can be done in quadratic time since we are assuming that the tangents generated by two elements of C can be calculated in constant time. Next we sort the elements of \mathcal{D} in $O(n^2 \log n)$ time. When we store each $\gamma_i \in \mathcal{D}$ we also store the elements of C that generate it. Next we construct \mathcal{T}_{γ_1} in $O(n \log n)$ time, coloring the vertices of the triangulation as follows:

- We color red the elements of C in the γ_1 -up-set of c_1 , including c_1 .
- We color blue the remaining elements of C.

At this stage, we also calculate the number of incoming arcs to each c_i whose initial vertex is blue, or red. Such a coloring can be done in O(n) time. Let c_i and c_j be the elements that generated γ_{k+1} . It is easy to check that if c_j was not a γ_k -cover of c_i or vice versa, then $P(\prec_{\gamma_k}, C) = P(\prec_{\gamma_{k+1}}, C)$ and the up-set of c_1 does not change. Suppose then that c_j was a γ_k -cover of c_i . By Lemma 2, $P(\prec_{\gamma_k}, C) \neq P(\prec_{\gamma_{k+1}}, C)$ and we can obtain $\mathcal{T}_{\gamma_{k+1}}$ from \mathcal{T}_{γ_k} in constant time. The crucial part of our procedure is how to update the up-set of c_1 .

Suppose first that the elements c_i and c_j that determine γ_{k+1} are different from c_1 .

Several cases arise.

- 1. $c_1 \prec_{\gamma_k} c_i$, $c_1 \prec_{\gamma_k} c_j$, and c_i is not comparable to c_j in $P(\prec_{\gamma_k}, C)$, but c_i is comparable to c_j in $P(\prec_{\gamma_{k+1}}, C)$. In this case, the up-set of c_1 remains unchanged.
- 2. $c_1 \prec_{\gamma_k} c_i$, $c_1 \prec_{\gamma_k} c_j$, and c_i is comparable to c_j in $P(\prec_{\gamma_k}, C)$, but c_i is not comparable to c_j in $P(\prec_{\gamma_{k+1}}, C)$. In this case the up-set of c_1 may change. Suppose that c_j is a γ_k -cover of c_i . Observe that c_i remains in the up-set of c_1 , but c_j may not belong to it anymore. In this case the arc from c_i to c_j is flipped. If at least one arc from a red element c_r to c_j remains then c_j remains in the up-set of c_1 , otherwise the up-set of c_1 changes, and is recalculated in linear time.

- 3. c_i and c_j do not belong to the up-set of c_1 . In this case, the up-set of c_1 does not change.
- 4. $c_i \prec_{\gamma_k} c_j$ and c_i is not in the up-set of c_1 in $P(\prec_{\gamma_k}, C)$. The up-set of c_1 remains unchanged in $P(\prec_{\gamma_{k+1}}, C)$.
- 5. $c_i \not\prec_{\gamma_k} c_j, c_j$ is not in the up-set of c_1 , and c_i belongs to the up-set of c_1 . In this case, c_j is an γ_{k+1} -cover of c_i and the up-set of c_1 changes. Therefore we must recalculate the up-set of c_1 .

Each time we recalculate the up-set of c_1 , we also recalculate for each c_i the number of incoming arcs starting at a blue or red point.

A similar case analysis is carried out when $c_i = c_1$, the details are left to the reader. By Lemma 4, we have to update the up-set of c_1 only a linear number of times, and thus the whole process takes $O(n^2 \log n)$ time. This proves Theorem 3.

5 Conclusions

In this paper we studied a variant of the classic separability problem. Given a set $C = \{c_1, \ldots, c_n\}$ of pairwise disjoint closed convex sets, find a direction α minimizing the number of elements of C that have to be removed, in the direction α , before we reach a particular element $c_1 \in C$. We presented an $O(n^2 \log n)$ -time algorithm to solve this problem, under the assumption that the internal tangents between any two sets of C can be calculated in constant time: For example, this is the case for circles and ellipses, convex polygons with a constant number of sides, and shapes defined by a constant number of Bézier curves.

We suspect that the complexity of our problem is $\Omega(n^2 \log n)$. In particular any approach that has to sort the elements of \mathcal{D} may in general take $O(n^2 \log n)$ time unless some extra restrictions on the elements of C are imposed. For example for circles, we can sort the slopes generated by them in quadratic time (using the dual space), improving the complexity of our algorithm to $O(n^2)$. The details of this will be given in the full version of this paper.

It is easy to see that if we want to calculate for each $c_i \in C$ the number of elements that have to be removed before we can remove c_i , this can be done in $O(n^3)$.

6 Acknowledgments

Research of José Miguel Díaz-Báñez and Inmaculada Ventura partially supported by Spanish Government under Project MEC MTM2009-08652; research of Marco A. Heredia and Canek Peláez partially supported by CONACYT of Mexico; research of J. Antoni Sellarès partially supported by the Spanish MCI grant TIN2010-20590-C02-02; and research of Jorge Urrutia partially supported by SEP-CONACYT of Mexico, Proyecto 80268, and by Spanish Government under Project MEC MTM2009-08652.

References

- M. Abellanas, S. Bereg, F. Hurtado, A. García Olaverri, D. Rappaport, J. Tejel. Moving coins. *Computational Geometry*, 34(1):35-48, 2006.
- [2] N. G. de Bruijn. Aufgaben 17 and 18 (in Dutch). Nieuw Archief voor Wikskunde, 2(954):67.
- [3] B. Chazelle, T. Ottmann, E. Soisalon-Soininen, D. Wood. The Complexity and Decidability of Separation ICALP 119-127, 1984.
- [4] A. Dumitrescu, M. Jiang. On Reconfiguration of Disks in the Plane and Related Problems WADS 254-265, 2009.
- [5] L. Fejes-Tóth and A. Heppes. Über stabile Körpersysteme (in German). Compositio Mathematica, 15(2):119–126, 1963.
- [6] S. Foldes, I. Rival and J. Urrutia. Light sources obstructions and spherical orders. *Discrete Mathematics*, 102(1):13–23, 1992.
- [7] L. J. Guibas and F. F. Yao. On translating a set of rectangles. In Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, 154–160, 1980.
- [8] F. Hurtado, M. Noy and Jorge Urrutia. Flipping Edges in Triangulations. Discrete & Computational Geometry, 22(3):333-346, 1999.
- [9] G. Toussaint. Movable separability of sets. Computational Geometry, North-Holland, Amsterdam, 335–375, 1985.
- [10] I. Rival and J. Urrutia. Representing orders on the plane by translating convex figures. Order, 4(4):319– 339, 1988.

On k-Gons and k-Holes in Point Sets

Oswin Aichholzer^{*} Ruy Fabila-Monroy[†] Hernán González-Aguilar [‡] Thomas Hackl^{*} Marco A. Heredia [§] Clemens Huemer[¶] Jorge Urrutia[‡] Pavel Valtr[∥] Birgit Vogtenhuber^{*}

Abstract

We consider a variation of the classical Erdős-Szekeres problems on the existence and number of convex k-gons and k-holes (empty k-gons) in a set of n points in the plane. Allowing the k-gons to be non-convex, we show bounds and structural results on maximizing and minimizing their numbers. Most noteworthy, for any k and sufficiently large n, we give a quadratic lower bound for the number of k-holes, and show that this number is maximized by sets in convex position. We also provide an improved lower bound for the number of convex 6-holes.

1 Introduction

Let S be a set of n points in general position in the plane. A k-gon is a simple polygon spanned by k points of S. A k-hole is an empty k-gon; that is, a k-gon which contains no points of S in its interior.

Around 1933 Esther Klein raised the following question which was (partially) answered in the classical paper by Erdős and Szekeres [12] in 1935: "Is it true that for any k there is a smallest integer g(k) such that any set of g(k) points contains at least one convex k-gon?" As observed by Klein, g(4) = 5, and Kalbfleisch et al. [19] solved the more involved case of g(5) = 9. The case k = 6 was only solved as recently as 2006 by Szekeres and Peters [23]. They showed that g(6) = 17 by an exhaustive computer search. The well known Erdős–Szekeres Theorem [12] states that g(k) is finite for any k. The current best bounds are $2^{k-2} + 1 \le g(k) \le {\binom{2k-5}{k-2}} + 1$ for $k \ge 5$; see [13, 24].

^{||}Department of Applied Mathematics and Institute for Computer Science (ITI), Charles University, Prague, Czech Republic Erdős and Guy [11] posed the following generalization: "What is the least number of convex k-gons determined by any set S of n points in the plane?" The trivial solution for the case k = 3 is $\binom{n}{3}$. But for convex 4-gons this question is related to the search for the rectilinear crossing number $\bar{cr}(S)$ of S; see the next section for details.

In 1978 Erdős [9] raised the following question for convex k-holes: "What is the smallest integer h(k) such that any set of h(k) points in the plane contains at least one convex k-hole?" As had been observed by Esther Klein, every set of 5 points determines a convex 4-hole, and 10 points always contain a convex 5-hole, a fact proved by Harborth [17]. However, in 1983 Horton showed that there exist arbitrarily large sets of points containing no convex 7-hole [18]. It took almost a quarter of a century after Horton's construction to answer the existence question for 6-holes. In 2007/08 Nicolás [21] and independently Gerken [16] proved that every sufficiently large point set contains a convex 6-hole.

Erdős also proposed the following variation of the problem [10]. "What is the least number $h_k(n)$ of convex k-holes determined by any set of n points in the plane?" We know by Horton's construction that $h_k(n) = 0$ for $k \ge 7$. Table 1 shows the current best lower and upper bounds for k = 3...6; see [4, 5, 7, 14] and Section 6.

$n^2 - O(n) \le h_2(n) \le 1.6196n^2 + o(n^2)$
$n^2 = O(1) \leq h_3(n) \leq 1.0100n^2 + O(n^2)$
$\frac{n}{2} - O(n) \le h_4(n) \le 1.9397n^2 + o(n^2)$
$3\lfloor \frac{n-4}{8} \rfloor \le h_5(n) \le 1.0207n^2 + o(n^2)$
$\left\lfloor \frac{n-1}{858} \right\rfloor - 2 \le h_6(n) \le 0.2006n^2 + o(n^2)$

Table 1: Bounds on the number $h_k(n)$ of convex k-holes.

In this paper we generalize the above questions by allowing k-gons and k-holes to be non-convex. Thus whenever we refer to a (general) k-gon or k-hole, unless it is specifically stated to be convex or non-convex, it could be either. We remark that in some related literature, k-holes are assumed to be convex.

A set of k points in convex position obviously spans precisely one convex k-hole. In contrast, a point set might admit exponentially many different polygoniza-

^{*}Institute for Software Technology, University of Technology, Graz, Austria, [oaich|thackl|bvogt]@ist.tugraz.at

[†]Departamento de Matemáticas, Cinvestav, Mexico City, Mexico, ruyfabila@math.cinvestav.edu.mx

[‡]Instituto de Matemáticas, Universidad Nacional Autónoma de México, Mexico City, Mexico, [hernan|urrutia]@matem.unam.mx

[§]Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Mexico City, Mexico, marco@ciencias.unam.mx

[¶]Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Barcelona, Spain, clemens.huemer@upc.edu

	numbers of k-gons				numbers of k-holes			
	convex	non-convex	general		convex	non-convex	gene	eral
	min	max	min	\max	min	max	min	max
k=4	$egin{array}{c} ar{c}r(n) \ \Theta(n^4) \end{array}$	$\begin{array}{c}3\binom{n}{4}-3c\bar{r}(n)\\\Theta(n^4)\end{array}$		$\begin{array}{c} 3\binom{n}{4} \\ -2\bar{cr}(n) \\ \Theta(n^4) \end{array}$	$ \frac{\geq \frac{n^2}{2} - O(n)}{\leq 1.9397n^2 + o(n^2)} \Theta(n^2) [5, 7] $	$ \leq \frac{n^3}{2} - O(n^2) \geq \frac{n^3}{2} - O(n^2 \log n) \Theta(n^3) [3] $	$ \geq \frac{5}{2}n^2 - O(n) \leq \frac{n^3}{2} + O(n^2) \Omega(n^2) [3], O(n^3) [Sec. 5] $	
k = 5	$\Theta(n^5)$ [6]	$ \begin{array}{c} 10\binom{n}{5} \\ -2(n-4)\bar{cr}(n) \\ \Theta(n^5) \ [4] \end{array} $	$ \begin{pmatrix} n \\ 5 \end{pmatrix} \\ \Theta(n^5) \ [4] $	$\begin{array}{l} \Theta(n^5)\\ [\text{Sec. 2}] \end{array}$	$ \begin{split} &\geq 3\lfloor \frac{n-4}{8} \rfloor \\ &\leq 1.0207n^2 + o(n^2) \\ &\Omega(n) \ [4], O(n^2) \ [5] \end{split} $	$ \begin{array}{l} \leq n!/(n-4)! \\ \Theta(n^4) \ [\text{Sec. 4}] \end{array} $	$ \begin{split} &\geq 17n^2 - O(n) \\ &\leq O(n^{\frac{7}{2}}) \\ &\Omega(n^2) \; [4], O(n^{\frac{7}{2}}) \; [\text{Sec. 5}] \end{split} $	
$k \ge 6$	$\Theta(n^k)$ [6]	$\Theta(n^k)$ [Sec. 2]		$\begin{array}{l} \Theta(n^k) \\ [\text{Sec. 2}] \end{array}$	$ \begin{aligned} k = 6 &: \ge \lfloor \frac{n-1}{588} \rfloor - 2 \\ O(n^2) & [5] \\ \Omega(n) & [Sec. 6] \\ k \ge 7 &: \emptyset & [18] \end{aligned} $	$ \frac{\leq n!/(n-k+1)!}{\Theta(n^{k-1}) \text{ [Sec. 4]}} $	$ \begin{split} &\geq n^2 - O(n) \\ &\leq O(n^{\frac{k+2}{2}}) \\ &\Omega(n^2), O(n^{\frac{k+2}{2}}) \text{ [Sec. 5]} \end{split} $	$ \binom{n}{k} \\ \Theta(n^k) \\ [Sec. 3] $

Table 2: Bounds on the numbers of convex, non-convex and general k-gons and k-holes for n points and constant k.

tions (spanning cycles) [8, 15, 22], which implies that the number of k-gons and k-holes can be larger than $\binom{n}{k}$. This makes questions like minimizing or maximizing the number of non-convex and general k-holes more challenging than they might appear at first glance.

Table 2 summarizes known bounds on the numbers of k-gons and k-holes, including the results of this paper. Every entry in the table shows lower and upper bounds, also in explicit form if available. Among other results, we generalize properties concerning 4-holes [3] and 5-holes [4] to $k \geq 6$. In Section 2 we give asymptotic bounds on the number of non-convex and general k-gons. In Section 3 we consider (general) k-holes. We show that for sufficiently small k their number is maximized by sets in convex position, which is not the case for large k. Section 4 provides a tight bound for the maximum number of non-convex k-holes, and Section 5 contains bounds for the minimum number of general k-holes. In Section 6 we improve the lower bound for convex 6-holes, and we conclude with open problems in Section 7.

2 General k-gons

For non-convex k-gons of small cardinalty their number can be related to the rectilinear crossing number $\bar{cr}(S)$ of a set S of n points in the plane. This is the number of proper intersections in the drawing of the complete straight line graph on S. By $\bar{cr}(n)$ we denote the minimum possible rectilinear crossing number over all point sets of cardinality n. Determining $\bar{cr}(n)$ is a well known problem in discrete geometry; see [6, 11] as general references and [2] for bounds on small sets. Asymptotically we have $\bar{cr}(n) \approx 0.38 {n \choose 4} = \Theta(n^4)$ [1].

It is easy to see that the number of convex 4-gons is equal to $\bar{cr}(S)$ and is thus minimized by sets realizing $\bar{cr}(n)$. Moreover, as four points in non-convex position span three non-convex 4-gons, we have at most $3\binom{n}{4} - 3\bar{cr}(n) \approx 1.86\binom{n}{4}$ non-convex and at most $3\binom{n}{4}-2c\bar{r}(n)\approx 2.24\binom{n}{4}$ general 4-gons. All these bounds are tight for point sets which minimize the rectilinear crossing number.

A similar relation has been obtained for the number of non-convex 5-gons in [4]: Any set of n points has at most $10\binom{n}{5} - 2(n-4)\bar{cr}(n) \approx 6.2\binom{n}{5}$ non-convex 5-gons, and again this bound is obtained for sets minimizing the rectilinear crossing number. Note that this number exceeds the maximum number of convex 5-gons. For the number of general 5-gons, and for non-convex and general k-gons with $k \geq 6$, no such direct relations to $\bar{cr}(n)$ are possible.



Figure 1: The so-called double chain DC(n).

Polygonizations, also called spanning cycles, can be considered as k-gons of maximal size (i.e., k = n). García et al. [15] construct a point set with $\Omega(4.64^n)$ spanning cycles, the so-called double chain DC(n), which is currently the best known example; see Figure 1. The upper bound on the number of spanning cycles of any *n*-point set was improved several times during the last years, most recently to $O(70.21^n)$ [22] and $O(68.664^n)$ [8], neglecting polynomial factors in the asymptotic expressions. The minimum is obtained by point sets in convex position, which have exactly one spanning cycle.

For the number of general k-gons this implies a lower bound of $\binom{n}{k}$, as well as an upper bound of $O(68.664^k\binom{n}{k})$. For constant k, we obtain $\Theta(n^k)$. On

the other hand, the double chain provides $\Omega(n^k)$ nonconvex k-gons, where $k \ge 4$ is again a constant. To see this, choose one vertex from the upper chain of DC(n) and $k-1 \ge 3$ vertices from the lower chain of DC(n), and connect them to a simple, non-convex polygon. This gives at least $\frac{n}{2} \binom{n/2}{k-1} = \Omega(n^k)$ non-convex k-gons. As the lower bound on the maximal number of non-convex k-gons asymptotically matches the upper bound on the maximal number of general k-gons, we get our first result.

Lemma 1 Let S be a set of n points in the plane in general position and $k \ge 3$ a constant. Then the maximum number of non-convex k-gons in S is $\Theta(n^k)$ and the maximum number of general k-gons in S is also $\Theta(n^k)$.

3 Maximizing the number of (general) k-holes

In [3] it is shown that the number of 4-holes is maximized for point sets in convex position if n is sufficiently large. It was conjectured that this is true for any constant $k \ge 4$. The following theorem settles this conjecture in the affirmative.

Theorem 2 For every $k \ge 4$ and $n \ge 2(k-1)!\binom{k}{4}+k-1$, the number of k-holes is maximized by a set of n points in convex position.

Proof. Every non-convex k-hole has as its vertex set a non-convex k-tuple, and every non-convex k-tuple has at least one triangle formed by three extreme points (i.e., points on the convex hull of the k-tuple) that contains points of the k-tuple in its interior. So consider such a non-empty triangle Δ . We count the number of non-convex k-holes having the three vertices of Δ as extreme points. Note that any such k-hole can be reduced to a (not necessarily simple) non-empty (k-1)-gon by removing a reflex vertex from its boundary.

Denote by \mathcal{K} the set of (not necessarily simple) nonempty (k-1)-gons having the vertices of Δ on their convex hull. First, $|\mathcal{K}|$ can be bounded from above by the number of (not necessarily simple) possibly empty (k-1)-gons having the three vertices of Δ on their boundary, which is $\frac{(k-2)!}{2} \binom{n-3}{k-4}$.

Further, every (k-1)-gon in \mathcal{K} can be completed to a (simple) non-convex k-hole in at most k-1 ways by adding a reflex vertex. Thus the number of non-convex k-holes having all vertices of Δ on their convex hull is bounded from above by

$$(k-1)\frac{(k-2)!}{2}\binom{n-3}{k-4} = \frac{(k-1)!}{2}\binom{n-3}{k-4}.$$

Considering convex k-holes, observe that every ktuple gives at most one convex k-hole. Denote by N the number of k-tuples that do *not* form a convex k-hole, and by T the number of non-empty triangles. Then we get (1) as a first upper bound on the number of (general) k-holes of a point set.

$$\binom{n}{k} - N + \left(\frac{(k-1)!}{2}\binom{n-3}{k-4}\right) \cdot T \tag{1}$$

To obtain an improved upper bound from (1), we need to derive a good lower bound for N. To this end, consider again a non-empty triangle Δ . As Δ is not empty, none of the $\binom{n-3}{k-3}$ k-tuples that contain all three vertices of Δ forms a convex k-hole. On the other hand, for such a k-tuple, all of its $\binom{k}{3}$ contained triangles might be nonempty. We obtain $T \cdot \binom{n-3}{k-3} / \binom{k}{3}$ as a lower bound for N, and thus (2) as an upper bound for the number of k-holes.

$$\binom{n}{k} + \left(\frac{(k-1)!}{2}\binom{n-3}{k-4} - \frac{\binom{n-3}{k-3}}{\binom{k}{3}}\right) \cdot T \qquad (2)$$

For $n \ge 2(k-1)!\binom{k}{4} + k - 1$ this is at most $\binom{n}{k}$, the number of k-holes of a set of n points in convex position, which proves the theorem.

The above theorem states that convexity maximizes the number of k-holes for $k = O(\frac{\log n}{\log \log n})$ and sufficiently large n. Moreover, the proof implies that any non-empty triangle in fact reduces the number of empty k-holes. Thus it follows that, for $k = O(\frac{\log n}{\log \log n})$ and n sufficiently large, the maximum number of convex k-holes is strictly larger than the maximum number of non-convex k-holes; see also the next section.

At the other extreme, for $k \approx n$ the statement does not hold: As already mentioned in the introduction, a set of k points spans at most one convex k-gon, but might admit exponentially many different non-convex k-gons.

Theorem 3 The number of k-holes in the double chain DC(n) on n points is at least

$$\binom{\frac{n-4}{2}}{\frac{n-k}{2}} \cdot \frac{n-k+2}{2} \cdot \Omega(4.64^k)$$

Proof. Recall that DC(n) admits $\Omega(4.64^n)$ polygonizations. Thus, for a double chain on k points (k/2 points on each chain), we have $\Omega(4.64^k)$ different k-polygonizations. We distribute the remaining n - k points among all possible positions, meaning that for each k-polygonization, we obtain the double chain on n points with a k-hole drawn, as shown in Figure 2.

In their proof, García et al. count paths that start at the first vertex of the upper chain and end at the last vertex of the lower chain. Before the first vertex on the lower chain, they add an additional point q to complete these paths to polygonizations. We slightly extend this principle, by also adding an additional point p on the upper chain after the last vertex. Then we complete each path C to a polygonization in one of the following



Figure 2: Two ways to complete a path to a polygonization.

ways: Either we add p to C directly next to $p_{\frac{k}{2}-1}$ and then complete C via q, obtaining P_q , or we add q to Cdirectly next to q_1 , and close the polygonization via p, obtaining P_p .

Note that this changes the number of polygonizations only by a constant factor and thus does not influence the asymptotic bound. However, the interior of P_q is the exterior of P_p , meaning that if we place a point somewhere on the double chain and it lies inside P_q , then it lies outside P_p , and vice versa. It follows that, in one of the two polygonizations, at least half of the k + 2positions to insert points are outside the polygonization. Hence we can distribute the $\frac{n-k}{2}$ points on each chain to at least $\frac{k}{2} + 1$ possible positions in total. Now, on one of the two chains we have at least $\frac{k}{4} + 1$ positions; see again Figure 2. More precisely, there are $\frac{k}{4} + j + 1$ positions on this chain (where $0 \le j < \frac{k}{4}$), and on the other chain there are (at least) $\max\{2, \frac{k}{4} - j\}$ positions. Using this, we obtain

$$\binom{\frac{n-k}{2}+\frac{k}{4}+j}{\frac{n-k}{2}}\cdot \max\left\{\binom{\frac{n-k}{2}+1}{\frac{n-k}{2}}, \binom{\frac{n-k}{2}+\frac{k}{4}-j-1}{\frac{n-k}{2}}\right\}$$

possibilities to place the remaining points on the two chains. This factor is minimized for $j = \frac{k}{4} - 2$, which yields the claimed lower bound for the number of k-holes of DC(n).

4 An upper bound for non-convex *k*-holes

The following theorem shows that, asymptotically, the maximum number of non-convex k-holes is smaller than the maximum number of convex k-holes.

Theorem 4 For any constant $k \geq 3$, the number of non-convex k-holes in a set of n points is bounded by $O(n^{k-1})$ and there exist sets with $\Theta(n^{k-1})$ non-convex k-holes.

Proof. We first show that there are at most $O(n^{k-1})$ non-convex k-holes by giving an algorithmic approach to generate all non-convex k-holes. We represent a non-convex k-hole by the counter-clockwise sequence of its vertices, where we require that the last vertex is reflex. Note that any non-convex k-hole has $r \ge 1$ such representations, where r is the number of its reflex vertices. Thus the number of different representations is an upper bound on the number of non-convex k-holes.

We have n possibilities to choose the first vertex v_1 , n-1 for the second vertex v_2 , and so on. Several of the sequences obtained might lead to non-simple polygons, but we are only interested in an upper bound. For the second-last vertex v_{k-1} we have n-k+2 possibilities, but the last vertex v_k is uniquely defined. As v_k is required to be reflex and the polygon has to be empty, we have to use the inner geodesic connecting v_{k-1} back to v_1 . Only if this geodesic contains exactly one point, namely v_k , we do obtain one non-convex k-hole (again ignoring possible non-simplicity). Thus we obtain at most $n!/(n-k+1)! = O(n^{k-1})$ non-convex k-holes.



Figure 3: A set with $\Theta(n^{k-1})$ non-convex k-holes.

For an example which achieves this bound see Figure 3. Each of the four indicated groups of points contains a linear fraction of the point set; e.g. $\frac{n}{4}$ points. It is sufficient to only consider the k-holes with triangular convex hull of the type indicated in the figure, which sums to $\Omega(n^3 \cdot \binom{n}{k-4}) = \Omega(n^{k-1})$ non-convex k-holes.

5 On the minimum number of (general) *k*-holes

Every set of k points admits at least one polygonization. Using this obvious fact, we obtain the following result.

Theorem 5 Let S be a set of n points in the plane in general position. For every c < 1 and every $k \leq c \cdot n$, S contains $\Omega(n^2)$ k-holes. **Proof.** We follow the lines of the proof of Theorem 5 in [3]. Consider the point set S in x-sorted order, $S = \{p_1, \ldots, p_n\}$, and sets $S_{i,j} = \{p_i, \ldots, p_j\} \subseteq S$. The number of sets $S_{i,j}$ of cardinality at least k is

$$\sum_{i=1}^{n-k+1} \sum_{j=i+k-1}^{n} 1 = \frac{(n-k+1)(n-k+2)}{2} = O(n^2)$$

For each $S_{i,j}$ use the k-2 points of $S_{i,j} \setminus \{p_i, p_j\}$ which are closest to the segment $p_i p_j$ to obtain a subset of kpoints including p_i and p_j . Each such set contains at least one k-hole which has p_i and p_j among its vertices. Moreover, as p_i and p_j are the left and rightmost points of $S_{i,j}$, they are also the left and rightmost points of this k-hole. This implies that any k-hole of S can count for at most one set $S_{i,j}$, which gives a lower bound of $\Omega(n^2)$ for the number of k-holes in S.

Theorem 6 For every constant $k \ge 4$ and every $n = m^2 \ge k$, there exist sets with n points in general position that admit at most $O(n^2(\sqrt{n}\log n)^{k-3})$ k-holes.

Proof. The point set S we consider is the squared Horton set of size $\sqrt{n} \times \sqrt{n}$; see [25]. Roughly speaking, S is a grid which is perturbed such that every set of originally collinear points forms a Horton set. It can be shown that for any two points $p, q \in S$, the number of empty triangles in S that contain the edge pq is $O(\sqrt{n} \log n)$, regardless of the choice of p and q; details will be given in the full version of this paper.

To estimate the number of k-holes in S, we will use triangulations and their dual: For a triangulation of a k-hole, the dual is a binary tree where every node represents a triangle. It can be rooted at any triangle that has an edge on the boundary of the k-hole; see [20]. It is well known that there are $C_{k-2} = O(4^k \cdot k^{-\frac{3}{2}})$ such rooted binary trees [20]. Although exponential in k, this bound is constant in the size n of S.

Now pick an empty triangle Δ in S and an arbitrary rooted binary tree B. Consider all k-holes which contain Δ and admit a triangulation that is represented by B rooted at Δ . As the number of empty triangles incident to an edge in S is $O(\sqrt{n} \log n)$, each of the n-3edges in B yields $O(\sqrt{n} \log n)$ possibilities to continue a triangulated k-hole, and we obtain an upper bound of $O((\sqrt{n} \log n)^{k-3})$ for the number of triangulations of k-holes for Δ that represent B.

Multiplying this by the (constant) number of rooted binary trees of size k-2 does not change the aysmptotics and thus yields an upper bound of $O((\sqrt{n} \log n)^{k-3})$ for the number of all triangulations of all k-holes containing Δ . As any k-hole can be triangulated, this is also an upper bound for the number of k-holes containing Δ .

Finally, there are $O(n^2)$ empty triangles in S (see again [5]), and thus we obtain $O(n^2(\sqrt{n}\log n)^{k-3})$ as an upper bound for the number of k-holes in S.

Note that the Horton set has $\Omega(n^3)$ 4-holes. A general super-quadratic lower bound for the number of 4-holes would solve a conjecture of Bárány to the positive, showing that every point set contains an edge that spans a super-constant number of 3-holes; see e.g. [6], Chapter 8.4, Problem 4. This would also imply a quadratic lower bound for the number of convex 5-holes. So far, not even a super-linear bound is known for the latter problem [6].

6 An improved lower bound for convex 6-holes

Gerken [16] showed that each set of at least 1717 points in general position contains a convex 6-hole. This immediately implies that each set of n points contains a linear number of convex 6-holes, namely at least $\lfloor \frac{n}{1717} \rfloor$. In the following we slightly improve on this bound. We start by showing a result for monochromatic convex 6-holes in two-colored point sets.

Lemma 7 Each set of r red points and b blue points in general position in the plane with $r \ge 1716 \left\lceil \frac{b}{2} \right\rceil + 1717$ contains a convex red 6-hole.

Proof. Consider a non-crossing perfect matching of the blue points; if *b* is odd, then allow one isolated point *p*. We extend the segments (in both directions) one by one, until each segment either hits another segment, the line of a previously extended segment or goes to infinity. If *b* is odd, we take an arbitrary segment through *p* and extend it as well. Altogether, this results in a decomposition of the plane into $\left\lceil \frac{b}{2} \right\rceil + 1$ convex regions. As the red points lie inside these regions, it follows by the pigeon-hole principle that at least one of these regions contains 1717 red points, and thus a red convex 6-hole by [16].

Theorem 8 Each set S of n points in general position in the plane contains at least $\lfloor \frac{n-1}{858} \rfloor - 2$ convex 6-holes.

Proof. We prove the statement by contradiction. Assume that the point set S contains strictly less than $\lfloor \frac{n-1}{858} \rfloor - 2$ convex 6-holes, and color the points of S red. Now we eliminate all red convex 6-holes by placing an additional blue point inside each of them, such that the resulting two-colored point set is in general position. By this, at most $b \leq \lfloor \frac{n-1}{858} \rfloor - 3$ blue points are added. Therefore the number n of red points is at least

$$n \ge 858(b+3) + 1 \ge 1716\left\lceil \frac{b}{2} \right\rceil + 1717.$$

By Lemma 7, any such two-colored point set contains a convex red 6-hole, a contradiction. $\hfill \Box$

7 Conclusion

We have shown various lower and upper bounds on the numbers of convex, non-convex, and general k-holes and k-gons in point sets. Several questions remain unsettled. For example, some of the presented bounds are not tight, like the classic question for the minimum number of convex k-holes for $k \leq 6$. Maybe the most intriguing open question in this context is whether there exists a super-quadratic lower bound for the number of general k-holes for $k \geq 4$.

Acknowledgments

Research on this topic was initiated during the *Third Workshop on Discrete Geometry and its Applications* in Morelia (Michoacán, Mexico). We thank Edgar Leonel Chávez González, and Feliu Sagols for helpful discussions. Research of Oswin Aichholzer, Thomas Hackl, and Birgit Vogtenhuber supported by the FWF [Austrian Fonds zur Förderung der Wissenschaftlichen Forschung] under grant S9205-N12, NFN Industrial Geometry. Research of Clemens Huemer partially supported by projects MEC MTM2009-07242 and Gen. Cat. DGR 2009SGR1040. Research of Marco Antonio Heredia, Hernán González-Aguilar, and Jorge Urrutia partially supported by CONACyT (Mexico) grant CB-2007/80268. Work by Pavel Valtr supported by project 1M0545 of the Ministry of Education of the Czech Republic. We thank the anonymous referees for their helpful comments.

References

- B. M. Ábrego, S. Fernández-Merchant, J. Leaños, and G. Salazar. A central approach to bound the number of crossings in a generalized configuration. *Electronic Notes in Discrete Mathematics*, 30:273–278, 2008.
- [2] O. Aichholzer. On the rectilinear crossing number. http://www.ist.tugraz.at/aichholzer/research/rp/ triangulations/crossing/.
- [3] O. Aichholzer, R. Fabila-Monroy, H. González-Aguilar, T. Hackl, M. A. Heredia, C. Huemer, J. Urrutia, and B. Vogtenhuber. 4-holes in point sets. In Proc. 27th European Workshop on Computational Geometry EuroCG'11, pages 115–118, Morschach, Switzerland, 2011.
- [4] O. Aichholzer, T. Hackl, and B. Vogtenhuber. On 5holes and 5-gons. In Proc. XIV Encuentros de Geometría Computacional ECG2011, pages 7–10, Alcalá de Henares, Spain, 2011.
- [5] I. Bárány and P. Valtr. Planar point sets with a small number of empty convex polygons. *Studia Scientiarum Mathematicarum Hungarica*, 41(2):243–269, 2004.
- [6] P. Brass, W. Moser, and J. Pach. Research problems in discrete geometry. *Springer*, 2005.
- [7] A. Dumitrescu. Planar sets with few empty convex polygons. *Studia Scientiarum Mathematicarum Hun*garica, 36(1-2):93–109, 2000.

- [8] A. Dumitrescu, A. Schulz, A. Sheffer, and C. Tóth. Bounds on the maximum multiplicity of some common geometric graphs. In Proc. 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011), Leibniz International Proceedings in Informatics (LIPIcs), pages 637–648, Dagstuhl, Germany, 2011.
- [9] P. Erdős. Some more problems on elementary geometry. Austral. Math. Soc. Gaz., 5:52–54, 1978.
- [10] P. Erdős. Some old and new problems in combinatorial geometry. In Convexity and Graph Theory, M. Rosenfeld et al., eds., Annals Discrete Math., 20:129– 136, 1984.
- [11] P. Erdős and R. Guy. Crossing number problems. Amer. Math. Monthly, 88:52–58, 1973.
- [12] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.
- [13] P. Erdős and G. Szekeres. On some extremum problems in elementary geometry. Ann. Univ. Sci. Budapest. Eötvös, Sect. Math., 3/4:53–62, 1960.
- [14] A. García. A note on the number of empty triangles. In Proc. XIV Encuentros de Geometría Computacional ECG2011, pages 101–104, Alcalá de Henares, Spain, 2011.
- [15] A. García, M. Noy, and J. Tejel. Lower bounds on the number of crossing-free subgraphs of K_n . Computational Geometry: Theory and Applications, 16:211–221, 2000.
- [16] T. Gerken. Empty convex hexagons in planar point sets. Disc. Comp. Geom., 39(1-3):239–272, 2008.
- [17] H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente Math.*, 33:116–118, 1978.
- [18] J. Horton. Sets with no empty convex 7-gons. Canad. Math. Bull., 26(4):482–484, 1983.
- [19] J. Kalbfleisch, J. Kalbfleisch, and R. Stanton. A combinatorial problem on convex n-gons. In Proc. Louisiana Conference on Combinatorics, Graph Theory and Computing, pages 180–188, Louisiana State University, 1970.
- [20] J. A. D. Loera, J. Rambau, and F. Santos. Triangulations: Structures for Algorithms and Applications, volume 25 of Algorithms and Computation in Mathematics. Springer-Verlag, 2010.
- [21] C. Nicolás. The empty hexagon theorem. Disc. Comp. Geom., 38(2):389–397, 2007.
- [22] M. Sharir and A. Sheffer. Counting triangulations of planar point sets. arXiv:0911.3352, 2009.
- [23] G. Szekeres and L. Peters. Computer solution to the 17point Erdős–Szekeres problem. *The ANZIAM Journal*, 48(2):151–164, 2006.
- [24] G. Tóth and P. Valtr. The Erdős–Szekeres theorem: upper bounds and related results. Combinatorial and Computational Geometry, J.E. Goodman, J. Pach, and E. Welzl (Eds.),, 52:557–568, 2005.
- [25] P. Valtr. Convex independent sets and 7-holes in restricted planar point sets. *Disc. Comp. Geom.*, 7:135– 152, 1992.

Hardness Results for Two-Dimensional Curvature-Constrained Motion Planning

David Kirkpatrick*

Irina Kostitsyna[†]

Valentin Polishchuk[‡]

Abstract

We revisit the problem of finding curvature-constrained paths in a polygonal domain with holes. We give a new proof that finding a *shortest* curvature-constrained path is NP-hard; our proof is substantially simpler, and makes fewer assumptions about the polygonal domain, than the earlier proof of [Reif and Wang, 1998]. We also prove that it is NP-hard to decide existence of a *simple* (i.e., non-self-intersecting) path.

1 Introduction

Understanding the feasibility and optimality of the motion of car-like robots in the presence of obstacles entails, among many things, an understanding of curvature-constrained paths between specified configurations in the plane, that avoid a given set of obstacles. The study of curvature-constrained path planning has a rich history that long predates and goes well beyond robot motion planning, for example the work of Markov [29] on the construction of railway segments.

1.1 Definitions

Let P be a polygonal domain with holes (forbidden regions, or obstacles) in \mathbb{R}^2 . Let $\pi : [0, L] \mapsto P$ be a continuous differentiable path, parameterized by arc length, and denote by $\pi'(t)$ the derivative of π at t. Path π is said to be *curvature constrained* if, for some constant c, the *average curvature* of π on every interval $[t_1, t_2] \subseteq [0, L]$, namely $||\pi'(t_1) - \pi'(t_2)||/|t_1 - t_2|$, is bounded above by c. Intuitively, every point on such a path can be sandwiched between two tangent circles of radius 1/c. We assume that $\pi(0) = s, \pi(L) = t$ are two given points in P, and $\pi'(0) = S, \pi'(L) = T$ are two given vectors; the pair (s, S) (resp., (t, T)) is called the *initial* (resp., *final*) configuration of π . The path π is simple if it has no self-intersections: $\pi(t_1) \neq \pi(t_2)$ for $t_1 \neq t_2$. Under suitable scaling we can assume that c = 1. Hereafter we will assume that all paths avoid the interior of obstacles, that is they remain within P; all such paths with average curvature bounded by 1 are referred to as *admissible* paths.

1.2 Background

A fundamental result in curvature-constrained motion planning, due to Dubins [15], states that in the absence of obstacles shortest admissible paths are one of two types: a (unit) circular arc followed by a line segment followed by another arc (*CLC*), or a sequence of three circular arcs (*CCC*)¹. Variations and generalizations of the problem were studied in [8,9,11,13,14,18,28,30,31, 33,34,36,37].

Dubins' characterization plays a fundamental role in establishing the existence as well as the optimality of curvature-constrained paths. Jacobs and Canny [22] showed that even in the presence of obstacles it suffices to restrict attention to paths of Dubins form between obstacle contacts and that if such a path exists then the shortest such path is well-defined. Fortune and Wilfong [19] give a super-exponential time algorithm for determining the existence of, but not actually constructing, such a path. Characterizing the intrinsic complexity of the existence problem for curvature-constrained paths is hampered by the fact that there are no known bounds on the minimum length or *intricacy* (number of elementary segments), expressed as a function of the description of the polygonal domain, of obstacle-avoiding paths in Dubins form. In a variety of restricted domains polynomial-time algorithms exist that construct shortest admissible paths [1, 2, 6].

The NP-hardness of computing a shortest admissible path amid polygonal obstacles was established by Reif and Wang [32]. This motivated a variety of approaches to approximating shortest admissible paths including [4, 5,22,35,38–40]. It is known, for example, that shortest robust paths, shortest paths of bounded intricacy and minimum intricacy paths of bounded length all have polynomial-time approximations. The books [25,26] are general references; for some very recent work on Dubins paths see [7, 12, 16, 17, 20, 21].

^{*}Department of Computer Science, University of British Columbia, kirk@cs.ubc.ca. Supported by the Natural Sciences and Engineering Research Council of Canada.

[†]Department of Computer Science, State University of New York at Stony Brook, ikost@cs.stonybrook.edu

[‡]Helsinki Institute for Information Technology, Department of Computer Science, University of Helsinki, polishch@helsinki.fi. Supported by the Academy of Finland grant 138520.

¹In general, any of the C or L segments could have zero length.

1.3 Results

In Section 2 we present a new proof that finding a shortest admissible path from (s, S) to (t, T) is NP-hard. Our proof is considerably simpler than the one given in [32]. In addition, our construction is more "robust" in that it applies even in non-degenerate polygonal domains, specifically domains that have no "pinhole" gaps between obstacles.

Our hardness proof in Section 2 depends critically on the fact that admissible paths can self-intersect. This leaves open the possibility that the problem of finding a shortest simple admissible path could be solved in polynomial time. (Note that the simplicity constraint is relevant in many applications; e.g. laying out a conveyor belt or designing a pipeline.) In Section 3 we show that this is not the case: even deciding the *existence* of an admissible path is NP-hard, if we restrict attention to simple paths (which, of course, implies that finding any approximation to the shortest such path is NP-hard).

2 NP-hardness of determining shortest curvatureconstrained paths

Our NP-hardness proof involves a reduction from 4CNF-satisfiability. Specifically, suppose that Φ is a formula in 4CNF involving m clauses and k variables X_0, \ldots, X_{k-1} . We show how to construct a polygonal environment E, whose description is bounded in size by some polynomial in k, together with configurations (s, S) and (t, T) and a distance D, such that there exists an admissible path from (s, S) to (t, T) whose length is at most D if and only if Φ is satisfiable.

Our proof, like that of Reif and Wang, uses the idea of path-encoding, introduced by Canny and Reif [10] in their proof that determining the shortest obstacleavoiding path, with no constraint on curvature, joining specified points in \mathbb{R}^3 , is NP-hard. The fact that our problem is set in \mathbb{R}^2 makes it difficult to adapt the Canny-Reif approach directly (further evidenced by the fact that shortest obstacle-avoiding paths in \mathbb{R}^2 can be constructed in polynomial time, at least in the familiar algebraic model of computation).

In general, the path encoding approach involves first constructing a basic environment that admits exactly 2^k distinct shortest paths (referred to as *canonical paths*) between the two specified placements. These canonical paths all have essentially the same length D_{Φ} that can be distinguished, using a number of bits that is polynomial in k, from all non-canonical paths. Canonical paths are associated with the distinct truth assignments to the variables X_0, \ldots, X_{k-1} . Next, the environment is augmented with additional obstacles that serve to block (filter) every canonical path whose associated truth assignment does not satisfy the formula Φ .

In Reif and Wang's proof canonical paths pass

through a sequence of checkpoints, at distinguishable angles and unequal—but essentially indistinguishable lengths. The complexity of their construction arises from the rather sensitive analysis needed to show that as paths continue and errors propagate these properties are preserved. This exploits, among other things, the existence of pinhole gaps between obstacles, at which the checkpoints are located.

2.1 Overview of the proof

We avoid the complexity of the Reif-Wang construction by mimicking the proof, due to Asano *et al.* [3], of the NP-hardness of minimum-length motion planning for a rod (measuring the trace length of any fixed point), the first construction to employ the path-encoding approach in a planar setting. In this variant, canonical paths all have *exactly* the same length D_{ϕ} . In fact, canonical paths all have exactly the same length as they pass a sequence of checkpoints, vertical lines in our construction. Between these checkpoints the environment consists of elementary modules, each of which performs some basic manipulation of the canonical paths that enter the module. In fact, as we will argue, the properties of our modular construction are *decomposable* in the sense that they assume paths respect curvature constraints only within modules; in the transitions between modules only continuity is assumed. As a consequence, local analysis alone supports a global conclusion: if Φ is not satisfiable then any admissible path from (s, S) to (t, T)has length that exceeds D_{Φ} by an amount that can be expressed in a number of bits that is polynomial in k. Hence, even though D_{Φ} itself may not be exactly expressible in a polynomial (in k) number of bits, there exists a distance $D \geq D_{\Phi}$ that can be so expressed, such that there exists a (relaxed) admissible path of length at most D if and only of Φ is satisfiable.

Figure 1 illustrates the full reduction in schematic form. As it suggests, the construction is based on a sequence of three top-level modules. The first module, what we call a Compound Beam Splitter, splits a single in-coming canonical path into 2^k parallel canonical paths, indexed from 0 on the topmost path to $2^k - 1$ on the bottommost path, with fixed separation σ . We interpret the *b*-th bit of the binary representation of the index of a canonical path as a truth assignment to the variable X_b . Each of these paths has exactly the same length, measured from their common start point S to the vertical line L_1 . The second module is a *Formula Filter* module that obstructs exactly those canonical paths whose associated truth assignment does not satisfy the formula Φ . The third module, a Compound Beam Combiner, is just the mirror image of the first, except that the 2^k in-coming paths have a smaller separation σ' , the result of having passed through the Formula Filter.

It turns out that all three of these modules can be con-



Figure 1: Full Reduction schematic.

structed by appending copies of one elementary (parameterized) module that we call a Wide Beam Splitter $WBS(\Delta)$ or its mirror image a Wide Beam Combiner $WBS^{-1}(\Delta)$ (Fig. 2). The details are analogous to those in the proof of Asano *et al.* [3].



Figure 2: Wide Beam Splitter (WBS) and Combiner (WBS^{-1}) schematics.

2.2 Details of the wide beam-splitter module

The detailed construction of our Wide Beam Splitter is shown in Figure 3. It is described in terms of two parameters w, the width of the module, and Δ_w , the separation of the two canonical paths that emerge from the module. Since w < 4, it is straightforward to see that as w decreases Δ_w decreases. More precisely, since the horizontal (resp., vertical) separation of the centers of the left and right turning circle pairs is w - 2 (resp., $\sqrt{4w - w^2}$), we have $\Delta_w = 4 - 2\sqrt{4w - w^2}$.

As illustrated there are two canonical traversals of the Wide Beam Splitter. Both share a horizontal segment starting at the left terminal. Thereafter one (shown in solid red) traces a C^+C^-L path², emerging at the lower terminal while the other (shown in dashed green) traces a C^-C^+L path, emerging at the upper terminal. It is not hard to confirm that all admissible traversals must make a turn of length at least π on a circle tangent



Figure 3: Wide Beam Splitter detail.

to the right boundary followed, not necessarily immediately, by a turn, of similar length but opposite direction, on a circle tangent to the left boundary.³ Since any (even locally) shortest admissible path must have Dubins form between obstacle contacts, it follows directly that every shortest admissible traversal must have form LCLCL, where the C-segments are doubly supported by obstacles and the middle L-segment may have zero length.

Now, if we focus on admissible paths of form LCLCL joining a point a on vertical line L_1 to a point b on a vertical line L_2 (see Figure 4(i)), it is easily confirmed that in any shortest such path the middle L-segment must have length zero, provided that w, the separation of L_1 and L_2 , is less than 2 and the vertical separation of a and b is at least $\Delta_w/2$ (otherwise, fixing one of the turning circles while moving the other so that the middle L segment degenerates to zero, shortens the path). Among all such LCCL-transitions joining points a and b we can show that the shortest transition has the symmetric form illustrated in Figure 4(ii), independent of the directions at a and b. Furthermore, the shortest such transition, over all pairs of points a and b with vertical separation at least $\Delta_w/2$ (again independent of the directions at a and b), is the one shown in Figure 4(iii) in which a and b have vertical separation exactly $\Delta_w/2.$

These minimality results are proved by reference to Figure 5. The *LC*-transition from *p* to *q* has length $\lambda = d_1 + d_2 + (\pi/2 - \theta) = (w - 1 + \sin\theta)/\cos\theta + (\pi/2 - \theta)$ and its *y*-projection has length $\lambda_y = y_1 + y_2 = ((w - 1)\sin\theta + 1)/\cos\theta$. It is straightforward to confirm that (i) the derivative, with respect to θ , of λ is $((w - 1)\sin\theta)/\cos^2\theta$ and (ii) the derivative, with respect to θ , of λ_y is $((w - 1) + \sin\theta)/\cos^2\theta$. It follows that not only is λ minimized, over configurations with $\theta \geq 0$, when $\theta = 0$, but so also is the derivative of λ

 $^{^2 \}rm We$ denote by $C^+,$ resp., C^- a clockwise (resp., counterclockwise) oriented unit circular arc.

 $^{^{3}}$ A skeptical reader may in fact see alternative traversals of our Wide Beam Splitter, as illustrated. We note that such alternatives can be eliminated, at a sacrifice in clarity of the figure, by narrowing all of the internal corridors sufficiently.



Figure 4: (i) Generic *LCLCL* transition, (ii) minimum length transition between two specified points, and (iii) minimum length transition between two parallel lines.

with respect to λ_y .



Figure 5: Generic LC transition joining points p and q.

2.3 Other remarks on the proof

Despite the comparative simplicity of our NP-hardness construction, the reader may object that we have traded one type of degeneracy in the construction (namely pinhole gaps between obstacles) for another (vertical obstacles spaced at distance exactly two, giving no horizontal freedom for turns in their midst). In fact, this property of our construction is imposed solely to simplify the argument; a very similar splitter construction is possible even if such degeneracies are forbidden.

Although this observation is not a distinguishing feature of our construction, it is worth noting that the hardness result remains intact even if we restrict our attention to shortest admissible paths of bounded intricacy.

2.4 Extensions

One of the advantages of introducing a simpler proof of the NP-hardness of finding shortest curvatureconstrained paths in \mathbb{R}^2 is the potential this raises for establishing the hardness of other related path-planning problems. As an example, we point out that our results for standard curvature-constrained paths are easily modified to apply to polygonal (piecewise linear) paths that satisfy a novel parameterized notion of *discrete bounded curvature* [23] that coincides with the standard notion in the limit. A wide beam splitter designed for discrete bounded curvature paths is illustrated in Figure 6.



Figure 6: Beam Splitter gadget for discrete boundedcurvature paths.

3 Staying simple is hard

We prove that deciding existence of a simple admissible path in a polygonal domain is NP-hard by a reduction from planar 3SAT. Recall that the graph of a 3SAT instance has a vertex for each variable and a vertex for each clause. The graph edges connect clause-variable pairs whenever the variable belongs to the clause. In addition, the graph contains the cycle through variables (Fig. 7). 3SAT is hard even when restricted to instances with planar graphs [27]. We identify a 3SAT instance with its graph.

We transform the graph I of a planar 3SAT instance to an instance of the path finding problem, following the steps analogous to those used to show hardness of finding a simple thick wire in a polygonal do-



Figure 7: (Graph of) an instance I of 3SAT. Variables are shaded circles, clauses are hollow circles.



Figure 8: *I* augmented with parent-child edges, sibling edges, and with variable-clause edges duplicated.



Figure 9: The closed spanning walk W; the numbers indicate the order in which edges are traversed.

main [24]: First, augment I with "parent-child" and "sibling" edges between clauses, and duplicate variableclause edges (Fig. 8). Define a DFS-walk W in the augmented I: the walk goes through orphan clauses, recursing to children clauses, then to variables and to sibling clauses (Fig. 9). Replace vertices of I by variable and clause gadgets (Figs. 10, 11). Finally, turn edges of Iinto corridors connecting clause and variable gadgets.

The crucial ingredients of the proof are the following properties: (1) an admissible path must follow the walk \mathcal{W} (the only flexibility is what clause-variable channel to use in each clause); (2) a variable gadget can be traversed in one of the two ways (setting the truth assignment); (3) for any clause gadget, one of the channels leading to a variable must be used by the path; (4) a variable-clause channel may be used only if the variable satisfies the clause (Fig. 12). Thus, there exists a simple admissible path in the instance iff in each clause there is a channel that can be used by a path to the variable satisfying the clause.



Figure 10: A variable gadget.



Figure 11: When a clause gadget is traversed, from left to right, one of the channels leading to variables must be used. Otherwise, 3 subpaths go through the top of the gadget leading to a self-intersection.



Figure 12: If a variable does not satisfy a clause the channel between them cannot be used by simple path.

References

- P. K. Agarwal, T. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides. Curvature-constrained shortest paths in a convex polygon. SoCG'98.
- [2] P. K. Agarwal, P. Raghavan, and H. Tamaki. Motion planning for a steering-constrained robot through moderate obstacles. *SToC'95*.
- [3] T. Asano, D. G. Kirkpatrick, and C.-K. Yap. Minimizing the trace length of a rod endpoint in the presence of polygonal obstacles is np-hard. *CCCG'03*.
- [4] J. Backer and D. Kirkpatrick. Finding curvatureconstrained paths that avoid polygonal obstacles. SoCG'07.
- [5] J. Backer and D. Kirkpatrick. A complete approximation algorithm for shortest bounded-curvature paths. *ISAAC'08.*
- [6] S. Bereg and D. Kirkpatrick. Curvature-bounded traversals of narrow corridors. SoCG'05.
- [7] S. Bitner, Y. K. Cheung, A. F. Cook, O. Daescu, A. Kurdia, and C. Wenk. Visiting a sequence of points with a bevel-tip needle. *LATIN*'10.
- [8] J.-D. Boissonnat and X.-N. Bui. Accessibility region for a car that only moves forwards along optimal paths. Research Report 2181, INRIA Sophia-Antipolis, 1994.
- [9] X.-N. Bui, P. Souères, J.-D. Boissonnat, and J.-P. Laumond. Shortest path synthesis for Dubins nonholonomic robot. *ICRA*'94.
- [10] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. *FoCS'87*.
- [11] H. Chitsaz and S. LaValle. Time-optimal paths for a Dubins airplane. Conf. Dec. and Cont., 2007.
- [12] H. Chitsaz, S. M. Lavalle, D. J. Balkcom, and M. T. Mason. Minimum wheel-rotation paths for differentialdrive mobile robots. *Int. J. Rob. Res.*, 28:66–80, 2009.
- [13] H. R. Chitsaz. Geodesic problems for mobile robots. PhD thesis, 2008.
- [14] K. Djath, A. Siadet, M. Dufaut, and D. Wolf. Navigation of a mobile robot by locally optimal trajectories. *Robotica*, 17:553–562, 1999.
- [15] L. E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *Amer. J. Math.*, 79:497–516, 1957.
- [16] V. Duindam, X. Jijie, R. Alterovitz, S. Sastry, and K. Goldberg. Three-dimensional motion planning algorithms for steerable needles using inverse kinematics. *Int. J. Rob. Res.*, 29:789–800, June 2010.
- [17] E. Edison and T. Shima. Integrated task assignment and path optimization for cooperating uninhabited aerial vehicles using genetic algorithms. *Comput. Oper. Res.*, 38:340–356, January 2011.
- [18] S. Foldes. Decomposition of planar motions into reflections and rotations with distance constraints. CCG'04.

- [19] S. Fortune and G. Wilfong. Planning constrained motion. Annals of Math. and AI, 3:21–82, 1991.
- [20] A. A. Furtuna and D. J. Balkcom. Generalizing Dubins curves: Minimum-time sequences of body-fixed rotations and translations in the plane. *Int. J. Rob. Res.*, 29:703–726, May 2010.
- [21] P. R. Giordano and M. Vendittelli. Shortest paths to obstacles for a polygonal dubins car. *IEEE Transactions* on Robotics, 25(5):1184–1191, 2009.
- [22] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Nonholonomic Motion Planning*, 1992.
- [23] D. Kirkpatrick and V. Polishchuk. Polygonal paths of bounded curvature. In preparation, 2011.
- [24] I. Kostitsyna and V. Polishchuk. Simple wriggling is hard unless you are a fat hippo. FUN'10.
- [25] J.-C. Latombe. Robot Motion Planning. Kluwer, 1991.
- [26] Z. Li and J. F. Canny, editors. Nonholonomic Motion Planning. Kluwer, 1992.
- [27] D. Lichtenstein. Planar formulae and their uses. SIAM Journal on Computing, 11(2):329–343, 1982.
- [28] X. Ma and D. A. Castacyn. Receding horizon planning for Dubins traveling salesman problems. *Conf. Dec. and Contr.*, 2006.
- [29] A. A. Markov. Some examples of the solution of a special kind of problem on greatest and least quantities. *Soobshch, Kharkovsk. Mat Obshch*, 1:250-276, 1887 (in Russian).
- [30] F. Morbidi, F. Bullo, and D. Prattichizzo. On visibility maintenance via controlled invariance for leaderfollower dubins-like vehicles. *Conf. Dec. and Contr.*, 2008.
- [31] J. Reif and H. Wang. Non-uniform discretization for kinodynamic motion planning and its applications. WAFR'96.
- [32] J. Reif and H. Wang. The complexity of 2d curvatureconstrained shortest-path problem. WAFR'98.
- [33] P. Robuffo Giordano and M. Vendittelli. The minimumtime crashing problem for the Dubins car. SYROCO'06.
- [34] K. Savla, E. Frazzoli, and F. Bullo. On the Dubins traveling salesperson problems: Novel approximation algorithms. *Robotics: Science and Systems II*, 2006.
- [35] J. Sellen. Approximation and decision algorithms for curvature-constrained path planning: A state-space approach. WAFR'98.
- [36] M. Sigalotti and Y. Chitour. Dubins' problem on surfaces ii: Nonpositive curvature. SIAM J. Control and Optimization, 45(2):457–482, 2006.
- [37] H. J. Sussman. Shortest 3-d paths with a prescribed curvature bound. Conf. Dec. and Contr., 1995.
- [38] H. Wang and P. K. Agarwal. Approximation algorithms for curvature constrained shortest paths. SoDA'96.
- [39] G. Wilfong. Motion planning for an autonomous vehicle. *ICRA*'98.
- [40] G. Wilfong. Shortest paths for autonomous vehicles. ICRA'89.

Optimizing Budget Allocation in Graphs

Boaz Ben-Moshe*

Michael Elkin[†]

Eran Omri[‡]

Abstract

In a classical facility location problem we consider a graph G with fixed weights on the edges of G. The goal is then to find an optimal positioning for a set of facilities on the graph with respect to some objective function. We consider a new model for facility location problems, where the weights on the graph edges are not fixed, but rather should be assigned. The goal is to find the valid assignment for which the resulting weighted graph optimizes the facility location objective function.

We present algorithms for finding the optimal *budget* allocation for the center point problem and for the median point problem on trees. Our algorithms work in linear time, both for the case that a candidate vertex is given as part of the input, and for the case where finding a vertex that optimizes the solution is part of the problem. We also present an $O(\log^2(n))$ approximation algorithm for the center point problem over general metric spaces.

1 Introduction

A typical facility location problem has the following structure: the input includes a weighted set D of demand locations, a set F of feasible facility locations, and a distance function d that measures the cost of travel between a pair of locations. For each $F' \subseteq F$, the quality of F' is determined by some underlying objective function (obj). The goal is to find a subset of facilities $F' \subseteq F$, such that obj(F') is optimized (maximized or minimized). One important class of facility location problems is the *center point*, in which the goal is to find one facility in F, that minimizes the maximum distance between a demand point and the facility. Henceforth, we refer to this distance as graph radius. In another important class of problems, graph median, the goal is to find the facility in F that minimizes the average distance (i.e., the sum of the distances) between a demand point and the facility. In this paper we consider a new model for facility location on graphs, for which both problems are addressed.

1.1 The New Model

This paper suggests a new model for budget allocation problems on weighted graphs. The new model addresses optimization problems of allocating a fixed budget onto the graph edges where the goal is to find a subgraph that optimizes some objective function (e.g., minimizing the graph radius). Problems such as center point and median point on trees and graphs have been studied extensively [4, 6, 8, 9]. Yet, in most cases the input for such problems consists of a given (fixed) graph. Motivated by well-known budget optimization problems [1, 3, 5, 10] raised in the context of communication networks, we consider the graph to be a communication graph, where the weight of each edge (link) corresponds to the delay time of transferring a (fixed length) message over the link. We suggest a *Quality of Service* model for which the weight of each edge in the graph depends on the budget assigned to it. In other words, paying more for a communication link decreases its delay time.

More formally, we consider the following model: Let $G = \langle V, E \rangle$ be an undirected graph induced by some length function $\ell(e)$ for each $e \in E$. Let B be a positive budget value. Allocating a budget $\mathcal{B}(e)$ to edge $e \in E$ with length $\ell(e)$ implies that the resulting weight of e is $\frac{\ell(e)}{\mathcal{B}(e)}$. Given this weight function and a special node (root) $r \in V$, the rooted budget radius problem can be defined as follows: Divide B among the edges of E in a way that the radius of (G, ω) with respect to r is minimized, where the weight function $\omega(e)$ is given by $\frac{\ell(e)}{\mathcal{B}(e)}$, for $\mathcal{B}(e) > 0$, such that $\sum_{e \in E} \mathcal{B}(e) = B$. In the unrooted budget radius problem the goal is to divide the budget B among the edges of E in a way that the radius of (G, ω) with respect to some vertex r is minimized, where $\omega(e) = \frac{\ell(e)}{\mathcal{B}(e)}$, B(e) > 0, and $\sum_{e \in E} \mathcal{B}(e) = B$.

Analogously, one can ask to minimize the *diameter* of (G, ω) , defined as the maximum distance between a pair of vertices in (G, ω) .

We also define the *median radius* of the graph (G, ω) with respect to a designated vertex r, denoted $MR((G, \omega), r)$, as the average distance $\frac{1}{n} \cdot \sum_{v \in V} DIST_{(G,\omega)}(r, v)$ between r and other vertices of the graph $(DIST_{(G,\omega)}(r, v)$ represents the distance between vertices r and v in the graph (G, ω) . The vertex

^{*}Department of Computer Science, Ariel University Center of Samaria, Ariel 40700, Israel, benmo@g.ariel.ac.il

[†]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, elkinm@cs.bgu.ac.il

This research has been supported by the Binational Science Foundation, grant No. 2008390.

[‡]Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900, Israel, omrier@gmail.com

r with the smallest median radius, i.e., $\operatorname{MR}((G, \omega), r) = \min_{v \in V} \operatorname{MR}((G, \omega), v)$ is called the *median* of (G, ω) . The *budget median radius* of (G, ℓ) with respect to a designated vertex r and budget B, denoted $\operatorname{BMR}(G, r)$, is the minimum median radius of $(G, \omega = \frac{\ell}{B})$ with respect to r, taken over all possible budget allocations $\mathcal{B}(\cdot)$. The *budget median radius* of (G, ℓ) with respect to budget B is the minimal budget median radius of (G, ℓ) with respect to some vertex v and budget B. Finally, the vertex r that realizes the budget median radius, i.e., such that $\operatorname{BMR}((G, \ell), r) = \min_{v \in V} \operatorname{BMR}((G, \ell), v)$ is called the *budget median* of the graph (G, ℓ) .

1.2 Motivation

We were motivated by communication optimization problems in which for a fixed 'budget' one needs to design the 'best' network layout. The quality of service (QoS) of a link between two nodes depends on two main factors: i) The distance between the nodes. ii) The infra-structure of the link (between the two nodes). While the location of the nodes is often fixed and cannot be changed, the infra-structure type and service can be upgraded - it is a price-dependent service.

Quality of service is related to different parameters like, bandwidth, delay time, jitter, packet error rate and many others. Given a network graph, the desired objective is to have the best QoS for a given (fixed) budget. In this paper we focus on minimizing the maximum and the average delay time using a fixed budget.

1.3 Related Work

The problems of Center Point, Median Point on graphs (networks) have been studied extensively, see [4, 8] for a detailed surveys on *facility location*. There are various optimization problems dealing with finding the best graph; A typical graph or network improvement problem considers a graph which needs to be improved by adding the smallest number of edges in order to satisfy some constraint (e.g., maximal radius), see [1, 2, 5, 10]. Spanner graph problems [7] consider what can be seen as the inverse case of network improvement problems. In a typical spanner problem we would like to keep the smallest subset of edges from the original graph while maintaining some constraint. See [7] for a detailed survey on spanners. Observe that both network improvement and spanner graph problems can be modeled as a discrete version of our suggested new model.

1.4 Our Results

In this paper we present linear time algorithms for rooted and unrooted budget radius and budget median problems on trees. We also devise an $O(\log^2(n))$ approximation algorithm for the budget radius problem on general metric spaces.

1.5 Definitions

Let $G = \langle V, E \rangle$ be a graph with some length $\ell(e)$ for each edge $e \in E$. We next introduce some definitions and notations to define the setting of the *budget radius* problem on graphs. We consider both the case where a candidate center node to the graph is given and an optimal budget allocation is sought, and the more general case where finding the center node yielding an optimal solution is part of the problem (as well as seeking an optimal budget allocation given such a center node). To simplify our notation we omit G from the notation whenever it is clear from the context.

Let $E = \{e_1, \ldots, e_{|E|}\}$. A valid budget allocation $\mathcal{B}(\cdot)$ to E is a non-negative real function, such that $\sum_{e \in E} \mathcal{B}(e) = 1$ (here and in the rest of the paper we assume that the total budget B equals 1; this is without loss of generality since an optimal solution with budget of 1 is easily scaled to any budget B). We denote $b_i \stackrel{\text{def}}{=} \mathcal{B}(e_i)$, and for every $E' \subseteq E$ we denote $\mathcal{B}(E') = \sum_{e_i \in E'} b_i$. Given a valid budget allocation \mathcal{B} to E, the weight of an edge $e \in E$, denoted $\omega_{\mathcal{B}}(e)$, is a function of $\ell(e)$ and $\mathcal{B}(e)$. Throughout this paper we consider the case where $\omega_{\mathcal{B}}(e) \stackrel{\text{def}}{=} \frac{\ell(e)}{\mathcal{B}(e)}$.

The weighted distance between two vertices $u, v \in V$, denoted $\delta_{\mathcal{B}}(u, v)$, is the minimum weight of a simple path between u and v. Namely, $\delta_{\mathcal{B}}(u, v) \stackrel{\text{def}}{=} \min(\{\sum_{e \in P} \omega_{\mathcal{B}}(e) : P \text{ is a simple path from } u \text{ to } v\}).$

Table 1 includes the notations used in this paper.

Notation	Explanation
$G = \langle V, E \rangle$	a general undirected graph
	(induced by some metric space)
$\ell(e)$	the (a priori) length of an edge $e \in E$.
$\mathcal{B}(e)$	the budget fraction allocated to $e \in E$.
$\mathcal{B} = \{b_1,, b_{ E }\}$	an alternative notation for the function \mathcal{B} .
$\omega(e) = \frac{\ell(e)}{\mathcal{B}(e)}$	the (budget implied) weight of $e \in E$.
$G(\mathcal{B})$	the <i>budget-graph</i> implied by an allocation \mathcal{B}
$\delta_{\mathcal{B}}(u,v)$	the distance between two vertices in $G(\mathcal{B})$.

Table 1: Notations that are used throughout the paper to present the new budget graph model.

Given a valid budget allocation \mathcal{B} to E and a vertex $r \in V$, the *weighted radius* of G with respect to r is defined as $\operatorname{wr}_{\mathcal{B}}(r) = \operatorname{wr}_{\mathcal{B}}(G, r) \stackrel{\text{def}}{=} \max_{v \in V} (\delta_{\mathcal{B}}(r, v)).$

Given a graph $G = \langle V, E \rangle$ (induced by some metric) and a node $r \in V$, we define the following:

An optimal allocation for (G, r): a valid allocation for which the weighted radius from r is minimized. There may be several optimal allocations. We denote an arbitrary optimal allocation by $\mathcal{B}_r^* = \mathcal{B}_r^*(G)$ and refer to it as the optimal allocation for (G, r).

The budget radius of G with center r: denoted

BR(r) = BR(G, r), is the weighted radius of G with center r with the optimal allocation for G and r, i.e., $BR(r) = wr_{\mathcal{B}^*_{\pi}}(G, r).$

The budget radius of G: BR = BR(G) $\stackrel{\text{def}}{=} \min_{v \in V} BR(G, v)$.

An optimal allocation for G: a pair (\mathcal{B}^*, r^*) , where \mathcal{B}^* is a valid allocation to E and $r^* \in V$ is the vertex with the smallest corresponding radius, i.e., $\operatorname{wr}_{\mathcal{B}^*}(r^*) = BR$.

We demonstrate the above definitions using the following toy-example in Figure 1.5.



Figure 1: A simple example of a budget graph problem on a tree with a given center (a). Assume that each edge e has length $\ell(e) = 1$, and let x denote the fraction of the budget assigned to each of the edges (b, c) and (b, d). Observe that in order to have a valid budget allocation, it must hold that $0 < x < \frac{1}{2}$. The optimal solution minimizes the following function: $f(x) = \frac{1}{1-2\cdot x} + \frac{1}{x}$. Note that in cases where x equals $\frac{1}{3}$ or $\frac{1}{4}$ the radius is 6, while the optimal allocation of x is approximately 0.293, and the radius is approximately 5.828.

Due to space limitations some of the proofs are omitted from this extended abstract and will appear in the full version of the paper.

2 The Budget Radius Problem for Trees

Given a connected graph $G = \langle V, E \rangle$ with a length function on E, one may consider any subgraph G' of G, induced by some subset $E' \subseteq E$ and look for an optimal budget allocation for G' (i.e., BR(G')). In particular, the class of trees is an important set of such subgraphs.

Lemma 1 An optimal budget allocation (with respect to the budget radius problem) (B^*, r^*) for G, has the property that $G(\mathcal{B}^*) = \langle V, E_{\mathcal{B}^*} \rangle$, where $E_{\mathcal{B}^*} = \{e_i \in E : b_i^* > 0\}$ is a tree spanning G.

Proof. Clearly, all vertices are connected to r^* in $G(\mathcal{B}^*)$. Assume towards a contradiction that $G(\mathcal{B}^*)$ contains a cycle. Let T_D be the tree of shortest paths obtained by invoking the Dijkstra algorithm on $G(\mathcal{B}^*)$ and r^* . Hence, there is a an edge $e_i \in E_{\mathcal{B}^*}$ (i.e., $b_i^* > 0$) such that e does not appear in any shortest path from r^* to any vertex in V. Thus, we can obtain a better budget allocation by (say, equally) dividing the budget

portion allocated to e_i among all edges in T_D . This is a contradiction to the optimality of \mathcal{B}^* .

We note that the above is not true for the problem of the minimum budget diameter of a graph (see figure 2).



Figure 2: (a) Three points in the plane: x, y, z are the nodes of a unite equilateral triangle. (b) Tree: optimal radius (2), non optimal diameter (4). (c) Cycle graph: non-optimal radius (3), optimal diameter (3).

The above lemma suggests that it is interesting to consider the Budget Radius problem for the subclass of trees. In the sequel, we present an algorithm solving this problem. We first consider the case where a designated center node r is given as a part of the input, and an optimal budget allocation \mathcal{B}^* is sought. We use the standard terminology and refer to r as the root of the tree (rather than, the center). Thereafter, we consider the general case in trees, where the problem is to find a pair (\mathcal{B}^*, r^*) minimizing the budget radius of the tree.

2.1 The Budget Radius for a Rooted Tree

We next consider two possible structures for rooted trees that will later be the basis for our recursive construction of an optimal valid budget allocation to the edges of a given tree. First, we consider a tree in which the root has only a single child.

Lemma 2 Let T be a tree rooted at r, with some length function ℓ on the edges of T. Assume r has a single child r' (the root of the subtree T'), and let R' = BR(T', r')and $d_1 = \ell(r, r')$. Then, an optimal budget allocation \mathcal{B}^* assigns to the edge $e_1 = (r, r')$ a fraction $b_1^* = \frac{\sqrt{d_1}}{\sqrt{R'} + \sqrt{d_1}}$. It follows that $BR(T, r) = \frac{d_1}{b_1^*} + \frac{R'}{1-b_1^*}$.

Proof. Let E be the set of edges in T and $E' = E \setminus \{e_1\}$ be the set of edges of T' (see Figure 2.1-a). Given any valid budget allocation \mathcal{B} to E, let \mathcal{B}' be the scaling of the restriction of \mathcal{B} to E', defined by $\mathcal{B}'(e') = \frac{\mathcal{B}(e')}{1-\mathcal{B}(e_1)}$ for every $e' \in E'$. Note that with this scaling, \mathcal{B}' is a valid budget allocation to E', i.e., $\sum_{e' \in E'} \mathcal{B}'(e') = 1$. Since any path from r to any leaf of T must start with the edge $e_1 = (r, r')$, it follows that

$$\operatorname{wr}_{\mathcal{B}}(r) = \frac{d_1}{b_1} + \frac{\operatorname{wr}_{\mathcal{B}'}(r')}{1 - b_1}.$$

Hence, for \mathcal{B} to be optimal for T with root r, we must have \mathcal{B}' be optimal for T' and r'. In addition, given R' = BR(T', r'), the budget radius of T with root r is obtained by finding b_1 that minimizes the function $wr_{\mathcal{B}}(r) = \frac{d_1}{b_1} + \frac{R'}{1-b_1}$. Therefore, it follows that $BR(T, r) = \frac{d_1}{b_1} + \frac{R'}{1-b_1}$ for $b_1 = \frac{\sqrt{d_1}}{\sqrt{R'} + \sqrt{d_1}}$. \Box

We next consider a more general tree structure (Figure 2.1-b). Let T = (V, E) be a tree, rooted at r, such that r has k children r_1, r_2, \ldots, r_k , where r_i is the root of the subtree $T_i = (V_i, E_i)$. Denote by $T'_i = (V'_i, E'_i)$ the subtree of T, rooted at r and containing T_i . Formally, $V'_i = V_i \cup \{r\}$, and $E'_i = E_i \cup \{(r, r_i)\}$. Clearly, the edges sets E'_i s are disjoint. Given a valid budget allocation \mathcal{B} to E, for each index $i \in \{1, 2, ..., k\}$ denote by $l_i \in V_i$ the leaf l for which $\delta_{\mathcal{B}}(r, l)$ is the largest within T'_i . Recall that the weighted radius $wr_{\mathcal{B}}(r)$ is determined by the maximum weighted distance to some l_i . In other words, $wr_{\mathcal{B}}(r) = \max_{1 \le i \le k} (\delta_{\mathcal{B}}(r, l_i))$. Next, we show that in any optimal budget allocation \mathcal{B}^* for such T, the fraction of the budget assigned to the edges of each subtree T'_i is directly correlated to its relative weighted radius.



Figure 3: (a) The case that r has only a single child. (b) The general case.

Lemma 3 Let T = (V, E) be a tree rooted at r, with some length function ℓ on the edges of T. Assume that r has k children r_1, r_2, \ldots, r_k where r_i is the root of the subtree $T_i = (V_i, E_i)$, and let \mathcal{B}^* be an optimal budget allocation to E. For each $1 \leq i \leq k$, let T'_i and l_i be as in the foregoing discussion (i.e., l_i is maximal in T'_i with respect to $\delta_{\mathcal{B}^*}(r, \cdot)$). It then holds for all $1 \leq i, j \leq k$ that $\delta_{\mathcal{B}^*}(r, l_i) = \delta_{\mathcal{B}^*}(r, l_j)$.

Proof. Assume that for some $1 \leq i, j \leq k$, it holds that $\delta_{\mathcal{B}^*}(r, l_i) > \delta_{\mathcal{B}^*}(r, l_j)$. We show that it is then possible to present a better budget allocation for T, which, in turn, leads to a contradiction. Let $\rho = \frac{\delta_{\mathcal{B}^*}(r, l_j)}{\delta_{\mathcal{B}^*}(r, l_i)}$ and consider an alternative budget allocation in which each edge e in E'_j were assigned a ρ fraction of its current budget, i.e. $\rho \cdot \mathcal{B}^*(e)$ (while assignment to all other edges stays the same as before). The length of each path from r to a leaf in T'_j would be multiplied by $1/\rho$. Hence, the maximum distance from r to any leaf in the T'_j would be at most $\delta_{\mathcal{B}^*}(r, l_i)$. This allocation is therefore as good as \mathcal{B}^* (with respect to the weighted radius) although

the sum of assigned values is not 1, but rather, $1 - (1 - \rho) \cdot \mathcal{B}^*(E'_j)$. Turning it into a valid budget allocation by dividing the remaining $(1 - \rho)\mathcal{B}^*(E'_j)$ budget equally among all edges in E, we obtain a better valid budget allocation to E. That is, we fix a new allocation \mathcal{B}' by setting $\mathcal{B}'(e) = \rho \cdot \mathcal{B}^*(e) + \frac{(1 - \rho)\mathcal{B}^*(E'_j)}{|E|}$ if $e \in E'_j$, and $\mathcal{B}'(e) = \mathcal{B}^*(e) + \frac{(1 - \rho)\mathcal{B}^*(E'_j)}{|E|}$ otherwise. We note that \mathcal{B}' may not be an optimal allocation, however it is a contradiction to the optimality of \mathcal{B}^* .

The following corollary describes how any optimal valid budget allocation must divide the budget among the disjoint sets of edges of the subtrees T'_i .

Corollary 4 Let T be a tree as above. Then in any optimal budget allocation \mathcal{B}^* to E it holds that $\mathcal{B}^*(E'_i) = \frac{\operatorname{BR}(T'_i,r)}{\sum_{j=1}^k \operatorname{BR}(T'_j,r)}$. Thus, an optimal solution in this case is given by $\operatorname{BR}(T,r) = \sum_{j=1}^k \operatorname{BR}(T'_j,r)$.

Theorem 5 Given a tree T rooted at r, it is possible to find an optimal valid budget allocation for T and r, in linear time in the size of T.

Proof. T is a rooted tree, thus an inductive construction is only natural. First, assume T is a single node r. In this case, no budget is needed and BR(T, r) = 0. Assume T is rooted at r, such that r has k children r_1, r_2, \ldots, r_k . Denote by T_i the subtree of T rooted at r_i , and containing all vertices (and edges) of the subtree rooted at r_i (and only these vertices). Denote by $T'_i = (V'_i, E'_i)$ the subtree of T rooted at r, induced by adding the edge (r, r_i) to T_i . Formally, $V'_i = V_i \cup \{r\}$, and $E'_i = E_i \cup \{(r, r_i)\}$. Thus, each T'_i is a rooted tree where the root (r) has a single child, and no T'_i, T'_j for $i \neq j$ share any vertex other than r and E'_i, E'_j are disjoint for all $i \neq j$.

By Corollary 4, if we know $BR(T'_i, r)$ for all $1 \le i \le k$, we can derive an optimal valid budget allocation for T, r. In order to obtain a $BR(T'_i, r)$, it suffices to have an optimal solution for the subtree of r_i , which, by the induction hypothesis can be done (using Lemma 2).

Note, that we evaluate the optimal solution for every subtree of every vertex in T exactly once and thus the procedure requires in linear time.

The following lemma proves helpful in the sequel, but is interesting in its own right. It captures some of the tricky nature of the budget radius problem, as it shows the connection between two seemingly unrelated quantities. The first is the weight of a minimum spanning tree (MST) of a given graph and the second is the optimal solution for the budget radius problem for that graph.

Lemma 6 Given a tree T = (V, E) rooted at r, with some length function ℓ on E, the budget radius of Tis at least the sum of lengths of the edges of T, i.e., $BR(T,r) \ge \sum_{e \in E} \ell(e)$.
Proof. We prove the above lemma by induction. If T has no edges, then both values are 0. If r has only one child r' (the root of the subtree T'), then by Lemma 2, since any optimal allocation \mathcal{B}^* must assign $\mathcal{B}^*((r,r')) > 0$, we have $\operatorname{BR}(T,r) > \ell((r,r')) + \operatorname{BR}(T',r')$, which, by the induction hypothesis is at least $\sum_{e \in E} \ell(e)$. Otherwise, assume r has k children $(r_1 \ldots r_k)$ and denote T_i the subtree induced by r and the vertices of the subtree of r_i . By Corollary 4 $\operatorname{BR}(T,r) = \sum_{i=1}^k \operatorname{BR}(T_i,r)$. Hence, by the induction hypothesis, the lemma follows.

2.2 The Budget Radius for Unrooted Trees

In this section we consider the budget radius problem for unrooted trees, i.e., where the root of the tree is not given as part of the input. Clearly, one can invoke the algorithm from Theorem 5 with every vertex v as a candidate center vertex r, and select the vertex v for which BR(T, v) is minimal as the ultimate center. However, this naive algorithm requires $O(n^2)$ time. We next show how to construct a linear time algorithm for this problem (indeed, for a tree T, our algorithm computes BR(T, v) for every v in T). Intuitively, this protocol uses the fact that given BR(T, r) and the partial computations made by algorithm of Theorem 5, applied to the T and r, it possible to compute in constant time BR(T, v) for every neighbor v of r. This intuition is formalized in Lemma 7.

Lemma 7 Let T = (V, E) be a tree rooted at r, with some length function ℓ on E. Let $v \in V$ be a neighbor (a child) of r. Denote by $T_v = (V_v, E_v)$ the subtree of v, and denote by T'_v the subtree of v augmented by the edge e = (r, v) (i.e., $T'_v = (V_v \cup r, E_v \cup e)$). It is possible to compute, in constant time, BR(T, v) given BR(T, r), BR (T_v, v) , and BR (T'_v, r) , see Figure 2.2.

Proof. Denote by \hat{T} the tree obtained by omitting T_v from T, formally, $\hat{T} = (\hat{V}, \hat{E})$, where $\hat{V} = V \setminus (V_v \setminus \{v\})$ and $\hat{E} = E \setminus E_v$. In addition, denote by \hat{T}' the tree obtained by omitting the edge e = (r, v) from \hat{T} , i.e., $\hat{T}' = (\hat{V} \setminus \{v\}, \hat{E} \setminus \{e\})$.

It can be easily derived from Corollary 4 that $BR(T, v) = BR(T_v, v) + BR(\hat{T}, v)$. By Lemma 2, we can compute $BR(\hat{T}, v)$ from $BR(\hat{T}', r)$ and $\ell(e)$, in constant time. Finally, we compute $BR(\hat{T}', v)$, using Corollary 4 again, to obtain $BR(\hat{T}', r) = BR(T, r) - BR(T'_v, r)$. \Box

Roughly, our algorithm will traverse the tree twice. First, we traverse the tree, computing the algorithm of Theorem 5 for an arbitrary root r (say, $r = v_1$). Recall that this algorithm traverses the tree in a bottom-up fashion, i.e., from the leaves to the root, and that an optimal solution for each vertex is calculated, with respect to the subtree below it. Thereafter, we traverse the tree in a top-down fashion, while for each vertex



Figure 4: (a) The original tree rooted at r. (b) Considering v as the root of T'_v . (c) The tree \hat{T} .



Figure 5: (a) The original tree rooted at r. (b) Considering v as the root of the tree. Computation of an optimal budget allocation for the tree rooted at v can be done in constant time, given an optimal budget allocation for the tree rooted at r.

v that is a child of v', we compute an optimal budget radius for the tree with root v, given an optimal budget radius for the tree with root v' and the information stored in v from the first traversal.

Theorem 8 Given a tree T = (V, E) with some length function ℓ on E, it is possible to compute an optimal allocation for T, i.e., a pair (\mathcal{B}^*, r^*) , such that $wr_{\mathcal{B}^*}(r^*) = BR(T)$. Furthermore, this can be done in linear time in the size of T.

3 Budget Radius – The General Case

In this section we consider the general case problem of optimizing the budget radius for a complete graph over n vertices, induced by some metric space M = (V, d). We present an $O(\log^2(n))$ approximation algorithm for this problem. We start by showing that a naive Minimum Spanning Tree (MST) heuristic may lead to an $O(n^{0.5})$ approximation factor. Assume we have n points on a square uniform grid. Its MST may have a path like shape, with $\Omega(n)$ radius. Hence its budget radius is $\Omega(n^2)$. On the other hand, each of the n points may be connected to the center with a path of length $O(n^{0.5})$. Hence, the budget radius of this metric is $O(n^{1.5})$.

3.1 The Special Case of a Line

We first consider a setup in which M is defined by some n points all residing on the interval [0, 1], where for any two points p_1, p_2 within this interval, $d(p_1, p_2)$ is the Euclidean distance between p_1 and p_2 . Let G = (V, E) be the complete graph induced by M. We present a



Figure 6: Using an MST-like heuristic may lead to an $O(n^{0.5})$ approximation ratio with respect to the *center* point problem (minimum radius). (a) A grid based set of points. (b) A path-like MST. (c) A solution with radius $O(n^{0.5})$.

valid budget allocation \mathcal{B} to E with budget radius at most $\log^2 n$ and such that the graph induced by $\{e : \mathcal{B}(e) > 0\}$ is a tree spanning V.

Lemma 9 Let G = (V, E) be the complete graph described above, then BR $(G) \leq \log^2 n$.

3.2 General Complete (Metric) Graphs

We next define an approximation algorithm \mathcal{A} , such that given a complete graph G = (V, E), induced by some metric space M = (V, d), approximates the Budget Radius problem for G by a factor of $O(\log^2 n)$. Assume that a minimum spanning tree for G has a total weight LB, we proceed as follows:

- 1. Find an Hamiltonian path (HP) visiting all nodes with weight no more than $2 \cdot LB$.
- 2. Let G' be the result of unfolding HP to a straight line, i.e., G' is defined by n points, situated on an interval, such that, the distance between every two points is the length of the path between them on the Hamiltonian path HP (specifically, the length of the whole interval is exactly the length of HP).
- 3. Scale the above (HP) interval length to 1.
- 4. Build a balanced binary search tree (BT) over G'.
- 5. Apply the algorithm of Theorem 5 to BT. Assign the appropriate budget to all edges in BT and 0 to all other edges in E.

Theorem 10 Let G = (V, E) be a complete graph induced by some metric space M = (V, d). Then, algorithm \mathcal{A} results in a valid budget allocation to the edges of E that approximates BR(G) by a $2\log^2(n)$ factor.

Proof. First note that finding an Hamiltonian path (HP) with weight no more than 2·LB is feasible using an MST for G. More importantly, note that by Lemma 6,

it holds that LB is a lower bound on the optimal solution (i.e., on BR(G)). This is true since an optimal budget allocation defines a tree (see Lemma 1), which has a total weight of at least LB (by the minimality of an MST). Thus, algorithm \mathcal{A} yields an optimal budget allocation for BT, which by Lemma 9 yields a budget radius of at most $2 \cdot \text{LB} \cdot \log^2(n) \leq 2 \cdot \text{BR}(G) \cdot \log^2(n)$. \Box

4 Conclusion and Future Work

The paper introduces a new model for optimization problems on graphs. The suggested *budget* model was used to define facility location problems such as center and median point. For the tree case, optimal algorithms are presented for both aforementioned problems. For the general metric center point problem, an $O(\log^2(n))$ approximation algorithm is presented. The new model raises a set of open problems e.g.,: i) Hardness: is the budget center point problem on general graphs NPhard¹. ii) Facility location: Find approximation algorithms for the k-center, k-median, and 1-median on general graphs. iii) Graph optimization: minimizing the diameter of the graph.

References

- A. M. Campbell, T. J. Lowe, and L. Zhang. Upgrading arcs to minimize the maximum travel time in a network. *Netw.*, 47(2):72–80, 2006.
- [2] V. Chepoi, H. Noltemeier, and Y. Vaxès. Upgrading trees under diameter and budget constraints. *Networks*, 41(1):24–35, 2003.
- [3] Victor Chepoi and Yann Vaxès. Augmenting trees to meet biconnectivity and diameter constraints. Algorithmica, 33(2):243-262, 2002.
- [4] M. S. Daskin. Network and Discrete Location: Models, Algorithms, and Applications. Wiley-Interscience, 1995.
- [5] S. T. McCormick Li, C. L. and D. Simchi-Levi. On the minimum-cost-bounded diameter and the fixed-budgetminimum-diameter edge addition problems. *Operations Research Letters*, 11:303–308, 1992.
- [6] N. Megiddo. The weighted Euclidean 1-center problem. Math. Oper. Res., 8(4):498–504, 1983.
- [7] G. Narasimhan and M. Smid. Geometric Spanner Networks. Cambridge University Press, 2007.
- [8] S. Nickel and J. Puerto. Location Theory: A Unified Approach. Springer, 2005.
- [9] A. Tamir. Improved complexity bounds for center location problems on networks by using dynamic data structures. SIAM J. Discret. Math., 1(3):377–396, 1988.
- [10] J. Z. Zhang, X. G. Yang, and M. C. Cai. A network improvement problem under different norms. *Comput. Optim. Appl.*, 27(3):305–319, 2004.

¹The discrete version of the center point budget problem is weakly NP-complete even on a path like graph (reduced to *perfect partition problem*).

Bottleneck Steiner Tree with Bounded Number of Steiner Vertices

A. Karim Abu-Affash*

Paz Carmi[†]

Matthew J. Katz[‡]

Abstract

Given a complete graph G = (V, E), where each vertex is labeled either terminal or Steiner, a distance function $d: E \to \mathbb{R}^+$, and a positive integer k, we study the problem of finding a Steiner tree T spanning all terminals and at most k Steiner vertices, such that the length of the longest edge is minimized. We first show that this problem is NP-hard and cannot be approximated within a factor $2 - \varepsilon$, for any $\varepsilon > 0$, unless P = NP. Then, we present a polynomial-time 2-approximation algorithm for this problem.

1 Introduction

Given an arbitrary weighted graph G = (V, E) with a distinguished subset $R \subseteq V$ of vertices, a *Steiner tree* is an acyclic subgraph of G spanning all vertices of R. The vertices of R are usually referred to as *terminals* and the vertices of $V \setminus R$ as *Steiner* vertices. The *Steiner tree* (ST) problem is to find a Steiner tree T such that the total length of the edges of T is minimized. This problem has been shown to be NP-complete [4, 10], even in the Euclidean or rectilinear version [11]. Arora [3] gave a PTAS for the Euclidean and rectilinear versions of the ST problem. For arbitrary weighted graphs, many approximation algorithms have been proposed [6, 7, 12, 15, 17, 18].

The bottleneck Steiner tree (BST) problem is to find a Steiner tree T such that the bottleneck (i.e., the length of the longest edge) of T is minimized. Unlike the ST problem, the BST problem can be solved exactly in polynomial time [19]. Both the ST and BST problems have many important applications in VLSI design, transportation and other networks, and computational biology [8, 9, 13, 14].

The k-Bottleneck Steiner Tree (k-BST) problem is a restricted version of the BST problem, in which there is a limit on the number of Steiner vertices that may be used in the constructed tree. More precisely, given a graph G = (V, E) and a subset $R \subseteq V$ of terminals, a distance function $d: E \to \mathbb{R}^+$, and a positive integer k, one has to find a Steiner tree T with at most k Steiner vertices such that the bottleneck of T is minimized.

A geometric version of the k-BST problem has been studied in [20]. In this version, we are given a set Pof n terminals in the plane and an integer k > 0, and we are asked to place at most k Steiner points, such that the obtained Steiner tree has bottleneck as small as possible. Wang and Du [20] showed that the problem is NP-hard to approximate within a factor of $\sqrt{2}$. The best known approximation ratio is 1.866 [21]. Bae et al. [5] presented an $\mathcal{O}(n \log n)$ -time algorithm for the problem for k = 1 and an $\mathcal{O}(n^2)$ -time algorithm for k = 2. Li et al. [16] presented a $(\sqrt{2} + \varepsilon)$ -approximation algorithm with inapproximability within $\sqrt{2}$ for a special case of the problem where edges between two Steiner points are not allowed.

Recently, Abu-Affash [1] studied the k-BST problem with the additional requirement that all terminals in the computed Steiner tree must be leaves. He presented a hardness result for the problem, as well as a polynomialtime approximation algorithm with performance ratio 4. In [2], the authors considered the following related problem. Given a set P of n points in the plane and two points $s, t \in P$, locate k Steiner points, so as to minimize the bottleneck of a bottleneck path between sand t. They showed how to solve this problem optimally in time $\mathcal{O}(n \log^2 n)$.

In this paper, we show that the k-BST problem is NP-hard and that it cannot be approximated to within a factor of $2 - \varepsilon$. We also present a polynomial-time 2-approximation algorithm for the problem.

2 Hardness Result

Given a complete graph G = (V, E) with a distinguished subset $R \subseteq V$ of terminals, a distance function $d : E \rightarrow \mathbb{R}^+$, and a positive integer k, the goal in the k-BST problem is to find a Steiner tree with at most k Steiner vertices and bottleneck as small as possible. In this section we prove a lower bound on the approximation

^{*}Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, abuaffas@cs.bgu.ac.il. Partially supported by the Lynn and William Frankel Center for Computer Sciences, by the Robert H. Arnow Center for Bedouin Studies and Development, by a fellowship for outstanding doctoral students from the Planning & Budgeting Committee of the Israel Council for Higher Education, and by a scholarship for advanced studies from the Israel Ministry of Science and Technology.

[†]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, carmip@cs.bgu.ac.il. Partially supported by a grant from the German-Israeli Foundation.

[‡]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, matya@cs.bgu.ac.il. Partially supported by grant 1045/10 from the Israel Science Foundation, and by the Israel Ministry of Industry, Trade and Labor (consortium CORNET).

ratio of polynomial-time approximation algorithms for the problem.

Theorem 1 It is NP-hard to approximate the k-BST problem within a factor $2 - \varepsilon$, for any $\varepsilon > 0$.

Proof. We present a reduction from connected vertex cover in planar graphs, which is known to be NP-complete [11].

Connected vertex cover in planar graphs: Given a planar graph G = (V, E) and an integer k, does there exist a vertex cover V^* of G, such that $|V^*| \leq k$ and the subgraph of G induced by V^* is connected?

Given a planar graph G = (V, E) and an integer k, we construct a complete graph G' = (V', E') with an appropriate distance function and appropriate integer k', such that G has a connected vertex cover of size at most k if and only if there exists a Steiner tree T in G'with at most k' Steiner vertices and bottleneck at most $(2 - \varepsilon)$, for some $\varepsilon > 0$.

Let $V = \{v_1, v_2, \ldots, v_n\}$ and let $E = \{e_1, e_2, \ldots, e_m\}$. For each edge $e = (v_i, v_j) \in E$, we add a vertex $t_{i,j}$ (e.g., at the middle of e) and connect it to both v_i and v_j . Let $R = \{t_{i,j} : (v_i, v_j) \in E\}$ and let $E'_1 = \{(v_i, t_{i,j}), (t_{i,j}, v_j) : (v_i, v_j) \in E\}$. We set $V' = V \cup R$, where V is the set of Steiner vertices and R is the set of terminals; see Figure 1. Let G' = (V', E') be the complete graph over V'. For each edge $e \in E'$, we assign length d(e) = 1, if $e \in E'_1$, and d(e) = 2, otherwise. Finally, we set k' = k.



Figure 1: (a) A planar graph G = (V, E), and (b) the vertices of G': circles indicate Steiner vertices and grey squares indicate terminals.

Now, we prove the correctness of the reduction. Clearly, if G has a connected vertex cover V^* with $|V^*| \leq k$, then, by selecting the Steiner vertices of V'corresponding to the vertices in V^* , we can construct a Steiner tree T with at most k' = k Steiner vertices, such that the length of each edge in T is exactly 1.

Conversely, suppose that there exists a Steiner tree Tin G' with at most k' Steiner vertices and bottleneck at most $2 - \varepsilon$. Let V^* be the subset of vertices of V that belong to T. By the construction, any two terminals are connected in E' by an edge of length 2. Thus, we deduce that each terminal is connected in T to a Steiner vertex in V^* . Since T is connected and each edge in Ecorresponds to one terminal in V', we conclude that V^* is a connected vertex cover of G, and its size is at most k = k'.

3 2-Approximation Algorithm

In this section, we design a polynomial-time approximation algorithm for computing a Steiner tree with at most k Steiner vertices (k-ST for short), such that its bottleneck is at most twice the bottleneck of an optimal (minimum-bottleneck) k-ST.

Let G = (V, E) be the complete graph with n vertices, let $R \subseteq V$ be the set of terminals, and let $d : E \rightarrow \mathbb{R}^+$ be a distance function. Let e_1, e_2, \ldots, e_m , where $m = \binom{n}{2}$, be the edges of G sorted by length, that is, $d(e_1) \leq d(e_2) \leq \cdots \leq d(e_m)$. Clearly, the bottleneck of an optimal k-ST is the length of an edge in E. For an edge $e_i \in E$, let $G_i = (V, E_i)$ be the graph obtained from G by deleting all edges of length greater than $d(e_i)$, that is, $E_i = \{e_i \in E : d(e_i) \leq d(e_i)\}$.

The idea behind our algorithm is to devise a procedure that, for a given edge $e_i \in E$, does one of the following:

- (i) It either constructs a k-ST in G with bottleneck at most $2d(e_i)$, or
- (ii) it returns the information that G_i does not contain a k-ST.

Let $e_i \in E$. For two terminals $p, q \in R$, let $\delta_i(p, q)$ be a path between p and q in G_i with minimum number of Steiner vertices. Let $G_R = (R, E_R)$ be the complete graph over R. For each edge $(p,q) \in E_R$, we assign a weight w(p,q) equal to the number of Steiner vertices in $\delta_i(p,q)$. Let T be a minimum spanning tree of G_R under w, and put $C(T) = \sum_{e \in T} \lfloor w(e)/2 \rfloor$. The following observation follows from Lemma 3 in [20].

Observation 1 For any spanning tree T' of G_R , $C(T) \leq C(T')$.

Lemma 2 If G_i contains a k-ST, then $C(T) \leq k$.

Proof. Let T^* be a k-ST in G_i . A Steiner tree is *full* if all its terminals are leaves. It is well known that every Steiner tree can be decomposed into a collection of full Steiner trees, by splitting each of the non-leaf terminals.

We begin by decomposing T^* into a collection of full Steiner trees. Next, for each full Steiner tree T_j^* in the collection, we construct in G_R a spanning tree T'_j of the terminals of T_j^* , such that the union of these trees is a spanning tree T' of G_R and $C(T') \leq k$. Finally, by Observation 1, we conclude that $C(T) \leq k$.

We now describe how to construct T'_j from T^*_j . Arbitrarily select one of the Steiner vertices as the root of T^*_j ; see Figure 2(a). The construction of T'_j is done by an iterative process applied to T^*_j . In each iteration, we select a deepest terminal p, among the terminals of the current rooted tree that have not yet been processed. From p we move up the tree until we reach a Steiner vertex s that has terminal descendants other than p. Let $q, q \neq p$, be a terminal descendant of s that is closest to s. We connect p to q by an edge of weight equal to the number of Steiner vertices between p and q in T^*_j , and remove the Steiner vertices between p and s (not including s). After processing all terminals but one, we remove all remaining Steiner vertices.



Figure 2: (a) The rooted tree T_j^* , and (b) the construction of T_j' .

In the example in Figure 2(b), we first select terminal a, which is the deepest one, connect it to terminal b by an edge of weight 3, and remove the vertices s_1 and s_2 . Next, we select terminal c, connect it to terminal d by an edge of weight 1, and do not remove any Steiner vertex. Next, we select terminal d, connect it to terminal h by an edge of weight 2, and remove the vertex s_3 . In the last iteration, we select terminal b, connect it to terminal h by an edge of weight 3 and remove the vertex s_4 . We can now remove all of the remaining Steiner vertices.

Clearly, the union T' of the trees T'_j is a spanning tree of G_R . We show below that $C(T') \leq k$. Notice that in each iteration during the construction of T'_j , if the weight of the added edge e is w(e), then we remove at least $\lfloor w(e)/2 \rfloor$ Steiner vertices from T^*_j . This implies that $C(T'_j) = \sum_{e \in T'_j} \lfloor w(e)/2 \rfloor \leq k_j$, where k_j is the number of Steiner vertices in T^*_j , and, therefore, $C(T') = \sum_j C(T'_j) \leq k$. \Box

We now present our 2-approximation algorithm. We consider the edges of E, one by one, by non-decreasing length. For each edge $e_i \in E$, we construct a minimum spanning tree T of $G_R = (R, E_R)$, using the weight function w induced by G_i , and check whether $C(T) \leq k$. If so, we construct a k-ST in G with bottleneck at most $2d(e_i)$, otherwise, we proceed to the next edge e_{i+1} .

Algorithm 1	BST(G =	= (V,	$\overline{E}), R,$	k)			
1: Let $e_1, e_2,$	\ldots, e_m be	the	edges	of E	sorted $\$	by	non-

- decreasing length 2: $G_R = (R, E_R) \leftarrow$ the complete graph over R
- 3: $C(T) \leftarrow \infty$
- 4: $i \leftarrow 0$
- 5: while C(T) > k do
- 6: $i \leftarrow i + 1$
- 7: construct the graph G_i
- 8: for each edge $(p,q) \in E_R$ do
- 9: $w(p,q) \leftarrow$ the number of Steiner vertices in $\delta_i(p,q)$
- 10: construct a minimum spanning tree T of G_R under w
- 11: $C(T) \leftarrow \sum_{e \in T} \lfloor w(e)/2 \rfloor$ 12: $Construct-k-ST(T,G_i)$

The construction of a k-ST (line 12 in the algorithm above) is done as follows. For each edge $e = (p,q) \in T$, we select at most $\lfloor w(e)/2 \rfloor$ Steiner vertices from the path $\delta_i(p,q)$, to obtain a path connecting between pand q with at most this number of Steiner vertices and bottleneck at most $2d(e_i)$; see Figure 3. Clearly, the obtained Steiner tree contains at most k Steiner vertices and its bottleneck is at most $2d(e_i)$.

Lemma 3 The algorithm above constructs a k-ST in G with bottleneck at most twice the bottleneck of an optimal k-ST.

Proof. Let e_i be the first edge satisfying the condition $C(T) \leq k$. Then, by Lemma 2, the bottleneck of any k-ST in G is at least $d(e_i)$, and, therefore, the constructed k-ST has a bottleneck at most twice the bottleneck of an optimal k-ST.



Figure 3: The constructed k-ST consists of the squares, solid circles and dotted edges.

The following theorem summarizes the main result of this section.

Theorem 4 There exists a polynomial-time 2approximation algorithm for the k-BST problem.

References

- A.K. Abu-Affash. On the Euclidean bottleneck full Steiner tree problem. In Proceedings of the 27th ACM Symposium on Computational Geometry (SoCG '11), pages 433–439, 2011.
- [2] A.K. Abu-Affash, P. Carmi, M.J. Katz, and M. Segal. The Euclidean bottleneck Steiner path problem. In *Proceedings of the 27th ACM Symposium* on Computational Geometry (SoCG '11), pages 440–447, 2011.
- [3] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45:735–782, 1998.
- [4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS '92)*, pages 14–23, 1992.
- [5] S.W. Bae, C. Lee, and S. Choi. On exact solutions to the Euclidean bottleneck Steiner tree problem. *Information Processing Letters*, 110:672–678, 2010.
- [6] P. Berman and V. Ramaiyer. Improved approximation for the Steiner tree problem. *Journal of Algorithms*, 17:381–408, 1994.
- [7] A. Borchers and D.Z. Du. The k-Steiner ratio in graphs. SIAM Journal on Computing, 26:857–869, 1997.
- [8] X. Cheng and D.Z. Du. Steiner Tree in Industry. Kluwer Academic Publishers, Dordrecht, Netherlands, 2001.

- [9] D.Z. Du, J.M. Smith, and J.H. Rubinstein. Advances in Steiner Tree. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.
- [10] M.R. Garey, R.L. Graham, and D.S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal of Applied Mathematics*, 32(4):835– 859, 1977.
- [11] M.R. Garey and D.S. Johnson. The rectilinear Steiner tree problem is NP-complete. SIAM Journal of Applied Mathematics, 32(4):826–834, 1977.
- [12] S. Hougardy and H.J. Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. In Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00), pages 448–453, 1999.
- [13] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics, Amsterdam, 1992.
- [14] A.B. Kahng and G. Robins. On Optimal Interconnection for VLSI. Kluwer Academic Publishers, Dordrecht, Netherlands, 1995.
- [15] M. Karpinski and A. Zelikovsky. New approximation algorithms for the Steiner tree problem. *Journal of Combinatorial Optimization*, 1(1):47– 65, 1997.
- [16] Z.-M. Li, D.-M. Zhu, and S.-H. Ma. Approximation algorithm for bottleneck Steiner tree problem in the Euclidean plane. *Journal of Computer Science and Technology*, 19(6):791–794, 2004.
- [17] H.J. Prömel and A. Steger. A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89– 101, 2000.
- [18] G. Robbins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. SIAM Journal on Discrete Mathematics, 19(1):122–134, 2005.
- [19] M. Sarrafzadeh and C.K. Wong. Bottleneck Steiner trees in the plane. *IEEE Transactions on Comput*ers, 41(3):370–374, 1992.
- [20] L. Wang and D.-Z. Du. Approximations for a bottleneck Steiner tree problem. *Algorithmica*, 32:554– 561, 2002.
- [21] L. Wang and Z.-M. Li. Approximation algorithm for a bottleneck k-Steiner tree problem in the Euclidean plane. *Information Processing Letters*, 81:151–156, 2002.

Connecting Two Trees with Optimal Routing Cost *

Mong-Jen Kao[¶]

Bastian Katz[§]

Marcus Krug \S D.T. Lee[¶]

Martin Nöllenburg^{§‡}

Ignaz Rutter[§]

Dorothea Wagner[§]

Abstract

We study the problem of finding the optimal connection between two disconnected vertex-weighted trees. We are given a distance function on the vertices and seek to minimize the routing cost of the tree resulting from adding one single edge between the two trees. The routing cost is defined as the sum of the weighted distances between all pairs of vertices in the induced tree-metric. The problem arises when augmenting and/or repairing communication networks or infrastructure networks.

We present an asymptotically optimal quadratic-time algorithm for the general case and show that the problem can be solved more efficiently for the Euclidean metric, when vertices are mapped to points in the plane, as well as for compactly representable graph metrics.

1 Introduction

In the construction of communication and infrastructure networks we often have to find a reasonable balance between the cost for establishing the links between the vertices in the network and the performance of the network in terms of various quality measures, such as routing cost, connectivity and diameter. While the cost should be minimized and increases with each established link, the performance of the network should be maximized and typically improves when more links are added. This tradeoff can be formalized in different ways. However, motivated by practical applications of this problem it is quite common to assume that we are given a limited budget for the construction cost and wish to optimize the performance of the network subject to this constraint.

The *Optimal Network Problem*, which has been introduced by Scott [8], addresses the problem of optimizing

§Faculty of Informatics, Karlsruhe Institute of Technology (KIT), firstname.lastname@kit.edu the *routing cost* of a network, defined by the sum of the shortest paths between all pairs of vertices in the graph. Due to its importance for communication networks, this problem has received considerable attention, among others by Dionne and Florian [1] and Wong [10].

If the budget for establishing the links in a network is rather tight, a tree is often the only affordable infrastructure. However, Johnson et al. [6] prove that the optimal network problem is NP-complete, even if all edges have the same length and the network must be a tree. This problem is also called the *Minimum Routing Cost Spanning Tree Problem (MRCST)*. More recently, Wu et al. presented an FPTAS for this problem [11] and Fischetti et al. [3] studied exact algorithms for computing the minimum routing cost spanning tree.

We consider the related problem of connecting a disconnected tree-network by adding a missing edge or repairing a broken network by removing the broken edge and establishing a new link.

1.1 Problem Definition

More formally, we consider the following problem. We are given a set of vertices V as well as some distance function d on V such that d(v, v) = 0 and d(u, v) = $d(v, u) \ge 0$ for all vertices $u, v \in V$. Further, we are given a partition of $V = V_1 \cup V_2$ and two disjoint trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ on V_1 and V_2 , respectively. We write $n = |V|, m = |E|, n_i = |V_i|$ and $m_i = |E_i|$ for $i \in \{1, 2\}$. For each tree T on a subset $V' \subseteq V$, we consider the tree metric d_T , which is defined on V' such that the distance between $u, v \in V'$ is equal to the sum of the distances on the uniquely defined path between u and v. Further, we assume each vertex $v \in V$ has some non-negative demand c(v). For $V' \subseteq V$ we write $c(V') := \sum_{v \in V'} c(v)$ as a shorthand. We define the weighted routing cost of T as

$$\operatorname{rc}(T) = \sum_{(u,v) \in V \times V} c(u) \cdot c(v) \cdot d_T(u,v) \; .$$

The demands can be considered to be an indicator for the importance of the vertices in the network. The amount of traffic between two vertices in the network is scaled by the product of the demands modeling the fact that important vertices are usually involved in more traffic than less important vertices and that the traffic

^{*}Supported by NSC-DFG Projects NSC98-2221-E-001-007-MY3 and WA 654/18.

 $^{^{\}dagger}\mathrm{Also}$ supported by the Institute of Information Science, Academia Sinica, Taiwan.

 $^{{}^{\}ddagger}Supported by the Concept for the Future of KIT under project YIG 10-209 within the framework of the German Excellence Initiative$

[¶]Dep. of Computer Science and Information Engineering, National Taiwan University, Taiwan, d97021@csie.ntu.edu.tw, dtlee@csie.ntu.edu.tw



Figure 1: Two vertex-weighted trees and best connection (dashed).

between two important vertices is usually larger than that between an important and a less important vertex.

The Optimal Routing Cost Augmentation Problem is to find vertices $u \in V_1$ and $v \in V_2$ such that the routing cost of the tree $T_{uv} = (V, E_1 \cup E_2 \cup \{uv\})$ is minimized; see Figure 1 for an example.

For the Optimal Routing Cost Replacement Problem we are additionally given a pair of vertices $u \in V_1$ and $v \in V_2$ that should be excluded from the solution (since the corresponding edge must be replaced). We can solve this problem by simultaneously computing the best and second-best solution. If the best solution coincides with uv, then we return the second-best solution.

1.2 Contribution

In Section 2 we consider general distance functions. We show that both the optimal routing cost augmentation problem and the optimal routing cost replacement problem can be solved in $\Theta(n_1 \cdot n_2)$ time, which is optimal. In Section 3 we assume that vertices are points in the plane and that the distance between points is equal to the Euclidean distance. We show that both the augmentation problem and the replacement problem can be solved more efficiently in $\mathcal{O}(n \log \min\{n_1, n_2\})$ time by querying the additively weighted Voronoi diagram of a suitably chosen set of points. We adapt this idea to general graph metrics by computing the additively weighted Voronoi diagram on graphs in Section 4. This yields an $\mathcal{O}(n \log n)$ -time algorithm for compactly representable metrics, i.e., metrics that are representable as sparse graphs. We conclude with some remarks and open problems in Section 5.

2 An optimal algorithm for the general case

In this section, we consider general distance functions on the vertex set. We show that the problem can be solved in $\Theta(n_1 \cdot n_2)$ time, which is optimal. For ease of notation we write $C_1 = c(V_1)$ and $C_2 = c(V_2)$ for the total demand in T_1 and T_2 , respectively. Given two vertices $u \in V_1$ and $v \in V_2$, the routing cost of the tree T_{uv} resulting from joining T_1 and T_2 by the edge uv is given by

$$rc(T_{uv}) = rc(T_1) + rc(T_2) + C_2 \cdot \sum_{u' \in V_1} c(u') \cdot d_{T_1}(u', u) + C_1 \cdot \sum_{v' \in V_2} c(v') \cdot d_{T_2}(v, v') + C_1 \cdot C_2 \cdot d(u, v) .$$
(1)

It is composed of the routing cost inside the subtrees T_1 and T_2 of T_{uv} , respectively, and the routing cost effected by the shortest paths using the edge uv between the two trees. Since the total sum of demands for these paths equals $C_1 \cdot C_2$, the edge uv contributes a total amount of $C_1 \cdot C_2 \cdot d(u, v)$ to the routing cost. Furthermore, each shortest path starting at u' in T_1 and ending at u can be extended to a shortest path ending at some vertex v' in T_2 . Hence, each shortest path of this kind contributes its length, weighted by its demand c(u') and the total sum of the demands C_2 in T_2 , to the routing cost. The situation is symmetrical for the paths starting in T_2 and ending at v.

Since the routing costs of T_1 and T_2 do not depend on the choice of the link between the two trees, our problem is equivalent to minimizing the remaining summands in equation (1).

We define the weight of a vertex $u \in V_1$, denoted by w(u), as the sum of lengths of all shortest paths starting at $u' \in V_1$ and ending at u, weighted by the demand of u', i.e.,

$$w(u) = \sum_{u' \in V_1} c(u') \cdot d_{T_1}(u', u) .$$

We define the weight of a vertex $v \in V_2$, denoted by w(v), analogously. Hence, we seek to minimize the term

$$rc'(T_{uv}) = C_2 w(u) + C_1 w(v) + C_1 \cdot C_2 \cdot d(u, v)$$
 (2)

over all possible combinations of $u \in V_1$ and $v \in V_2$.

The weights of the trees can be computed in linear time as follows. First we compute the total demands in T_1 and T_2 , respectively. We compute the weights in T_1 by rooting the tree in some vertex r and performing one bottom-up pass over the tree, followed by a top-down pass. For a vertex u in T_1 we denote the subtree rooted in u by T_u .

In the bottom-up pass, we compute two values for each vertex $u \in V_1$: the total demand $\gamma(u)$ of the vertices in T_u , and the sum $\lambda(u)$ of the shortest paths starting at some vertex u' in T_u and ending at u, weighted by the demand of u', i.e.,

$$\gamma(u) = \sum_{u' \in V(T_u)} c(u')$$

and

$$\lambda(u) = \sum_{u' \in V(T_u)} c(u') d(u', u)$$

For a vertex u with children u_1, \ldots, u_k these values can be computed in linear time as

$$\gamma(u) = c(u) + \sum_{i=1}^{k} \gamma(u_i)$$

and

$$\lambda(u) = \sum_{i=1}^{k} \left(\lambda(u_i) + \gamma(u_i) \cdot d(u_i, u) \right) \,,$$

respectively. In the top-down pass, we compute the weight for each vertex $v \in V_1$. For the root r this weight is equal to $\lambda(r)$. For a vertex v with father $u \in V_1$ the weight can be computed by

$$w(v) = w(u) + (C_1 - 2\gamma(v))d(u, v)$$

This equation is due to the fact that the weight of v is obtained from the weight of u by removing the demand $\gamma(v)$ in the subtree of v from the edge uv and adding the remaining demand $C_1 - \gamma(v)$ to the edge uv. For T_2 we proceed analogously.

Having this, we can compute the best and secondbest connection between the two trees by enumerating all possible pairs uv such that $u \in V_1$ and $v \in V_2$, which yields a total running time of $\mathcal{O}(n_1 \cdot n_2)$. Note, that the described algorithm only finds the best or secondbest solution, but does not compute the routing cost of this solution. If we have no restriction on the distance between the vertices, however, the algorithm is optimal.

Theorem 1 The optimal routing cost augmentation problem and the optimal routing cost replacement problem can be solved in $\mathcal{O}(n_1 \cdot n_2)$ time for general distance function. This is optimal in the algebraic decision tree model.

Proof. We have already outlined the algorithm and argued why it runs within the stated time complexity. It remains to show the lower bound on the running time. For this, we assume that we are given a set of integers a_1, \ldots, a_N . We construct an instance of the optimal routing cost augmentation problem such that finding the minimum routing cost connection between the two trees is equivalent to the minimum of the numbers a_1, \ldots, a_N . For this problem, we need at least N - 1 comparisons in the algebraic decision tree model of computation.

Let $N = n_1 n_2$ be any factorization of N and let V be a set of $n_1 + n_2$ vertices. Further, let $V_1, V_2 \subseteq V$ be a partition of V such that $|V_1| = n_1$ and $|V_2| = n_2$ and let T_1 and T_2 be two arbitrary trees on V_1 and V_2 , respectively. We set the distance between two vertices in the same tree equal to one. Let $x: V_1 \times V_2 \rightarrow \{a_1, \ldots, a_N\}$ be a bijective mapping between the pairs of vertices in V_1 and V_2 and the numbers a_i . Then we choose the remaining distances as follows. Let W_1 and W_2 be the maximum weights of the vertices in T_1 and T_2 , respectively. For $u \in V_1$ and $v \in V_2$ we define

$$d_0(u,v) = C_2 W_1 + C_1 W_2 - C_2 w(u) + C_1 w(v) .$$

Further, we set

$$d(u, v) = \frac{d_0(u, v) + x(u, v)}{C_1 C_2}$$

Then $\operatorname{rc}'(T_{uv}) = C_2W_1 + C_1W_2 + x(u, v)$. For both the augmentation and the replacement problem we need to compute the minimum routing cost solution. However, minimizing the routing cost for the given instance is equivalent to computing the minimum over the values x(u, v) for $u \in V_1$ and $v \in V_2$. Hence, in the algebraic decision tree model of computation, we need at least $n_1 \cdot n_2 - 1$ comparisons, which completes the proof. \Box

3 An Efficient Algorithm for the Euclidean Metric

The proof for the lower bound in the previous section crucially exploits the fact that we can choose distances between the vertices in an arbitrary fashion. If this is not the case, we can come up with more efficient algorithms.

In this section we consider the case that vertices are points in the plane and that the considered metric dis the Euclidean metric. In this case, we can compute the best connection between two trees in $\mathcal{O}((n_1 + n_2) \log \min\{n_1, n_2\})$ time. Throughout the section, we do not distinguish between vertices and points.

Theorem 2 The optimal augmentation problem for the Euclidean metric can be solved in $\mathcal{O}((n_1 + n_2)\log\min\{n_1, n_2\})$ time.

Proof. Without loss of generality we may assume that $n_2 \leq n_1$. Let $\sigma : \mathbb{R}^2 \to \mathbb{R}^2$ be an isotropic scaling with scale factor $s = C_1 \cdot C_2$, i.e., σ scales distances by a factor s and we thus have

$$d(\sigma u, \sigma v) = C_1 \cdot C_2 \cdot d(u, v) .$$
(3)

Let σV_1 and σV_2 denote the scaled sets of points.

For $x \in \mathbb{R}^2$ and $\tilde{v} \in \sigma V_2$ we define a new distance function, defined by $d_+(x, \tilde{v}) := d(x, \tilde{v}) + C_1 \cdot w(v)$, where w is defined as in the previous section. The *additively weighted Voronoi cell of* \tilde{v} is the locus of points

$$\{x \in \mathbb{R}^2 \mid \forall \widetilde{u} \in \sigma V_2 \setminus \{\widetilde{v}\} : d_+(x, \widetilde{v}) < d_+(x, \widetilde{u})\} \quad (4)$$

The additively weighted Voronoi diagram \mathcal{V} defined by d_+ consists of the additively weighted Voronoi cells of the points in σV_2 and can be computed in $\mathcal{O}(n_2 \log n_2)$ time [4].

For each point $u \in V_1$, we locate the nearest neighbor σv of σu in \mathcal{V} using an algorithm with $\mathcal{O}(\log n_2)$ query time described by Kirkpatrick [7]. Then σv satisfies

$$d_{+}(\sigma u, \sigma v) = \min_{v' \in V_2} d_{+}(\sigma u, \sigma v')$$
(5)

and we have

$$d_{+}(\sigma u, \sigma v) = d(\sigma u, \sigma v) + C_{1} \cdot w(v) \tag{6}$$

$$= C_1 \cdot C_2 \cdot d(u, v) + C_1 \cdot w(v) .$$
 (7)

Hence, $v \in V_2$ is the best endpoint of an edge starting at $u \in V_1$ with respect to routing cost. Minimizing $C_2 \cdot w(u) + d_+(\sigma u, \sigma v)$ over all vertices $\sigma u \in V_1$ and their respective nearest neighbor $\sigma v \in V_2$ will thus minimize the overall routing cost. The resulting overall running time is $\mathcal{O}(n_1 \log n_2 + n_2 \log n_2)$.

In order to solve the replacement problem, we also need to compute the second-best solution. We can do this as follows. Let $u^* \in V_1$ and $v^* \in V_2$ be the best solution computed by the algorithm above. This algorithm can trivially be modified to simultaneously compute

$$\min_{u \in V_1 \setminus \{u^*\}, v \in V_2} \operatorname{rc}'(T_{uv})$$

in the same time complexity. By additionally computing the Voronoi diagram only for the points in $V_2 \setminus \{v^*\}$ and repeating the algorithm on this instance, we can also compute

$$\min_{u \in V_1, v \in V_2 \setminus \{v^*\}} \operatorname{rc}'(T_{uv})$$

Clearly, the second-best solution is either of the two. Hence, we have the following corollary.

Corollary 1 The optimal routing cost replacement problem for the Euclidean metric can be solved in time $\mathcal{O}((n_1 + n_2) \log n_2).$

Note that the same approach can also be used in a planar setting, i.e., when the newly introduced edge connecting the two trees may not intersect any other edge of the two trees. In this case we compute an additively weighted constrained Voronoi diagram, which can be done by adapting Fortune's sweepline algorithm [4] with $\mathcal{O}(n \log n)$ running time. In a constrained Voronoi diagram, we are given an additional set of line segments representing obstacles. Whenever the straight line connecting two points intersects one of the obstacles, the distance between the two points is assumed to be infinity, otherwise, it is equal to the (weighted) Euclidean

distance between the points. In our application each edge defined by one of the trees is one such obstacle. Seidel shows how to adapt Fortune's algorithm to compute the constrained Voronoi diagram [9]. The adaption to additively weighted sites has been sketched in Fortune's original paper [4].

Corollary 2 The planar augmentation problem for the Euclidean metric can be solved in $O((n_1 + n_2) \log n_2)$ time.

4 General Metrics

Every finite metric d can be encoded by a finite graph M = (V, D) where each edge $e \in D$ has some length $\ell(e)$ and the distance d between two vertices in V is equal to the sum of the lengths of the shortest path between the vertices in the graph in terms of the edge lengths. We can directly translate our idea from the previous section to this setting by computing the additively weighted Voronoi diagram in M instead. Although the computation of various Voronoi diagrams on graphs has been considered by Hurtado et al. [5], among them a multiplicatively weighted Voronoi diagram, we are not aware of any investigation of the additively weighted Voronoi diagram on graphs. The following theorem is similar to the results by Hurtado et al. [5]. We assume that the additively weighted Voronoi diagram of a a set of sites $S \subseteq V$ on a metric graph G = (V, E) is completely known if every vertex $v \in V \setminus S$ knows its nearest neighbor in S and we know the bisector point for each edge, if it exists.

Theorem 3 The additively weighted Voronoi diagram of a set of sites $S \subseteq V$ on a graph G = (V, E) has complexity $\Theta(m)$ and can be computed in time $\mathcal{O}(m + n \log n)$.

Proof. Each edge of the graph contains at most one bisector point, since moving along the edge will alter the additively weighted distances by the same amount—either increasing or decreasing—for all distances. Hence we have at most m bisector points. On the other hand, we can have exactly m bisectors by setting V' = V. Hence, the complexity of the additively weighted Voronoi diagram is $\Theta(m)$.

To compute the additively weighted Voronoi diagram in G we use the parallel Dijkstra algorithm proposed by Erwig [2] with running time $\mathcal{O}(m+n\log n)$. To compute the diagram, we run Dijkstra's algorithm in parallel using the vertices in S as starting points. For a vertex $v \in V \setminus S$ and some vertex $s \in S$ the distance between vand s is $d_s(v, s) = d_G(v, s) + w(s)$. Whenever a vertex $v \in V \setminus S$ is settled, we update its closest neighbor in S. The bisector points can be computed in $\mathcal{O}(m)$ time from this information. \Box Using this result, we can almost directly translate the technique for the Euclidean case to the general metric case studied in this section.

Theorem 4 The optimal routing cost augmentation problem for general metrics can be solved in time $\mathcal{O}(m+n\log n)$ if the metric is given by a graph M = (V, D)with edge length function ℓ .

Proof. Instead of scaling the point set as in the Euclidean case, we scale the lengths of the edges in G by a factor C_1C_2 , i.e., instead of using ℓ to assess the distance between two vertices in M, we use $C_1C_2\ell$. The rest of the proof is completely analogous. We compute the additively weighted Voronoi diagram on M for the set of sites V_2 . Then we locate the vertex $u \in V_1$ that minimizes $C_2 \cdot w(u) + d_+(u, v)$ where $d_+(u, v)$ is the scaled and additively weighted distance between u and its closest neighbor v. The resulting time complexity is $\mathcal{O}(m+n\log n)$.

Again we can proceed as in the Euclidean case in order to compute the second-best connection between the two trees.

Corollary 3 The optimal routing cost replacement problem for general metrics can be solved in time $\mathcal{O}(m+n\log n)$ if the metric is given by a graph M = (V, D)with edge length function ℓ .

Although this result does not provide an asymptotic improvement in the worst-case, it does show that we can efficiently solve the augmentation problem for compactly representable metrics. If the graph representing the metric is sparse, then the above theorem states that we can solve the augmentation problem in $\mathcal{O}(n \log n)$ as in the Euclidean case.

5 Comments and Open Problems

We have studied a special class of augmentation problems, where the goal is to find the best connection between two disconnected trees in terms of routing cost. Although the problem can not be solved in subquadratic time for general distance functions in the algebraic decision tree model, it can be solved in $\mathcal{O}(n \log n)$ time for the Euclidean metric and sparse graph metrics.

It is an open question, for which graph metrics the problem can be solved in sub-quadratic time. Also, there are some interesting variants of the problem, for instance, when there are more than two disconnected trees. This problem arises, when a vertex of the network fails to work. Additionally, we could consider a Steiner-variant of the problem, in which we are allowed to introduce an additional vertex to which the disconnected components must be connected.

References

- R. Dionne and M. Florian. Exact and approximate algorithms for optimal network design. *Networks*, 9:37–60, 1979.
- [2] Martin Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.
- [3] Matteo Fischetti, Giuseppe Lancia, and Paolo Serafini. Exact algorithms for minimum routing cost trees. *Networks*, 39(3):161–173, 2002.
- [4] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.
- [5] Ferran Hurtado, Rolf Klein, Elmar Langetepe, and Vera Sacristán. The weighted farthest color Voronoi diagram on trees and graphs. *Computational Geometry*, 27(1):13 – 26, 2004.
- [6] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.
- [7] David Kirkpatrick. Optimal search in planar subdivisions. SIAM Journal on Computing, 12(1):28–35, 1983.
- [8] A. J. Scott. The optimal network problem: Some computational procedures. *Transportation Re*search, 3(2):201–210, 1969.
- [9] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria, 1988.
- [10] Richard T. Wong. Worst-case analysis of network design problem heuristics. SIAM Journal on Algebraic and Discrete Methods, 1(1):51–63, 1980.
- [11] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, R. Ravi, and Chuan Yi Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.*, 29:761–778, 1999.

Minimum Many-to-Many Matchings for Computing the Distance Between Two Sequences

Mustafa Mohamad *

David Rappaport[†]

Godfried Toussaint[‡]

Abstract

Motivated by a problem in music theory of measuring the distance between chords and scales we consider algorithms for obtaining a minimum-weight many-to-many matching between two sets of points on the real line. Given sets A and B, we want to find the best rigid translation of B and a many-to-many matching that minimizes the sum of the squares of the distances between matched points. We provide a discrete algorithm that solves this continuous optimization problem, and discuss other related matters.

1 Introduction

Measuring the similarity between two sequences is a problem that arises in many fields including: computational biology [1], computational music theory [11], [12], [13] computer vision [5], and natural language processing [2]. There is a variety of ways to measure the distance between two sequences depending on the specific field of study. Let $A = \{a_1, a_2, \dots, a_m\}$ denote points on a line, such that $a_i < a_{i+1}$ for all $i, 1 \le i \le m - 1$. Similarly we use $B = \{b_1, b_2, \dots, b_n\}$ to denote a sorted set of distinct points on a line. A many-to-many matching pairs one point in A to at least one point in B and vice versa. Given a cost function d(a, b) defined on each matched pair, the cost of the matching is the sum of the costs of all matched pairs. A minimum-weight many-to many matching is one that minimizes cost. We can use the value of the cost of the minimum-weight many-to-many matching to measure the distance between A and B, which we denote by d(A, B).

Our result. We tackle this problem with two different cost measures for d(a, b). The first is the absolute value

of the difference:

$$d_1(a,b) = |a-b|$$
(1)

The second is the square of the difference:

$$d_2(a,b) = d_1^2 \tag{2}$$

We review algorithms for computing these measures to characterize their differences.

A more difficult version of this problem considers similarity measures between A and B allowing rigid translation. That is, we define $B^t = \{b_1 + t, b_2 + t, ..., b_n + t\}$ as a rigid translation of B by the amount t. We present algorithms for computing the minimum-weight many-to-many matching between A and B under such rigid translations. We provide an O(mn) algorithm for computing the minimum $d_1(A, B^t)$ and an $O(3^{mn})$ for computing the minimum $d_2(A, B^t)$. The theoretical upper bound for our algorithm for minimizing $d_2(A, B^t)$ is useful to show that our algorithm is guaranteed to terminate. We also provide experimental results that exhibits polynomial running time of our algorithm on random data.

Preliminaries. In what follows we use N to denote the size of the input. Previous work by Karp and Li [7] and Werman et al. [14] propose an $O(N \log N)$ algorithm for computing the minimum weight one-to-one matching for two equal cardinality point set. The minimum-weight one-to-one matching in this case is the identity matching which is computed by first sorting the points and then mapping a point a_i to a point b_i . Karp and Li [7] also solve the case where $|A| \neq |B|$ in $O(N \log N)$. Colannino et al. [3] extended the work of Karp and Li [7] to compute the minimum-weight many-to-one matching on the real line in $O(N \log N)$. All these results are for the d_1 measure.

The minimum-weight many-to-many matching has also been studied extensively. In a graph theoretic setting, this is equivalent to finding a minimum-weight edge cover of a complete bipartite graph. For an arbitrary bipartite graph, the minimum-weight edge cover can be computed by reducing the problem to the the minimum-weight perfect matching problem [6], [10] which can be computed in $O(N^3)$ time using the Hungarian Algorithm proposed by Kuhn [8]. There is an

^{*}School of Computing, Queen's University, Kingston, ON mustafa@cs.queensu.ca.

[†]School of Computing, Queen's University, Kingston, ON Research supported by NSERC Discovery Grant 388-329. daver@cs.queensu.ca

[‡]Department of Music, Harvard University, Cambridge, MA, Department of Computer Science, Tufts University, Medford, MA, School of Computer Science, McGill University, Montreal, QC. godfried@cs.mcgill.ca

 $O(N^{\omega})$ algorithm for optimal weighted matching in bipartite graphs due to Mucha and Sankowski [9], where ω is the exponent in the best matrix multiplication algorithm (currently $\omega = 2.38$). For the special case where A and B are points on the real line and using the d_1 weight, Colannino et al. [4] provide an $O(N \log N)$ algorithm.

2 Computing a Minimum-Weight Many-to-Many Matching

We review the $O(N \log N)$ algorithm for solving the minimum-weight many-to-many matching problem using the d_1 measure due to Colannino et al. [4], and then show why properties that are used to gain efficiencies do not hold when using the d_2 measure. Without loss of generality, the set A is assumed to have the leftmost element in $A \cup B$. The algorithm partitions the set of sorted points into P_0, P_1, P_2, \ldots subsets such that all points in P_i are less than all points in P_{i+1} , where P_0 is a maximal subset of consecutive points in A, P_1 is a maximal subset of consecutive points in B, and so on (see Figure 1).



Figure 1: Partitioning of the set $A \cup B$

We use the term consecutive partitions to refer to two neighbouring partitions such as P_0 and P_1 . The matching is computed using an optimized dynamic programming approach that uses special properties of the structure of the optimal matching to reduce the complexity of the dynamic program from O(mn) to O(N) for sorted point sets. One of the d_1 properties that allows for an efficient algorithm is the fact that the optimal way to match s consecutive points in two consecutive partitions is to use the identity matching where a_i is paired with b_i for i = 1...s. However, this property does not hold for the d_2 measure (see Figure 2). With the d_2 measure all possibilities of matching two subsets of s points in two consecutive partitions must be checked.

In order to compute the many-to-many matching that minimizes $d_2(A, B)$ we use a *dynamic programming* algorithm. (Note: A similar dynamic programming algorithm has been described by Tymoczko [13]). The algo-



Figure 2: The identity matching on the left is optimal for d_1 with $d_1(A, B) = 6$. However it is not optimal for d_2 where $d_2(A, B) = 12$. The matching on the right is optimal for d_2 with $d_2(A, B) = 10$. As can be seen the edge (2, 4) is only optimal for d_2 if $\epsilon > \sqrt{2}$ or $\epsilon < -\sqrt{2}$

rithm stores the optimal solutions to each subproblem in a table, W, of dimension $m \times n$. The entry w_{ij} in table W stores the optimal matching, $d_2(A_i, B_j)$, of A_i and B_j , where $A_i = \{a_1, a_2, ..., a_i\}$ and $B_j = \{b_1, b_2, ..., b_j\}$. Therefore, the entry w_{mn} will store the weight of the minimum weight many-to-many matching. Refer to Algorithm 1 for the pseudocode. The following lemma proves our claim that w_{mn} stores the minimum weight.

Lemma 1 The optimal value of W_{ij} is given by $W^* + d(a_i, b_j)$ where $W^* = min(W_{i-1,j}, W_{i,j-1}, W_{i-1,j-1})$

Proof. Suppose we have a many-to-many matching M of A_i and B_j such that $\{a_i, b_j\} \notin M$. Therefore a_i is matched with a $b_\ell \in B_j$ such that $b_\ell < bj$ and b_j is matched with an $a_k \in A_i$ with $a_k < a_i$. Observe that the cost of this matching can be lowered by replacing $\{a_i, b_\ell\}$ and $\{a_k, b_j\}$ by $\{a_i, b_j\}$ and $\{a_k, b_\ell\}$, because $(b_j - a_k)^2 + (a_i - b_\ell)^2 - (a_k - b_\ell)^2 - (a_i - b_j)^2 = (a_i - a_k)(2b_j - 2b_\ell)$ is positive. This implies that the edge $\{a_i, b_j\}$ must be part of the minimal many-to-many matching of A_i and B_j . Furthermore, the edge $\{a_i, b_j\}$ is connected to a minimum cost many-to-many matching of A_{i-1} and B_j or A_i and B_{j-1} or A_{i-1} and B_{j-1} . Since the best subproblem is chosen, W_{ij} must be optimal.

Once table W is computed, the actual matching can be extracted from it in O(mn) time. The idea is to traverse table W from the entry $W_{m,n}$ backwards until the entry $W_{1,1}$ is reached. At each step in the traversal, there are three choices to make and the one with the minimum weight is chosen. Clearly the combined complexity of both algorithms is bounded by the size of table W, therefore the total complexity of finding the minimum-weight matching is O(mn).

3 Finding the Minimum-Weight Many-to-Many Matching under Translations

Given sets of points on a line A and B, a *coincident pair* is a point $a \in A$ and a point $b \in B$ such that a = b. When using the d_1 measure we show that there is always Algorithm 1 Dynamic programming algorithm to compute the minimum weight many-to-many matching using the d_2 measure

{Initialization} $W_{1,1} \leftarrow d(a_1, b_1)$ for i = 2 to m do $W_{i,1} \leftarrow W_{i-1,1} + d(a_i, b_1)$ end for for j = 2 to n do $W_{1,j} \leftarrow W_{1,j-1} + d(a_1, b_j)$ end for {Main Loop} for i = 2 to m do for j = 2 to n do $W^* \leftarrow \min(W_{i-1,j}, W_{i,j-1}, W_{i-1,j-1})$ $W_{i,j} \leftarrow W^* + d(a_i, b_j)$ end for end for $\{W_{mn} \text{ stores the weight of the optimal many-to-many}\}$ matching }

an instance of the optimal many-to-many matching under translation with a coincident pair.

return W_{mn}

Lemma 2 Let M be a many-to-many matching of point sets A and B. Then there exists a rigid translation t of the point set B yielding at least one coincident pair such that:

$$\sum_{\{a_i,b_j\}\in M} |a_i - b_j - t| \le \sum_{\{a_i,b_j\}\in M} |a_i - b_j|.$$
 (3)

Proof. If a point from A coincides with a point from B then we are done. For $a \in A$, $b \in B$, and $\{a, b\} \in M$ an edge is a *left edge* if a < b and a *right edge* if a > b. Since none of the points coincide, we don't have the case where a = b. If the number of left edges is greater than the number of right edges we set t for a rigid translation that moves the points B to the right to encounter the first coincident pair, and if the number of left edges we set t for a rigid translation that moves the point that moves the points B to the right to encounter the first coincident pair, and if the number of left edges we set t for a rigid translation that moves the points B to the left to encounter the first coincident pair. In either case it is easy to verify that we get the desired inequality. \Box

Lemma 2 implies that an optimal many-to-many minimum weight matching allowing translations can be found in O(mn) time by applying the algorithm due to Colannino et al. [4] to each alignment of A and Bthat realizes a coincident pair.

The same argument cannot be extended to the d_2 measure. To see this, suppose $A = (a_1 = 2, a_2 = 4, a_3 = 6)$, $B = (b_1 = 2, b_2 = 4)$. Aligning any element of A with any element of B results in a $d_2(A, B) = 4$.

On the other hand, if we translate B by t = 1, we get $d_2(A, B^1) = 3$. In fact, this is still not optimal. The optimal value is t = 1.33, where $d_2(A, B^{1.33}) = 2.667$. To the best of our knowledge, currently, there does not exist an "easy" way of computing the optimal translation, $t_{optimal}$, that would lead to minimizing $d_2(A, B^t)$.

We have developed an algorithm that uses a finite number of steps to find $t_{optimal}$. Let M be the matching for $d_2(A, B)$. In table W in Algorithm 1, $W_{i,j} = \sum_{\{a_i, b_j\} \in M} (a_i - b_j)^2$. We now add the translation variable, t, to every entry of the table W. The modified entry is $W_{i,j} = \sum_{\{a_i, b_j\} \in M} (a_i - b_j - t)^2$. Therefore the cost of a matching is captured by the function:

$$f(t) = \sum_{\{a_i, b_j\} \in M} (a_i - b_j - t)^2$$
(4)

In order to find the t value that optimizes f(t) we take the first derivative of f(t) and set it to zero to get

$$t = \frac{\sum_{\{a_i, b_j\} \in M} (a_i - b_j)}{|M|}.$$
(5)

Our approach is to iteratively translate the point set B by a positive amount a finite number of times, ensuring that we do not pass over an optimal location for B. Recall that to compute the value of $W_{i,j}$, the cost of edge $\{a_i, b_i\}$ is summed to one of the following three subproblems: $W_{i-1,j}, W_{i,j-1}, W_{i-1,j-1}$. Therefore a change in W happens when one of the non-chosen subproblems becomes a better choice than the currently chosen subproblem. Graphically, each subproblem is represented by a parabola, therefore it is easy to determine where a change might happen by computing the intersection of the parabola representing the current choice with the parabolas representing the two other choices. Using this idea we formulate an algorithm for finding the optimal translation, $t_{optimal}$. We start with the set B all the way to the left of A and compute W. Next, we find all intersections between each chosen subproblem (i.e part of the matching) and the two non-chosen subproblems related to it. Out of all intersections, we pick the one with the smallest positive t value. We translate B by this t value and repeat the process until B is translated all the way to the right of A. We store the minimum cost for the matchings as B is being translated. This process guarantees that at least one of the iterations finds the best many-to-many matching. Once B has been translated all the way to the right of A, we pick the smallest value out of all stored values. (Refer to Algorithm 2).

To bound the number of iterations that the algorithm uses and to show that it terminates we define a table, F, that stores the choice of the subproblem made for each entry W_{ij} in W. Entries in the table store the values 1 or 2 or 3 depending on whether $W_{i-1,j-1}, W_{i-1,j}, W_{i,j-1}$ is the optimal subproblem for W_{ij} . We use F_k to represent **Algorithm 2** Algorithm for computing $d_2(A, B^t)$ {Initialization} $results \leftarrow an empty list$ {Shift B to the left of A (i.e. $b_n = a_1$)} $t \leftarrow 0$ $limit \leftarrow a_m$ {Used to stop the loop when all of B is the right of A (i.e. $b_1 = a_m$) while $b_1 \leq limit$ do Compute optimal matching M for (A, B^t) using Algorithm 1 Compute the minimum cost R for M using equation 5 Add (R, M) to results for i from 2 to m do for j from 2 to n do $intersects[i, j] \leftarrow positive intersections of cur$ rent subproblem with the other two subproblems for $W_{i,j}$ end for end for $nexTrans \leftarrow min(intersects[i, j])$ $B^t \leftarrow B^t + nextTrans$ $t \leftarrow t + nextTrans$ end while index = min(first column of results) $bestMatching \leftarrow results[index, 2]$ **return** bestMatching

the state of table F at the end of iteration k of the algorithm.

Theorem 1 Algorithm 2 terminates after $O(3^{mn})$ iterations.

Proof. There are three possible values for every entry in the $m \times n$ table F, so $O(3^{mn})$ is an upper bound on the total number of distinctly different instances of F. We argue that no two instances of F, F_k , and F_ℓ at iterations k and ℓ respectively, are identical. Assume for the sake of contradiction that such a pair of tables exist. Thus, the algorithm will iterate forever in a cycle between these instances. Recall that at each iteration of the algorithm we translate the points B by some positive t. Therefore, at iteration ℓ the location of the points B, call it B_{ℓ} , are to the right of the points at iteration k, B_k . Since we are in a cyclic pattern we have $F_{\ell+1}$ identical to F_{k+1} , and so on as the cycle repeats. This implies that there are a pair of parabolas, p and q that have infinitely many intersection points, a contradiction. Thus, Algorithm 2 cannot cycle, and must terminate after $O(3^{mn})$ iterations.

Theorem 2 Algorithm 2 computes $d_2(A, B^t)$

Proof. Assume there is a translation, t, of the set B for which a minimal $d_2(A, B^t)$ is realized. Clearly,

 $a_1 - b_n \leq t \leq a_m - b_1$. Therefore, t must correspond to a matching in which B falls within the above range. Algorithm 2 considers all possible matchings between A and B that occur as B is being translated within $(a_1 - b_n, a_m - b_1)$. It does so by recomputing the matching every time one of the subproblems becomes a better choice than any other subproblem anywhere in table W. Therefore, the matching that corresponds to t must be found by Algorithm 2. For every matching considered, the algorithm finds the translation that optimizes it. This translation must be equal to t since t is optimal. \Box

3.1 Experimental Results for Algorithm 2

The upper bound that we provide for Algorithm 2 does not appear to be tight. We have not been able to construct an example that requires a super-polynomial number of steps. We ran various experiments to gain a better understanding of the true running time of the algorithm. We know that for each translation that the algorithm makes it takes O(mn) time to compute the distance using dynamic programming. What concerns us is the number of translations that Algorithm 2 makes before finding the optimal translation. Therefore, the number of translations determines whether the algorithm has a polynomial running time or not. We compare the total number of translations to the the product of the cardinalities mn. We defined a ratio, R, by the following equation:

$$R = \frac{\text{number of translations}}{mn}$$

The first experiment was to randomly generate the sets A and B with specific cardinalities, m and n. For the cardinalities chosen, A and B were randomly generated 5 times. The pair that caused the largest number of translations is reported in Table 1. It can be seen that the number of translations of the algorithm is much less than the theoretical upper bound of $O(3^{mn})$. However, the ratio R seems to be slowly increasing and therefore we cannot experimentally bound the number of iterations by mn. Testing the conjecture that the number of translations is poly-logarithmic, we define a new ratio R_2 as:

$$R_2 = \frac{\text{number of translations}}{mn \times \log(mn)}$$

Referring to Table 1, it can be seen that R_2 increases at first and then continues to decrease. This appears to indicate that the running time of Algorithm 2 cannot be larger than a $constant \times mn \log(mn)$. Our experimental results indicate that the constant should not be greater than 2.

In our second experiment, A and B were two randomly generated sets of 10 points where A and B are

m	n	Number of Translations	R	R_2
5	5	63	2.52	0.78
8	10	247	3.09	0.70
15	11	589	3.57	0.70
16	20	1182	3.69	0.64
21	20	1556	3.70	0.61
25	28	2655	3.79	0.62
30	30	3520	3.91	0.62
30	20	2379	3.96	0.58
15	31	1758	3.78	0.57
40	45	7038	3.91	0.52
50	55	10990	4.00	0.54
60	61	14769	4.01	0.51
25	80	8151	4.08	0.49
90	100	37176	4.13	0.45

Table 1: Experimental Results of running Algorithm 2

the same set. We compressed B by dividing the elements of B by an ever increasing factor and ran the algorithm. Overall, R_2 , first increased as B was further compressed by dividing by a larger number. However, the trend reached a peak, and as B was compressed further R_2 started to continuously decrease. Table 2 summarizes our results.

Table 2: Experimental Results of running Algorithm 2 on Compressed B datasets where m = n = 10

Divisor	Number of Translations	R	R_2
1	180	1.80	0.39
2	110	1.10	0.24
4	207	2.07	0.45
6	225	2.25	0.49
10	282	2.82	0.61
10^{2}	540	5.40	1.17
10^{3}	560	5.60	1.22
10^{4}	426	4.26	0.93
10^{5}	367	2.67	0.80
10^{6}	223	2.23	0.48
10^{7}	224	2.24	0.49
10^{8}	116	1.16	0.25
10^{9}	66	0.66	0.14

Picking the largest number of translations in this dataset, we can see that the value of R_2 seems to be slightly higher than the random data set of Table 1. Namely, 1.22 versus 0.78. We performed the same experiment with different cardinalities. The same pattern was noticed. R_2 increased at first and then continuously decreased. In this case, the peak R_2 is 1.54. For each cardinality, Table 3 shows the results of the compression that caused the largest number of translations and therefore the largest R_2 . The R_2 values are generally higher than all previous results, however, we note that

m = n	$m \times n$	Number of Translations	R	R_2
15	225	1817	8.07	1.49
20	400	2680	6.70	1.18
25	625	4266	6.83	1.06
30	900	9402	10.45	1.54
35	1225	10822	8.83	1.24
40	1600	13741	8.59	1.16
45	2025	14761	7.29	0.96

Table 3: Experimental Results of running Algorithm 2 on Compressed B datasets of different cardinalities

the largest R_2 value of 1.54 is still less than 2. Similar results were obtained for $A \neq B$.

Another possibility for a data set that might not have been well captured by generating points randomly, is to have configurations that contain clusters of contiguous points. We generated different cluster configurations for m = 15 and n = 11. In Table 4, a configuration of (3,2) indicates the set A is composed of 3 clusters of points separated by some distance, and likewise the set B is composed of 2 clusters. Each cluster is composed of a fixed number of points that fall within a specific range. The cluster points are generated randomly within the specified range for each cluster. For each cluster configuration, the experiment was run 10 times and the trial with the largest number of translations is shown in Table 4. It can be seen in Table 4 that the largest value for R_2 is 0.80 which is still less than 2.

Table 4: Experimental Results of running Algorithm 2 on different clustered configuration where |A| = 15 and |B| = 11

Configuration	Number of Translations	R	R_2
(3,3)	615	3.73	0.73
(3,2)	676	4.10	0.80
(4,4)	616	3.73	0.73
(4,2)	644	3.90	0.77
(4,1)	678	4.11	0.80

Figure 3 shows that the function $f(mn) = 2mn \log(mn)$ is an upper bound for all our experimental results. Given these results, we conclude that for the data that we have generated Algorithm 2 exhibits a polynomial running time.

4 Conclusion

We have presented an iterative algorithm to solve a continuous optimization problem, that appears to be efficient when applied to randomly generated instances. To date, we have not been able to determine a means to analyze the method to obtain reasonable bounds on the worst case running time of the algorithm. Therefore, we



Figure 3: Summary of experimental results shows that all our data fall under the curve $2mn \log(mn)$

leave open the issue of obtaining a better bound on the number of iterations used by this algorithm. It may be that some modifications could be made to the existing algorithm so that it would be easier to analyze. It may also be the case that an entirely different approach could be used to solve the problem in polynomial time. Alternately, perhaps it can be shown that this optimization problem is NP-hard.

5 Acknowledgements

This work was initiated at the 2nd Bellairs Winter Workshop on Mathematics and Music, co-organized by Dmitri Tymoczko and Godfried Toussaint, held on February 6-12, 2010. We thank the other participants of that workshop, Fernando Benadon, Adrian Childs, Richard Cohn, Rachel Hall , John Halle, Jay Rahn, Bill Sethares, and Steve Taylor, for providing a stimulating research environment.

References

- A. Ben-Dor, R. M. Karp, B. Schwikowski, and R. Shamir. The restriction scaffold problem. *Journal* of Computational Biology, 10(2):385–398, 2003.
- [2] S. R. Buss and P. N. Yianilos. A bipartite matching approach to approximate string comparison and search. Technical report, NEC Research Institute, 1995.
- [3] J. Colannino, M. Damian, F. Hurtado, J. Iacono, H. Meijer, S. Ramaswami, and G. Toussaint. An

 $O(n \log n)$ time algorithm for the restriction scaffold assignment problem. Journal of Computational Biology, 13(4):979-989, 2006.

- [4] J. Colannino, M. Damian, F. Hurtado, S. Langerman, H. Meijer, S. Ramaswami, D. Souvaine, and G. Toussaint. Efficient many-to-many point matching in one dimension. *Graphs and Combinatorics*, 23:169–178, 2007.
- [5] M. F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, and S. Dickinson. Object recognition as many-tomany feature matching. *International Journal of Computer Vision*, 69(2):203–222, 2006.
- [6] T. Eiter and H. Mannila. Distance measures for point sets and their computation. Acta Informatica, 34(2):109–133, February 1997.
- [7] R. M. Karp and S.-Y. R. Li. Two special cases of the assignment problem. *Discrete Mathematics*, 13:129–142, 1975.
- [8] H. W. Kuhn. The Hungarian method for the assignment problem. Naval Research Logistics, 2:83–97, 1955.
- [9] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on, pages 248 – 255, oct. 2004.
- [10] A. Schrijver. Combinatorial Optimization:Polyhedra and Efficiency. Springer-Verlag Berlin Heidelberg, 2003.
- [11] G. Toussaint. A comparison of rhythmic similarity measures. In Proceedings of the 5th International Conference on Music Information Retrieval, pages 242–245, 2004.
- [12] G. Toussaint. The geometry of musical rhythm. In Selected Papers of the Japanese Conference on Discrete and Computational Geometry, J. Akiyama et al., editors, volume 3742 of LNCS, pages 198–212. Springer-Verlag, Berlin, Heidelberg, 2005.
- [13] D. Tymoczko. The geometry of musical chords. Science, 313(5783):72 - 74, July 2006.
- [14] M. Werman, S. Peleg, R. Melter, and T. Y. Kong. Bipartite graph matching for points on a line or a circle. *Journal of Algorithms*, 7:277–284, 1986.

Staying Close to a Curve^{*}

Anil Maheshwari

Jörg-Rüdiger Sack

Kaveh Shahbaz

Hamid Zarrabi-Zadeh

Abstract

Given a point set S and a polygonal curve P in \mathbb{R}^d , we study the problem of finding a polygonal curve through S, which has minimum Fréchet distance to P. We present an efficient algorithm to solve the decision version of this problem in $O(nk^2)$ time, where n and krepresent the sizes of P and S, respectively. A curve minimizing the Fréchet distance can be computed in $O(nk^2 \log(nk))$ time. As a by-product, we improve the map matching algorithm of Alt *et al.* by an $O(\log k)$ factor for the case when the map is a complete graph.

1 Introduction

Matching two geometric patterns is a fundamental problem in pattern recognition, protein structure prediction, computer vision, geographic information systems, etc. Usually these patterns consist of line segments and polygonal curves.

One of the most popular ways to measure the similarity of two curves is to use the Fréchet distance. An intuitive way to illustrate the Fréchet distance is as follows. Imagine a person walking his/her dog, where the person and the dog, each travels a pre-specified curve, from beginning to the end, without ever letting go of the leash or backtracking. The Fréchet distance between the two curves is the minimum length of a leash which is necessary. The leash length determines how similar the two curves are to each other: a short leash means the curves are similar, and a long leash means that the curves are different from each other.

Two problem instances naturally arise: decision and optimization. In the *decision problem*, one wants to decide whether two polygonal curves P and Q are within ε Fréchet distance to each other. In the *optimization problem*, one wishes to determine the minimum such ε . Alt and Godau [2] presented an $O(n^2)$ -time algorithm for the decision problem, where n denotes the total number of segments in the curves. They also solved the corresponding optimization problem in $O(n^2 \log n)$ time.

In this paper, we address the following variant of the



Figure 1: A problem instance.

Fréchet distance problem. Given a point set S and a polygonal curve P in \mathbb{R}^d $(d \ge 2)$, find a polygonal curve Q, with its vertices chosen from S, such that the Fréchet distance between P and Q is minimum. Note that each point of S can be used more than once in Q. In the decision version of the problem, we want to decide if there is polygonal curve Q through S whose Fréchet distance to P is at most ε , for a given $\varepsilon \ge 0$. An instance of the decision problem is illustrated in Figure 1.

One can use the map matching algorithm of Alt *et al.* [1] to solve the decision version of this problem by constructing a complete graph G on top of S, and then running Alt *et al.*'s algorithm on G and P. If n and k represent the sizes of P and S, respectively, this leads to a running time of $O(nk^2 \log k)$ for solving the decision problem.

In this paper, we present a simple algorithm to solve the decision version of the above problem in $O(nk^2)$ time. This improves upon the algorithm of Alt *et al.* [1] by a $O(\log k)$ factor for the case when a curve is matched in a complete graph. Our approach is different from and simpler than the approach taken by Alt *et al.* which is a mixture of line sweep, dynamic programming, and Dijkstra's algorithm.

2 Preliminaries

Let $\varepsilon \ge 0$ be a real number, and $d \ge 2$ be a fixed integer. For any point $p \in \mathbb{R}^d$, we define $\mathcal{B}(p,\varepsilon) \equiv \{q \in \mathbb{R}^d : \|pq\| \le \varepsilon\}$ to be a *ball* of radius ε centered at p, where $\|\cdot\|$ denotes the Euclidean distance. Given a line segment $L \subset \mathbb{R}^d$, we define $\mathcal{C}(L,\varepsilon) \equiv \bigcup_{p \in L} \mathcal{B}(p,\varepsilon)$ to be a *cylinder* of radius ε around L (see Figure 2).

A curve in \mathbb{R}^d can be represented as a continuous function $P: [0,1] \to \mathbb{R}^d$. Given two points $u, v \in P$, we write $u \prec v$, if u is located before v on P. The relation \preceq

^{*}Research supported by NSERC, HPCVL, and SUN Microsystems. Authors' affiliation: School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada. Email: {anil,sack,kshahbaz,zarrabi}@scs.carleton.ca. Fourth author's affiliation: Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, zarrabi@ce.sharif.edu.



Figure 2: A cylinder of radius ε around segment L.

is defined analogously. For a subcurve $R \subseteq P$, we denote by left(R) and right(R) the first and the last point of R along P, respectively.

Given two curves $\alpha, \beta : [0,1] \to \mathbb{R}^d$, the *Fréchet* distance between α and β is defined as $\delta_F(\alpha, \beta) = \inf_{\sigma,\tau} \max_{t \in [0,1]} \|\alpha(\sigma(t)), \beta(\tau(t))\|$, where $\sigma, \tau : [0,1] \to [0,1]$ range over all continuous non-decreasing surjective functions. The following two observations are immediate.

Observation 1 Given four points $a, b, c, d \in \mathbb{R}^d$, if $||ab|| \leq \varepsilon$ and $||cd|| \leq \varepsilon$, then $\delta_F(\overrightarrow{ac}, \overrightarrow{bd}) \leq \varepsilon$.

Observation 2 Let α_1 , α_2 , β_1 , and β_2 be four curves such that $\delta_F(\alpha_1, \beta_1) \leq \varepsilon$ and $\delta_F(\alpha_2, \beta_2) \leq \varepsilon$. If the ending point of α_1 (resp., β_1), is the same as the starting point of α_2 (resp., β_2), then $\delta_F(\alpha_1 + \alpha_2, \beta_1 + \beta_2) \leq \varepsilon$, where + denotes the concatenation of two curves.

3 The Decision Algorithm

Let P be a polygonal curve composed of n line segments P_1, \ldots, P_n , and let S be a set of k points in \mathbb{R}^d . In this section, we provide an algorithm to decide whether there exists a polygonal curve Q whose vertices are chosen from S, such that $\delta_F(P, Q) \leq \varepsilon$, for a given $\varepsilon \geq 0$.

We denote by s and t the starting and the ending point of P, respectively. For each segment P_i of P, we denote by C_i the cylinder $\mathcal{C}(P_i, \varepsilon)$, and by S_i the set $S \cap C_i$. Furthermore, for each point $u \in C_i$, we denote by $P_i[u]$ the line segment $P_i \cap \mathcal{B}(u, \varepsilon)$.

We call a polygonal curve Q feasible if all its vertices are from S, and $\delta_F(Q, P') \leq \varepsilon$ for a subcurve $P' \subseteq P$ starting at s. If Q ends at a point $v \in S$ and P' ends at a point $p \in P$, we call the pair (v, p) a feasible pair. A point $v \in S_i$ is called reachable (at cylinder C_i) if there is a feasible curve ending at v in C_i .

Consider a feasible curve Q starting at a point $u \in S_1$ and ending at a point $v \in S_i$. Since no backtracking is allowed in the definition of Fréchet distance, Q traverses all cylinders C_1 to C_i in order, until it reaches v. Moreover, by our definition of reachability, each vertex of Q is reachable at some cylinder C_j , $1 \leq j \leq i$.

Our approach for solving the decision problem is to process the cylinders one by one from C_1 to C_n , and identify at each cylinder C_i all points of S which are reachable at C_i . The decision problem will be then reduced (by Observation 2) to checking whether there is a reachable point in the ball $\mathcal{B}(t, \varepsilon)$.

To propagate the reachability information through the cylinders, we need a primitive operation described below. Let $u \in S_i$ be a point reachable at cylinder C_i , and let Q be a feasible curve ending at u. For each point $v \in S$, we denote by $r_i(u, v)$ the index of the furthest cylinder we can reach by the curve Q + uv. In other words, $r_i(u, v)$ is the largest index $\ell \ge i$ such that $v \in S_\ell$ is reachable via $u \in S_i$. If Q + uv is not feasible, we set $r_i(u, v) = 0$. The following lemma is a direct corollary of a similar one proved in [1] (Lemma 3) for computing the so-called right pointers.

Lemma 1 ([1]) Given two points $u, v \in S$, we can compute $r_i(u, v)$ for all $1 \leq i \leq n$ in O(n) total time.

We use the following lemma in our algorithm.

Lemma 2 Let $r_i(u, v) = \ell$. For all $i \leq j \leq \ell$, if $v \in S_j$, then v is reachable at C_j .

Proof. Let Q be a feasible curve starting at a point $w \in S \cap \mathcal{B}(s,\varepsilon)$ and ending at u, and let $Q' = Q + \overline{uv}$. Since v is reachable at C_{ℓ} via Q', there is a subcurve P' of P starting at s and ending at a point $p \in P_{\ell}[v]$ (see Figure 3). Consider two point objects \mathcal{O}_P and \mathcal{O}_Q traversing P' and Q', respectively, from beginning to the end, while keeping ε distance to each other. Since v is reachable via $u \in S_i$, \mathcal{O}_P is at a point $a \in P_i$ when \mathcal{O}_Q is at u. Fix a cylinder C_j , $i < j \leq \ell$, such that $v \in C_j$. When \mathcal{O}_P reaches the point $b = \operatorname{left}(P_j[v])$, \mathcal{O}_Q is at a point $x \in \overline{uv}$ such that $||bx|| \leq \varepsilon$. The subcurve of Q' from w to x has Fréchet distance at most ε to the subcurve of P from s to b, and the segment \overline{xv} has Fréchet distance at most ε to the point b by Observation 1. Therefore, by Observation 2, the whole curve Q' has Fréchet distance at most ε to the subcurve P' from s to b, meaning that v is reachable at C_i .

The above proof, not only shows that v is reachable at C_j , but it also shows that the pair $(v, \text{left}(P_j[v]))$ is feasible. The following lemma is therefore immediate.

Lemma 3 If $r_i(u, v) = \ell$ and $v \in S_j$, $i < j \leq \ell$, then $(v, \operatorname{left}(P_j[v]))$ is a feasible pair.



Figure 3: Proof of Lemma 2

The Algorithm Our algorithm for solving the decision problem is provided in Algorithm 1. It maintains, for each cylinder C_i , a set \mathcal{R}_i of all points in S_i which are reachable at C_i . To handle the base case more easily, we assume, w.l.o.g., that the curve P starts with a segment P_0 consisting of a single point $\{s\}$. Every point of S inside the cylinder $C_0 = \mathcal{B}(s, \varepsilon)$ is reachable by definition. Therefore, we initially set $\mathcal{R}_0 = S \cap \mathcal{B}(s, \varepsilon)$ (in line 4).

For each point $v \in S$, the algorithm maintains an index ℓ_v , whose value at the beginning of each iteration i is the following: $\ell_v = \max_{0 \leq j < i, u \in \mathcal{R}_j} r_j(u, v)$. In other words, ℓ_v points to the largest index ℓ for which v is reachable at C_ℓ via a reachable point u in some earlier cylinder C_j , j < i. Initially, we set $\ell_v = 1$ for all points in \mathcal{R}_0 , because all points in \mathcal{R}_0 are also reachable in C_1 , as $C_0 \subseteq C_1$. For all other points, ℓ_v is set to 0 in the initialization step. The following invariant holds during the execution of the algorithm.

Lemma 4 After the *i*-th iteration of Algorithm 1, the set \Re_i consists of all points in S_i which are reachable at cylinder C_i .

Proof. We prove the lemma by induction on i. The base case i = 0 trivially holds. Suppose by induction that, for each $0 \leq j < i$, the set \mathcal{R}_i is computed correctly. In the *i*-th iteration, we first add to R_i (in line 7) all points in S_i which are reachable through a point in a set \mathcal{R}_j , for $1 \leq j < i$. We call these points entry points of cylinder C_i . We then add to \mathcal{R}_i in lines 8–11 all points in S_i which are reachable through the entry points of C_i (see Figure 4 for an example).

We first show that all points added to R_i are reachable at C_i . For each point $v \in S_i$ added to R_i in line 7, we have $\ell_v \ge i$. It means that there is a point $u \in R_j$, for some j < i, such that $r_j(u, v) \ge i$. Therefore, Lemma 2

Algorithm 1 DECISION (S, P, ε)
1: Initialize:
2: compute $r_i(u, v)$ for all $u, v \in S$ and $1 \leq i \leq n$
3: set $\ell_v = 0$ for all $v \in S$
4: let $\mathcal{R}_0 = S \cap \mathcal{B}(s, \varepsilon)$
5: set $\ell_v = 1$ for all $v \in \mathcal{R}_0$
6: for $i = 1$ to n do
7: let $\mathcal{R}_i = \{ v \in S_i : \ell_v \ge i \}$
8: let $q = \min_{v \in \mathcal{R}_i} \operatorname{left}(P_i[v])$
9: for all $v \in S_i \setminus \mathcal{R}_i$ do
10: if $q \leq \operatorname{right}(P_i[v])$ then
11: add v to \mathcal{R}_i
12: for all $(u, v) \in \mathcal{R}_i \times S$ do
13: $\ell_v \leftarrow \max\left\{\ell_v, r_i(u, v)\right\}$
14: return YES if $\mathcal{R}_n \cap \mathcal{B}(t, \varepsilon) \neq \emptyset$



Figure 4: Point v is an entry point of C_i .

implies that v is reachable at C_i . Now, consider a point v added to \mathcal{R}_i in line 11. According to the condition in line 10, there is an entry point w in C_i such that left $(P_i[w]) \preceq \operatorname{right}(P_i[v])$. By Observation 1, the segment \overline{wv} is within ε Fréchet distance to the line segment from left $(P_i[w])$ to right $(P_i[v])$. Moreover, by Lemma 3, $(w, \operatorname{left}(P_i[w])$ is a feasible pair. Therefore, by Observation 2, v is reachable.

Next, we show that any reachable point at C_i is added to \mathcal{R}_i by the algorithm. Suppose that there is a point $v \in S_i$ which is reachable at C_i , but is not added to \mathcal{R}_i . Let Q be a feasible curve ending at v, and w be the first point on Q which is reachable at C_i . By our definition, w is an entry point of C_i . If w = v, then v must be added to R_i in line 7, which is a contradiction. If w is before v on Q, then we have left $(P_i[w]) \preceq \operatorname{right}(P_i[v])$. Now, by our selection of q in line 8, we have $q \preceq \operatorname{left}(P_i[w]) \preceq$ right $(P_i[v])$, and hence, v is added to R_i in line 11, which is again a contradiction. \Box

Theorem 5 Given a polygonal curve P of n segments and a set S of k points in \mathbb{R}^d , we can decide in $O(nk^2)$ time whether there is a polygonal curve Q through Ssuch that $\delta_F(P,Q) \leq \varepsilon$, for a given $\varepsilon \geq 0$. A polygonal curve Q through S of size $O(\min\{n,k\})$ minimizing $\delta_F(P,Q)$ can be computed in $O(nk^2 \log(nk))$ time.

Proof. The correctness of the decision algorithm (Algorithm 1) directly follows from Lemma 4. Line 2 of the algorithm takes $O(nk^2)$ time by Lemma 1. The other three lines in the initialization step (lines 3–5) take only O(k) time. In the main loop, lines 7–11 take O(k) time, and lines 12–13 require $O(k^2)$ time. Therefore, the whole loop takes $O(nk^2)$ time in total.

Once the algorithm finds a reachable point $v \in S_n \cap \mathcal{B}(t,\varepsilon)$, we can construct a feasible curve Q ending at v by keeping, for each reachable point u at a cylinder C_i , a back pointer to a reachable point w at C_j , $j \leq i$, from which u is reachable. The feasible curve Q can be then constructed by following the back pointers from v to a point in $S_1 \cap \mathcal{B}(s,\varepsilon)$. Since at most two points from each cylinder are selected in this process, the curve Q has $O(\min\{n,k\})$ segments. For the optimization problem, we use parametric search as in [1, 2], to find a curve minimizing $\delta_F(P,Q)$ by an extra $\log(nk)$ -factor in $O(nk^2 \log(nk))$ time.

4 Conclusions

In this paper, we presented a simple efficient algorithm for finding a polygonal curve through a given point set S in \mathbb{R}^d such that its Fréchet distance to a given polygonal curve P is minimized. Several interesting problems remain open. For a fixed ε , one can easily modify the algorithm provided in this paper to find a curve with a minimum number of segments, having Fréchet distance at most ε to P. It can be done by keeping reachable points in a priority queue, and propagating the reachability information in a Dijkstra-like manner. However, we cannot see any easy adaptation of our algorithm to find a curve passing through a maximum number of points for a fixed ε . Another major open problem is whether an efficient algorithm exists for computing a curve passing through "all" points of S with a minimum Fréchet distance to P.

The algorithm presented in this paper improves the map matching algorithm of Alt *et al.* [1] for the case of matching a curve in a complete graph. The current lower bound available for the problem is $\Omega((n + k) \log(n + k))$ due to Buchin *et al.* [3]. It is therefore open whether a better algorithm is available, or whether the algorithm obtained in this paper is optimal.

References

- H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. J. Algorithms, 49(2):262–283, 2003.
- [2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. Int. J. of Comput. Geom. Appl., 5:75–91, 1995.
- [3] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? In Proc. 23rd European Workshop Comput. Geom., pages 170– 173, 2007.

Isotopic Fréchet Distance

Tao Ju[†]

Erin W. Chambers^{*}

David Letscher[‡]

Lu Liu[§]

Abstract

We present a variant of the Fréchet distance (as well as geodesic and homotopic Fréchet distance) which forces the motion between the input objects to follow an ambient isotopy. This provides a measure of how much you need to continuously deform one shape into another while maintaining topologically equivalently shapes throughout the deformation.

1 Introduction

We are interested in defining a distance measure between two (homeomorphic) shapes. This measure has a number of potential applications in computer graphics and vision, such as assessing the error when approximating a continuous function by a discrete one, or evaluating the similarity between two shapes. We propose a new distance measure which intuitively is the least effort of morphing a source shape into a target shape, such that each intermediate shape during the morph is homeomorphic to the source. For a given morph, this "effort" is measured as the maximum distance traveled by any point on the source shape.

Our measure is closely related to Fréchet distance, which can be defined as the least travel distance among all possible deformations between the two shapes. Homotopic Fréchet distance [4] further restricts the deformations to be continuous, particularly in the presence of obstacles. However, the intermediate shapes during the deformation may not be homeomorphic to the source shape. For example, they may have self-intersections even though the source shape is intersection-free. Our measure, called *isotopic Fréchet Distance*, enforces the deformation of the source shape to induce a continuous deformation of the ambient space.

Two other distance measures similar to ours, in the special case of curves, were geodesic width [6] and minimum deformation area [10]. Geodesic width considers a class of deformations between two planar curves that is more restricted than what we consider in this work, in that no two intermediate curves during the deformation can intersect. Note that this restriction means that geodesic width is applicable only to non-intersecting curves. The deformation area is defined between two curves lying on any 2-manifold, and considers a similar class of deformations as in our work. The key difference is that the deformation area evaluates the "effort" of morphing as the area swept by the deformation, while the isotropic Fréchet Distance considers the longest distance traveled. Practical work on this problem has also been done, although with no real guarantee of optimality [9].

In this paper, we formulate isotropic Fréchet Distance and compare it with homotopic Fréchet distance. In particular, we give an example in 2D where this new measure better characterizes the dissimilarity between two curves. We also briefly touch on the challenges in computing the measure and its potential applications.

2 Definitions

Consider two homeomorphic subsets A and B of a metric space M. Often M will be Euclidean space or Euclidean space with obstacles removed from it. There are a variety of ways to measure how "close" A and B are. These include Hausdorff distance, Fréchet distance, geodesic Fréchet distance and homotopic Fréchet distance. Hausdorff distance measures how large of a neighborhood of A is needed to contain B and viceversa.

Definition 1 Given $A, B \subset M$, the Hausdorff distance between them is

$$\mathcal{H}(A,B) = \max\{\sup_{a \in A} \inf_{b \in B} d(a,b), \sup_{b \in B} \inf_{a \in A} d(a,b)\}$$

However, Hausdorff distance is solely based on geometry and ignores the topology of both A and B. Fréchet distance considers all possible homeomorphic pairings between points in A and B and how far away paired points are. The distance is defined to be the minimum over all homeomorphisms between A and B of the maximum distance between any pair identified points. In many applications, Fréchet distance is only defined for curves; this definition generalizes this for arbitrary Aand B, see [3] for a similar definition. Unless otherwise specified, all maps are assumed to be continuous.

^{*}Department of Mathematics and Computer Science, Saint Louis University, echambe5@slu.edu

[†]Department of Computer Science, Washington University in St. Louis, taoju@cse.wustl.edu

[‡]Department of Mathematics and Computer Science, Saint Louis University, letscher@slu.edu

 $^{^{\$}} Department of Computer Science, Washington University in St. Louis, ll10@cse.wustl.edu$

Definition 2 Given $A, B \subset M$ with $X \cong A \cong B$, the Fréchet distance between them is

$$\mathcal{F}(A,B) = \inf_{\substack{f,g:X \to M \\ f(X) = A, g(X) = B}} \sup_{x \in X} d(f(x),g(x))$$

Following geodesic paths between identified pairs of points (and around any obstacles) gives a way to deform one shape to another, yielding the geodesic Fréchet distance [5]. However, under either geodesic or standard Fréchet distance, nearby points do not follow similar paths when obstacles are present in the underlying space. For example, if A and B are curves and the geodesics are thought of as the traditional "dog leash" connecting the curves, the leash might jump discontinuously over any obstacles in the space.

Homotopic Fréchet distance [4] restricts Fréchet distance still further and only considers continuous deformations of one shape to another. For these definitions we must assume that M has a Riemannian metric or some other structure that allows the measurement of the length of curves.

Note that in the following definition, we consider X to be an abstract representation of A and B (which are homeomorphic), allowing us to match different pairs of points in A and B by fitting them to points in the reference shape X. In previous work, this distance was only defined if A and B were curves, and it was assumed that the parametrizations of the curves were non-decreasing. However, if only monotonic parametrizations are considered then the infimum does not change. Notice that monotonic parametrizations are homeomorphisms, so the following definition does generalize the definition of homotopic Fréchet distance to general spaces.

Definition 3 Given $A, B \subset M$ with $X \cong A \cong B$, the homotopic Fréchet distance between them is

$$\overline{\mathcal{F}}(A,B) = \inf_{\substack{h: X \times [0,1] \to M \\ h(X,0) = A, h(X,1) = B}} \max_{x \in X} \ln h(x,\cdot)$$

Continuous deformations of one space to another can change the topology along the way. If we want to ensure that all intermediate spaces are identical and are embedded into Euclidean space then we replace homotopies by ambient isotopies. In essence, isotopic Fréchet distance treats the intermediate curves or shapes themselves as obstacles during deformations.

Definition 4 Given $A, B \subset M$ with $X \cong A \cong B$, the (ambiently) isotopic Fréchet distance between them is

$$\begin{aligned} \mathcal{I}(A,B) = & \inf_{\substack{h: \ M \times I \to M \\ h(\cdot,t) \ homeomorphism \\ h(x,0) = x \ \forall x \in X \\ h(A,1) = B }} & \max_{x \in X} \ln h(x,\cdot) \end{aligned}$$



Figure 1: (a) Two curves with significantly different Hausdorff and Fréchet distances. (b) With an obstacle between them the homotopic Fréchet distance is larger than the Fréchet distance.

Ambient isotopies continuously deform both the shape and the space containing it to another shape. For example, any pair of knots in \mathbb{R}^3 are homotopic, but distinct knots are not ambiently isotopic, since we cannot continuously morph between them without any selfintersection along the way. This means that there are homeomorphic subsets of \mathbb{R}^3 that have infinite isotopic Fréchet distance.

Any ambient isotopy also defines a homotopy between A and B, so isotopic Fréchet distances is at least as large as homotopic Fréchet distance. In fact, we have

$$\mathcal{H}(A,B) \le \mathcal{F}(A,B) \le \overline{\mathcal{F}}(A,B) \le \mathcal{I}(A,B)$$

Figure 1 shows examples where Fréchet distance is strictly larger than Hausdorff distance and homotopic Fréchet distance is strictly larger than geodesic Fréchet distance. In section 5 we will give an example where homotopic and isotopic Fréchet distance differ.

Before examining any examples, we will demonstrate that it is appropriate to consider isotopic Fréchet distance a "distance".

Lemma 1 Given any subset $X \subset M$, isotopic Fréchet distance is a metric on the space of all embeddings $X \rightarrow M$.

Proof. Since isotopic Fréchet distance is defined as an infimum of a set of lengths of curves, it is clearly non-negative. And if the isotopic Fréchet distance is zero, then in the limit, points are moved a distance of 0. Thus $\mathcal{I}(A, B) = 0$ implies that A and B are equal.

If $h: M \times I \to M$ is a isotopy from A to B then define $h': M \times I \to M$ by

$$h'(x,t) = h(g^{-1}(x), 1-t)$$

where g(x) = h(x, 1). Clearly for any $t, h(\cdot, t)$ is a homeomorphism and

$$h'(x,0) = h(g^{-1}(x),1) = g(g^{-1}(x)) = x$$

 $h'(B,1) = h(g^{-1}(B),0) = h(A,0) = A$

This shows that h' is an ambient isotopy from B to A. The lengths of these two isotopies are identical so the infimum over all possible isotopies for A to B and Bto A, respectively must be the same. This shows that isotopic Fréchet distance is symmetric.

Finally, we need to show that Fréchet distance satisfies the triangle inequality. Assume that $X \cong A \cong B \cong C$. It is enough to show that for any $\epsilon > 0$ there exists an isotopy $h: M \times I \to M$ from A to C such that

$$\max_{x \in X} \ln h(x, \cdot) \le \mathcal{I}(A, B) + \mathcal{I}(B, C) + \epsilon$$

Chose any $\epsilon > 0$. By definition of isotopic Fréchet distance there exists an isotopy $h_1 : M \times I \to M$ from Ato B such that

$$\max_{x \in X} \ln h_1(x, \cdot) \le \mathcal{I}(A, B) + \epsilon/2$$

and an isotopy $h_2: M \times I \to M$ from A to B such that

$$\max_{x \in \mathbf{Y}} \ln h_2(x, \cdot) \le \mathcal{I}(B, C) + \epsilon/2$$

Define the isotopy $h: M \times I \to M$ by

$$h(x,t) = \begin{cases} h_1(x,2t) & \text{if } t \le \frac{1}{2} \\ h_2(h_1(x,1),2t-1) & \text{if } t > \frac{1}{2} \end{cases}$$

h is continuous since both functions agree when $t = \frac{1}{2}$. Furthermore, we see that

$$h(x,0) = h_1(x,0) = x$$

$$h(x,1) = h_2(h_1(A,1),1)$$

$$= h_2(B,1) = C$$

Thus for any $\epsilon > 0$ we have

$$\begin{split} \mathcal{I}(A,C) &\leq \max_{x\in X} \ln h(x,\cdot) \\ &\leq \max_{x\in X} \ln h_1(x,\cdot) + \max_{x\in X} \ln h_2(x,\cdot) \\ &\leq \mathcal{I}(A,B) + \epsilon/2 + \mathcal{I}(B,C) + \epsilon/2 \\ &= \mathcal{I}(A,B) + \mathcal{I}(B,C) + \epsilon \end{split}$$

which completes the proof.

3 An Extra Constraint on the Isotopy

All of the distance measures between shapes which we have discussed are determined only by a single maximal distance, and many homotopies realize this distance. We can expand upon these definitions to consider the lengths of all leashes, with our end goal being to somehow minimize the distance any point travels in the homotopy realizing the minimum isotopic Fréchet distance. For any point x, the curves $h(x, \cdot)$ will be referred to as the *trajectories* of the point under the homotopy; this is also sometimes referred to as the set of *leashes*.

For a homotopy $h: X \times I \to M$ (possibly induced by an isotopy), its length function $L: X \to \mathbb{R}^+$ is defined by $L(x) = \operatorname{len} h(x, \cdot)$. Homotopic Fréchet and isotopic Fréchet distances focus on minimizing the maximum of L(x) over the space of homotopies and isotopies, respectively. There are other measures of complexity, however, For example we could minimize the area or L_2 norm of these homotopies, which is equal to $\sqrt{\int_X (L(x))^2 dx}$, similar to what is done in [10], or we could consider some other L_p norm $(\int_X (L(x))^p dx))^{1/p}$. However, homotopies and isotopies minimizing these norms will not realize homotopic and isotopic Fréchet distance, respectively.

If there were only finitely many lengths to consider then we could sort them in decreasing order and then compare them. The lexicographic minimum would not only minimize the maximum of L(x), but also minimize the length of the second longest leash length among homotopies minimizing the maximum. Similarly statements hold for the third longest curve and so on. However, this comparison process would only work for discrete sets. This notion can be generalized to the continuous case by consider the set of trajectories lengths for points that are local maximi of the function L(x). When these sets of lengths are minimized lexicographically not only is the length of the longest curve minimized but the next largest local maximum in lengths is also minimized and so on. This yields a complexity measure on homotopies that not only realizes homotopic or Fréchet distances (depending on the space the infimum is taken over) but also moves other points as little as possible. In fact, algorithms used to compute Fréchet, geodesic Fréchet and homotopic Fréchet distances all produce pairs that minimize these more general complexities.

In 3 dimensions, minimal isotopies can be used to morph between homeomorphic shapes. Isotopies that minimize complexity would, in some sense, be minimal morphs between the two shapes. If an efficient algorithm could be found to minimize this complexity then it would yield morphs with some quality guarantees.

4 An Example

Consider the spiral curve in figure 2 compared to a straight line segment. In the minimal homotopy between them, most of the spiral collapses to a single point. This is not an ambient isotopy because at time 1 in the homotopy there is an instantaneous change in



Figure 2: Comparing a spiral to a straight line. The minimal homotopy of a spiral to a line collapse the spiral to a point and the "obvious" isotopy of the spiral unravels it (note: this is not minimal).

the topology of small neighborhoods of this collapsing point.

This means that this minimal homotopy does not come from an isotopy. A natural possibility for an isotopy between the two curves is also shown in figure 2. This isotopy unravels the spiral until it flattens out completely. It is conceivable that this "obvious" isotopy realizes isotopic Fréchet distance.

In fact, for this spiral curve the isotopic Fréchet distance is equal to the Fréchet distance. To see this notice that the homotopy that realizes Fréchet distance is an isotopy arbitrarily close to time t = 1. So this homotopy can be followed until the spiral is as small as desired and then unwrapped. This will result in an isotopy whose longest trajectory is arbitrarily close to the Fréchet distance. This gives a sequence of isotopies whose longest trajectory length limits to the Fréchet distance proving that the two distance measures are (somewhat surprisingly) the same in this particular instance.



Figure 3: Two curves with Fréchet distance ϵ , but isotopic Fréchet distance at least $\frac{2}{9}L$. (Conjecturally the isotopic Fréchet distance is $\sqrt{L^2 + \epsilon^2}$.)

5 Isotopic Fréchet \neq Homotopic Fréchet

The pair of oppositely oriented "zig-zag" curves in figure 3 give an example where the curves are very close in terms of Fréchet distance but very far apart in isotopic Fréchet distance. The minimal homotopy between these curves is shown in figure 4. The homotopy preserves x coordinates of all of the points. It narrows the zig-zag until it flattens out and then expands it in the opposite direction. This is not an isotopy, and unlike the previous example, it cannot be modified to yield an isotopy. In fact, we will show that we can achieve arbitrarily large isotopic Fréchet distance relative to Fréchet distance by modifying the width and height of this figure.

Proposition 2 For any L > 0 and $\epsilon \in (0, L/2)$, there exists a pair of curves $C_1, C_2 \subset \mathbb{R}^2$ with

$$\mathcal{F}(C_1, C_2) = \mathcal{H}(C_1, C_2) = \epsilon$$
$$\mathcal{I}(C_1, C_2) \geq \frac{2}{9}I$$

Proof. Consider the two curves in Figure 3, where the vertices are the points s = (0,0), $s^+ = (0,\epsilon/2)$, $s^- = (0, -\epsilon/2)$, t = (L,0), $t^+ = (L,\epsilon/2)$ and $t^- = (L, -\epsilon/2)$. The first curve, C_1 , consists of line segments $s \to t^+ \to s^- \to t$ and the second, C_2 , travels from $s \to t^- \to s^+ \to t$. An easy exercise in calculating Fréchet distance shows that $\mathcal{F}(C_1, C_2) = \epsilon$. Furthermore, the maximum distances are realized by identifying t^+ to t^- and s^+ to s^- . (Note that since there are no obstacles, the Fréchet distance between these curves is the same as the homotopic Fréchet distance.)

Assume $h: M \times I \to M$ is a minimal isotopy between C_1 and C_2 and that it moves each point in C_1 along a curve whose length is at most $\frac{2}{9}L$. So we may assume that the points t^+ and s^- are moved a distance at most $\frac{2}{9}L$ by the isotopy. Let p be the point $(\frac{4}{9}L, -\frac{2}{9}\epsilon)$ on the curve C_1 . Assume that after the isotopy p is sent to p' = h(p, 1). If p' is not on the line segment from s^+ to t to the left of the line $x = \frac{2}{3}L$ then some point on C_1 between p and t is moved a distance greater than $\frac{2}{9}L$, a

contradiction. So, we will assume that p' is on the line segment from s^+ to t with x coordinate at most $\frac{2}{3}L$.



Let l be the line segment from s^- to t^+ . The point p is below l and the isotopy takes p to p' which is above l. Both the line and point move during the isotopy, but they cannot cross. The furthest we are allowing s^- to move under the isotopy is $\frac{2}{9}L$, so the x coordinate never exceeds $\frac{2}{9}L$. Similarly, the x coordinate of t^+ never drops under $\frac{7}{9}L$ as the point moves. Hence, during it's path in the isotopy p must have it's x-coordinate either go below $\frac{2}{9}L$ or above $\frac{7}{9}L$. So the length that p moves is at least L. This implies that any isotopy must move some point a distance of at least $\frac{2}{9}L$, providing the lower bound on isotopic Fréchet distance.

In figure 4, a few intermediate curves of an isotopy between the two curves are shown. This isotopy leaves s and t fixed, s^+ is sent to t^- at unit speed and s^- is sent to t^+ at unit speed. The line segments are sent to straight lines connecting these points as they move. The corners are the points that move furthest, and they move a distance of $\sqrt{L^2 + \epsilon^2}$. We conjecture that this is the isotopic Fréchet distance between the two curves. Note that in this isotopy, the trajectories of every point follows a straight line, but this will not be not true in general.

6 Calculating Isotopic Fréchet Distance

For curves in the plane, Fréchet distance can be calculated in quadratic time [1], and when polygonal obstacles are present, homotopic Fréchet distance can also be calculated in polynomial time [4]. These algorithms rely on the fact that trajectories on any point or leash must be a straight line if no obstacles are present and a geodesic in general. However, for isotopies trajectories, we must avoid other intermediate points, and so the trajectories will typically be piecewise linear. Also, isotopies do not need to proceed monotonically; in fact, they may have to back-track multiple times in the course of a minimal isotopy. For piecewise-linear curves or surfaces, isotopic Fréchet distance can be approximated by turning it into a high dimensional motion planning problem. This approach would work for both 2 and 3 dimensional shapes. While it might yield good approximations to isotopic Fréchet distance, we would not expect these algorithms to be particularly fast.

It is also possible that previous approaches to morphing, such as [9], may yield computations that would realize the isotopic Fréchet distance, although the connection is not clear.

7 Applications

As shown above, the isotropic Fréchet distance more faithfully captures the effort of deforming one shape into another when compared to the homotopic Fréchet distance, particularly between undulating shapes. Hence it can serve as a better similarity measure between such shapes, which occur in many relevant settings such as human cortical surfaces which contain numerous folds (sulci and gyri). It could also yield a similarity measure between different structures for the same protein. Moreover, the algorithm for finding the isotropic Fréchet distance would also yield an optimal morphing sequence where each intermediate shape is free of intersections. Such intersection-free morphing is highly desirable for computer graphics applications such as animation, yet computational methods are scarce [7, 8, 9].

8 Future Work

Obviously, the most interesting open problem remaining is to determine an algorithm to compute isotopic Fréchet distance. The main challenge here is that we cannot fix obstacles in this measure, as is done in both geodesic and homotopic Fréchet distance, since the obstacles are the curves themselves as they change over time, so the problem seems harder than computing Fréchet distance between curves.

In more general spaces, not much is known beyond the fact that computing Fréchet distance between surfaces is upper semi-computable [3] and hard for some cases [2], so it is perhaps more reasonable to look for approximation algorithms to compute isotopic Fréchet distance in settings such as this.

References

- H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *IJCGA*, 5(1–2):75–91, 1995.
- [2] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: some simple hard cases. In *Proceedings* of the 18th annual European conference on Algorithms: Part II, ESA'10, pages 63–74, 2010.



Figure 4: (a) A minimal homotopy between the curves C_1 and C_2 . Note that this homotopy is not an isotopy. (b) A conjectured minimal isotopy between the two curves C_1 and C_2 , this isotopy gives an upper bound on isotopic Fréchet distance of $\sqrt{L^2 + \epsilon^2}$.

- [3] M. Buchin. Semi-computability of the Fréchet distance between surfaces. In In Proc. 21st European Workshop on Computational Geometry, pages 45–48, 2005.
- [4] E. W. Chambers, É. C. de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Comput. Geom.*, 43(3), 2010.
- [5] A. F. Cook, Iv, and C. Wenk. Geodesic Fréchet distance inside a simple polygon. In Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS, pages 193–204, 2008.
- [6] A. Efrat, L. J. Guibas, S. Har-Peled, J. S. B. Mitchell, and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002.
- [7] C. Erten, S. G. Kobourov, and A. Pitta. Intersectionfree morphing of planar graphs. In *In Proc. GD 2003*, *LNCS 2912*, pages 320–331. Springer, 2003.
- [8] C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. Computers & Graphics, 25(1):67–75, 2001.
- [9] H. Iben, J. O'Brien, and E. Demaine. Refolding planar polygons. *Discrete and Computational Geometry*, 41:444-460, 2009.
- [10] Y. Wang. Measuring similarity between curves on 2manifolds via minimum deformation area, 2008.

Edge Unfoldings of Platonic Solids Never Overlap

Takashi Horiyama*

```
Wataru Shoji*
```

Abstract

Is every edge unfolding of every Platonic solid overlapfree? The answer is yes. In other words, if we develop a Platonic solid by cutting along its edges, we always obtain a flat nonoverlapping simple polygon.

We also give self-overlapping general unfoldings of Platonic solids other than the tetrahedron (i.e., a cube, an octahedron, a dodecahedron, and an icosahedron), and edge unfoldings of some Archimedean solids: a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron.

1 Introduction

"Does every convex polyhedron have a nonoverlapping edge unfolding?" An unfolding (also called a general unfolding) of a polyhedron is a simple polygon obtained by cutting the surface of the polyhedron and unfolding it into a plane. For an edge unfolding, only cutting along the edges is allowed. The origin of unfoldings of a polyhedron goes back to the 16th century: In 1525, Albrecht Dürer, a painter and a mathematician, published a book entitled "Unterweysung der Messung mit dem Zirkel un Richtscheyt in Linien Ebnen uhnd Gantzen Corporen" [11]. In this book, he gave edge unfoldings of Platonic solids (also called regular convex polyhedra) and Archimedean solids (also called semi-regular convex polyhedra). There is no evidence that Dürer was aware of the question, but he seems to have some insight to the question based on his unfoldings [10]. The first explicit statement was given by Shephard in 1975 [25].

Although it was believed that every edge unfolding of a convex polyhedron never overlaps, some unfortunate cut may lead to overlapping unfoldings [9, 10, 20, 22]. If we relax the restriction of convexity, there exist nonconvex polyhedra whose every edge unfolding is selfoverlapping [4, 12]. If we allow general unfolding, there are two techniques for unfolding any convex polyhedron to a simple polygon [3, 21, 24], i.e., any convex polyhedron has at least one general unfolding. In [23], the probability of overlap is investigated for a random unfolding of a random polyhedron constructed using random points on a sphere.



Figure 1: Overlapping general unfoldings of a cube.



Figure 2: Overlapping general unfoldings of an octahedron, a dodecahedron, and an icosahedron.

In this paper, we consider the problem from another point of view. What happens if our polyhedron is more restricted and has a regular structure? "Are there any overlapping general unfoldings for Platonic solids?" Although this seems to be a naïve question at a first glance, we have the following two interesting observations with slightly relaxed conditions.

First, let us consider the case with general unfolding. As for a tetrahedron, any unfolding is a fundamental domain of tiling [2], i.e., any unfolding can tile the plane. This statement implicitly says that any general unfolding never overlaps. Surprisingly, for other Platonic solids, we found overlapping unfoldings: Figures 1 (a) and (b) are unfoldings of a cube that overlap in a point and a line, respectively. If we cut along the dotted line in Figure 1 (c), and glue the edges labeled a, we obtain an overlapping unfolding in Figure 1 (d). (Gray hatch indicates the overlap.) We can find similar overlapping unfoldings for an octahedron, a dodecahedron, and an icosahedron, respectively, in Figure 2.

Next, let us consider the case with Archimedean solids. It is known that a snub dodecahedron has an overlapping edge unfolding [9]. We also found overlapping edge unfoldings of a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron. By cutting along the bold lines of the polyhedra in Figure 3, we obtain their overlapping edge unfoldings.

As a result of the above observations, we will focus on the case for edge unfoldings: "Is every edge unfolding

^{*}Graduate School of Science and Engineering, Saitama University, horiyama@al.ics.saitama-u.ac.jp, s10mm309@mail .saitama-u.ac.jp.

of every Platonic solid overlap-free?" In other words, "Are there any overlapping edge unfoldings for Platonic solids?" As for a tetrahedron, a cube, and an octahedron, they have 2, 11, and 11 edge unfoldings [15], respectively, and we can check all of them are overlapfree by drawing them one by one. For a long time, it was believed that the same situation holds for a dodecahedron and an icosahedron. We solve this problem and say that it is correct.

Theorem 1 (Main result) If we unfold a Platonic solid by cutting along its edges, we always obtain a flat nonoverlapping simple polygon.

We solve the problem by enumerating all edge unfoldings, and check whether they are overlapping or not. It is known that a dodecahedron and an icosahedron have 43,380 edge unfoldings [6, 13]. Note that they are dual to each other. Our contribution is to strengthen this result by making a catalogue of edge unfoldings for Platonic solids. For each pair of non-neighboring faces in the unfoldings, we check whether their circumscribed circles overlap or not. Since there are no overlap, we confirm the claim that has been believed for a long time.

Our contribution also includes a proposal of enumeration algorithms by binary decision diagrams (BDDs) [1, 7] for solving problems in computational geometry. A BDD is a directed acyclic graph representing a Boolean function, and can be considered as a variant of a decision tree. By restricting the order of variable appearance and by sharing isomorphic subgraphs, BDDs have the following useful properties: (1) When an ordering of variables is specified, a BDD has the unique reduced canonical form for each Boolean function. (2) Many Boolean functions appearing in practice can be compactly represented. (3) When a BDD is given, satisfiability and tautology of the represented function can be easily checked in constant time. (4) There are efficient algorithms for many other Boolean operations on BDDs. As a result of these properties, BDDs (and its variants) are used for various practical applications, especially in computer-aided design and verification of digital systems (see e.g., [8, 17, 26]). Recently, BDDs are widely used in various fields (see e.g., [14, 19]). Knuth devoted notable space in "The Art of Computer Programming" with BDDs [16]. BDDs are regarded as a succinct data structure with efficient manipulation algorithms.

The rest of this paper is organized as follows. The next section gives fundamental concepts on BDDs. We propose algorithms for enumerating edge unfoldings and checking whether they are overlap-free or not in Section 3, and their results are given in Sections 4.

2 Binary Decision Diagrams

A binary decision diagram (BDD) is a directed acyclic graph that represents a Boolean function. It has two



Figure 3: Overlapping unfoldings of a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron.

sink nodes 0 and 1, called the 0-node and the 1-node, respectively (which are together called the constant nodes). Other nodes are called variable nodes, and each variable node v is labeled by one of the variables x_1, x_2, \ldots, x_n . Let var(v) denote the label of node v. Each variable node has exactly two outgoing edges, called 0-edge and 1-edge, respectively. One of the variable nodes becomes the unique source node, which is called the root node. Let $X = \{x_1, x_2, \dots, x_n\}$ denote the set of n variables. A variable ordering is a total ordering $(x_{\pi(n)}, x_{\pi(n-1)}, \ldots, x_{\pi(1)})$, associated with each BDD, where π is a permutation $\{1, 2, \ldots, n\} \rightarrow$ $\{1, 2, \ldots, n\}$. The level of a variable $x_{\pi(i)}$ is defined to be *i*. Similarly, the level of a node v is defined by its label; if node v has label $x_{\pi(i)}$, its level is defined to be i. That is, the root node is in level n and has label $x_{\pi(n)}$, the nodes in level n-1 have label $x_{\pi(n-1)}$ and so on. The level of the constant nodes is defined to be 0. On every path from the root node to a constant node in an BDD, each variable appears at most once in the decreasing order of their levels. The size of a BDD is the number of nodes in it.

Every node v of a BDD represents a Boolean function f_v , defined by the subgraph consisting of those edges and nodes reachable from v. If node v is a constant node, f_v equals to its label. If node v is a variable node, f_v is defined as $var(v)f_{0-succ(v)} \vee var(v)f_{1-succ(v)}$ by Shannon's expansion, where 0-succ(v) and 1-succ(v), respectively, denote the nodes pointed by the 0-edge and the 1-edge from node v. The function f represented by a BDD is the one represented by the root node. When two nodes u and v in a BDD represent the same function, and their levels are the same, they are called equivalent. A node whose 0-edge and 1-edge both point to the same node is called redundant. A BDD which has no mutually equivalent nodes and no redundant nodes is reduced. In the following, we assume that all BDDs are reduced.

An assignment to variables in X can be regarded as a subset $S \subseteq X$, and a Boolean function f can be regarded as a family $\mathcal{F} \subseteq 2^X$. For example, an assignment $(x_3, x_2, x_1) = (1, 1, 0)$ can be regarded as a set $\{x_3, x_2\}$, and $x_1(x_2\overline{x}_3 \lor \overline{x}_2 x_3)$ can be regarded as a family $\{\{x_2, x_1\}, \{x_3, x_1\}\}$. By using BDDs for representing families of a set, we can use Boolean operations on BDDs as a family algebra (see e.g., [16]), or set operations. Later in this paper, we identify a Boolean function with its corresponding family \mathcal{F} , unless confusion arises.

For solving a constraint satisfaction problem, all we have to do is to interpret the restrictions of the problem as a form of Boolean functions, and represent them by BDDs. By applying AND operation to the BDDs, we can obtain the BDD representing the solutions satisfying all of the restrictions. Once such BDD is obtained, paths from the root node to the 1-node correspond to satisfying assignments for its function. Thus, we can enumerate all solutions by traversing the BDD.

3 Algorithms for Enumerating and Checking Edge Unfoldings

By the following three steps, we enumerate edge unfoldings of Platonic solids and check whether they are overlap-free: (1) We represent the constraints for edge unfoldings as BDDs. (2) We eliminate mutually equivalent unfoldings. (3) We check whether they are overlapfree or not. We propose algorithms for these subproblems, which are applicable to any of the Platonic solids. Later in this paper, we denote n and m as the number of vertices and edges of a Platonic solid, respectively.

3.1 Enumeration of Edge Unfoldings

We start with the following lemma that gives a good insight for edge unfoldings.

Lemma 2 (See [10, Lemma 22.1.1]) The cut edges of an edge unfolding of a convex polyhedron form a spanning tree of the 1-skeleton (i.e., the graph formed by the vertices and the edges) of the polyhedron.

This lemma implies two characterizations of edge unfoldings, and we propose two algorithms according to these characterizations. The first characterization is that a set S of cut edges gives an edge unfolding if and only if (1) S consists of exactly n - 1 edges and (2) no edges in S form a cycle. For interpreting these constraints as Boolean functions, we use m Boolean variables x_1, x_2, \ldots, x_m representing whether edges are cut or not: $x_i = 1$ if its corresponding edge e_i is cut, otherwise $x_i = 0$.

Condition (1) is represented as a Boolean function that outputs 1 if and only if exactly n-1 out of m variables are 1's. Such function is one of symmetric functions, whose BDD is of size $O(m^2)$ [16]. Figure 4 shows a BDD of a function that outputs 1 if and only if 3 out of 6 variables are 1's. The left-most column of the variable nodes implies that no 1's have been received yet.



Figure 4: A BDD representing that exactly 3 out of 6 variables are 1's.

Similarly, the four columns in Figure 4 represent that we have received no 1's, one 1, two 1's, and three 1's, respectively. We call this BDD construction procedure as Choose(n-1, m).

As for Condition (2), we first construct a BDD for a set of cycles, and then construct a BDD for prohibiting cycles. For constructing a BDD for cycles, we begin a set of edges in a face. Then, we repeat adding a new face and constructing cycle with the edges of the face. Figure 5 is the detail of this idea, and Figure 6 illustrates the first three iterations of Step 2.

In Procedure EnumerateCycles, we use f_{cycle} to denote the obtained set of cycles. In Step 1, f_{cycle} is set to be empty. In the first iteration of Step 2, we pick face F_1 , and add a cycle with its edges. (The cycle is illustrated with a bold line in Figure 6 (a).) The cycle (more precisely, the set of edges in the cycle) is set to f_1 . f_2 is empty since f_{cvcle} is empty. Now, the family f_{cvcle} contains the cycle of face F_1 . In the second iteration, we pick face F_2 , and its corresponding cycle is set to f_1 (see Figure 6 (b)). By combining the edges of F_2 with the already obtained cycle (the cycle in Figure 6 (a)), we can obtain a new cycle (i.e., the cycle contains the edges of F_1 and F_2). More precisely, if edge x_i of F_2 is not in the already obtained cycle, the edge exists in the new cycle. Otherwise, the edge does not exist in the new cycle. By adding the above two cycles, f_{cycle} becomes a family of the cycles in Figures 6 (a) and (b). In the following iterations, Step 2-2 combines the edges of F_i with already obtained cycles. (Although f_{cycle} may contain a set of edges that consists of two (or more) cycles, there is no influence on the next procedure, i.e., the construction of a BDD for prohibiting cycles.) In Step 3, we omit an empty set of edges from f_{cycle} . Note that the empty set is obtained as $f_{\text{empty}} := \bigwedge_{x_j \in X} \overline{x_j}$ and the set difference is obtained by $f_{\text{cycle}} \wedge \overline{f_{\text{empty}}}$.

Now, we have a family of cycles and will construct a BDD for prohibiting cycles. The complement of f_{cycle} is not sufficient for this task. We should prohibit a set $S \ (\subseteq X)$ of edges if S is in the monotone extension [5] of f_{cycle} , where the monotone extension of a family \mathcal{F} of sets is a family $\{T \mid \text{there exists a set } T' \in \mathcal{F} \text{ satisfying } T' \subseteq T\}$. Procedure MonotoneExtension in Figure 7 construct a BDD of the monotone

Procedure EnumerateCycles

Input: A Polyhedron. **Output:** A BDD representing a family of cycles in the give Polyhedron.

Step 1 (initialize). $f_{cycle} := 0$. Step 2 (iterate). For each face F_i , apply Steps 2-1, 2-2, and 2-3, where F_i has a set of edges $E_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$. Step 2-1. Construct a BDD of $f_1 := (\bigwedge_{x_j \in E_i} x_j) \land (\bigwedge_{x_j \in X \setminus E_i} \overline{x_j})$. Step 2-2. Construct a BDD of f_2 which is obtained from the BDD of f_{cycle} by exchanging the roles of 0-edges and 1-edges of the variable nodes labeled by $x_j \in E_i$. Step 2-3. Construct a BDD of $f_{cycle} := f_{cycle} \lor f_1 \lor f_2$. Step 3. Construct a BDD of $f_{cycle} \land (\bigvee_{x_i \in X} x_j)$, and output it.

Figure 5: Procedure EnumerateCycles to construct a BDD representing the set of cycles in a polyhedron.



Figure 6: Example of the execution of Step 2 in Procedure EnumerateCycles.

extension of a given BDD. By combining the above procedures, we can obtain the BDD representing a family of edge unfoldings: $f_{\text{unfolding}} := \text{Choose}(n - 1, m) \land \overline{\text{MonotoneExtension}(\text{EnumerateCycles})}$, i.e., a set difference $\text{Choose}(n - 1, m) \land \overline{\text{MonotoneExtension}(\text{EnumerateCycles})}$. We call this Algorithm 1-1.

Another characterization of edge unfoldings by Lemma 2 is as follows: A set S of cut edges leads to an edge unfolding if and only if (1) S consists of exactly n-1 edges and (2) all vertices are connected by the edges in S. Condition (2) is obtained by a small modification of the procedure for constructing a BDD of Hamiltonian cycles for traveling salesman problem [18]. We do not use the restriction that every vertex has exactly two edges, but use the restriction that every vertex has at least one edge. We call this Procedure EnumerateConnected. By combining this procedure with Pro**Procedure MonotoneExtension Input:** A BDD G representing f. (v is the root node of G.) **Output:** A BDD representing the monotone extension of f. Step 1 (termination). If f = 0 or f = 1, return G. Step 2 (recursion). Let G_0 and G_1 be the BDDs whose root nodes are 0-succ(v) and 1-succ(v), respectively. Construct BDDs G_{m0} and G_{m1} of $f_{m0} := \text{MonotoneExtension}(G_0)$ and $f_{m1} := \text{MonotoneExtension}(G_1).$ Step 3 (construction). Construct a BDD G_{m*} of $f_{m*} := f_{m0} \vee f_{m1}$. Then, construct a BDD G_m whose root node v_m is labeled by the same variable with node v, 0-succ (v_m) and 1-succ (v_m) are the root nodes of G_{m0} and G_{m*} , respectively. Output G_m .

Figure 7: Procedure MonotoneExtension to construct a BDD representing the monotone extension of f.

cedure Choose, we can obtain the BDD representing a family of edge unfoldings: $f_{\text{unfolding}} := \text{Choose}(n-1, m) \land \text{EnumerateConnected}$. We call this Algorithm 1-2.

The algorithms 1-1 and 1-2 enumerate sets of cut edges. In this family, different sets of cut edges may give the same edge unfolding. We omit mutually isomorphic edge unfoldings by the lexicographic order. For example, the two sets of cut edges in Figure 8 give the same edge unfolding. The sets of cut edges in Figure 8 are represented by assignments (a) 1011010010111 and (b) 110100101101, respectively, where the most significant (i.e., left most) bit corresponds to x_{12} and the least significant (i.e., right most) bit corresponds to x_1 . As (b) is lexicographically larger than (a), we omit (a). We can implement this process by manipulating BDDs.

3.2 Overlapping Check of Edge Unfoldings

Now, we have a family of edge unfoldings. All we have to do is to check whether each of them is overlappingfree or not. As for a tetrahedron, a cube, an octahedron,



Figure 8: Isomorphic edge unfoldings.

Figure 9: Neighboring faces.

and an icosahedron, the faces are equilateral triangles or squares. Thus, we can place all faces of their edge unfoldings on an equilateral triangular lattice or a square lattice. Unfortunately, the faces of a dodecahedron are pentagons, which cannot make a lattice. In this paper, we propose an algorithm that can be applied to any Platonic solid.

Given a set of cut edges, we can obtain the x-y coordinates for the centers of the faces. For example, an unfolding of a cube has the centers on (0,0), $(a\cos(\frac{3}{2}\pi), a\sin(\frac{3}{2}\pi))$, $(a\cos 0, a\sin 0)$, $(a\cos 0 + a\cos(\frac{1}{2}\pi), a\sin 0 + a\sin(\frac{1}{2}\pi))$, $(2a\cos 0, 2a\sin 0)$, $(a\cos \pi, a\sin \pi)$, where a is a distance between the centers of two neighboring faces. That is, $a = 2\sin\frac{n_f-2}{2n_f}\pi$ holds, where n_f is the number of vertices in a face. We assume that the circumscribed circle of a face has radius 1. We check where the distances between any two centers are larger than a by Mathematica.

For any pair of faces, we check whether their circumscribed circles overlap or not. We do not apply this check to neighboring faces. We call this Algorithm 3. We emphasize here that two faces may not overlap even if their circumscribed circles overlaps. Nevertheless, as shown in the next section, there exist no overlapping circumscribed circles for any pair of the faces. In other words, there are no overlapping faces except for neighboring ones.

Figure 9 illustrates that edges e_2 , e_3 and e_6 meet on a vertex in the original Platonic solid, and that $x_3 = x_6 = 1$ (i.e., e_3 and e_6 are cut edges) and $x_2 = 0$ (i.e., e_2 is not a cut edge). If a vertex has k cut edges, its surrounding faces are separated into k - 1 sets. (Recall that we have at least one cut edge for every vertex.) The faces in Figure 9 are separated into $\{F_1, F_2\}$ and $\{F_3\}$. If two faces are in the same set, they are called neighboring. In case k = 1, the surrounding faces are in a set, and thus, any two faces of the set are neighboring.

4 Experimental Results

We implemented the algorithms in Section 3 in programming language C. Table 1 gives a comparison between Algorithms 1-1 and 1-2. The computation time is measured on Intel(R) Core(TM) 2 Duo E7300 2.66GHz, 2GB memory, Ubuntu 10.04. Both algorithms give the same number of edge unfoldings: A cube and an octahedron have 384 edge unfoldings, which corresponds to the counting result in [15]. A dodecahedron and an icosahedron have 5,184,000 edge unfoldings, which corresponds to the counting result in [13]. Algorithm 1-1 runs faster than Algorithm 1-2, while both give the same number of unfoldings.

In Algorithm 1-2, we update the BDDs of F_1, \ldots, F_n for n-1 times, and each update of F_i requires n-1 OR operations. Thus, $O(n^3)$ OR operations are required

Table 1: Comparison between Algorithms 1-1 and 1-2.

	Algorithm 1-1		Algorithm 1-2		
	#unfoldings	$\begin{array}{c} \text{Time} \\ (\text{sec}) \end{array}$	#unfoldings	$\begin{array}{c} \text{Time} \\ (\text{sec}) \end{array}$	
Tetrahedron	16	0.01	16	0.01	
Cube	384	0.01	384	0.01	
Octahedron	384	0.01	384	0.01	
$\operatorname{Dodecahedron}$	5,184,000	0.01	$5,\!184,\!000$	0.61	
Icosahedron	5,184,000	0.02	$5,\!184,\!000$	2.91	



Figure 10: Partial list of edge unfoldings of a dodecahedron and an icosahedron.

in total. This is why the computation time grows so quickly for Algorithm 1-2. By omitting mutually isomorphic edge unfoldings, we obtain 2 unfoldings for a tetrahedron, 11 unfoldings for a cube and an octahedron, and 43,380 unfoldings for a dodecahedron and an icosahedron, respectively. Figure 10 is a partial list of enumerated edge unfoldings of a dodecahedron and an icosahedron.

Table 2 gives the total and average computation time of Algorithm 3. The average means the average computation time for an edge unfolding. For each edge unfolding, we have $O(F^2)$ pairs of faces to check whether they overlap or not, where F is the number of faces. The results on average computation time are almost same among all Platonic solids. Algorithm 3 says that, in any edge unfolding, there is no pair of faces (except for neighboring faces) that have overlapping circumscribed circles. In other words, edge unfoldings of Platonic solids are simple and nonoverlapping. The list of all cut-edges for Platonic solids, their corresponding sets of inequalities in Mathematica format, and a catalogue for edge unfoldings of Platonic solids are shown in http://www.al.ics.saitama-u.ac.jp/horiyama/ research/unfolding/.

5 Conclusions

We have proposed algorithms making use of BDDs for enumeration of edge unfoldings, and made a catalogue of edge unfoldings for Platonic solids. We furthermore proved that no edge unfolding of a Platonic solid overlaps. Our algorithms are applicable to Archimedean solids. It is also interesting to use ZDDs [16, 17] since it is also suitable for handling sets and families. We

	Total Time	Average Time
Tetrahedron	0.51s	0.25s
Cube	2.70s	0.25s
Octahedron	2.67s	0.24s
Dodecahedron	198m 39.16s	0.27s
Icosahedron	200m 55.15s	0.28s

Table 2: Computation Time for Algorithm 3.

also emphasize here that our approach for making use of BDDs is applicable to many other problems in computational geometry.

References

- S. B. Akers, Binary decision diagrams, *IEEE Trans. Com.*, C-27:509–516, 1978.
- [2] J. Akiyama, Tile-Makers and Semi-Tile-Makers, Math. Assoc. America, 114:602–609, 2007.
- [3] B. Aronov and J. O'Rourke, Nonoverlap of the star unfolding, *Disc. Comp. Geom.*, 8:219–250, 1992.
- [4] T. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, and S. Whitesides, Unfolding some classes of orthogonal polyhedra, *Proc. CCCG*, 70–71, 1998.
- [5] E. Boros, T. Ibaraki, and K. Makino, Monotone extensions of Boolean data sets, *Proc. ALT*, LNCS 1316, 161–175, 1997.
- [6] S. Bouzette, and F. Vandamme, The regular Dodecahedron and Icosahedron unfold in 43380 ways, Unpublished manuscript.
- [7] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Com.*, C-35: 677–691, 1986.
- [8] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, Sequential circuit verification using symbolic model checking, *Proc. DAC*, 46–51, 1990.
- [9] H. T. Croft, K. J. Falconer, and R. K. Guy, Unsolved Problems in Geometry, Springer-Verlag, Reissue edition, 1995.
- [10] E. D. Demaine and J. O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, 2007.
- [11] A. Dürer, Unterweysung der Messung mit dem Zirkel un Richtscheyt in Linien Ebnen uhnd Gantzen Corporen, 1525.

- [12] B. Grünbaum, Are your polyhedra the same as my polyhedra?, In B. Aronov, et al. (eds.), Discrete and Computational Geometry: The Goodman-Pollack Festschrift, 461–488, Springer, 2003.
- [13] C. Hippenmeyer, Die Anzahl der inkongruenten ebenen Netze eines regulären Ikosaeders, Elem. Math., 34:61–63, 1979.
- [14] T. Horiyama and T. Ibaraki, Reasoning with ordered binary decision diagrams, *Proc. ISAAC*, LNCS 1969, 120–131, 2000.
- [15] M. Jeger, Über die Anzahl der inkongruenten ebenen Netze des Würfels und des regulären Oktaeders, Elemente der Mathematik, 30:73–83, 1975.
- [16] D. E. Knuth, The art of computer programming, vol. 4, fascicle 1, Bitwise tricks & techniques, Binary decision diagrams, Addison-Wesley, 2009.
- [17] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *Proc. DAC*, 272–277, 1993.
- [18] S. Minato, Arithmetic Boolean expression manipulator using BDDs, Formal methods in system design, 10:221–242, Kluwer Academic, 1997.
- [19] S. Minato and H. Arimura, Frequent Pattern Mining and Knowledge Indexing Basedon Zero-Suppressed BDDs, Proc. KDID, 152–169, 2006.
- [20] J. Mitani and R. Uehara, Polygons Folding to Plural Incongruent Orthogonal Boxes, *Proc. CCCG*, 31–34, 2008.
- [21] D. M. Mount, On folding shortest paths on convex polyhedra, Technical Report 1495, Department of Computer Science, University of Maryland, 1985.
- [22] M. Namiki and K. Fukuda, Unfolding 3dimensional convex polytopes: A package for Mathematica 1.2 or 2.0, Mathematica Notebook, University of Tokyo, 1993.
- [23] C. Schevon and J. O'Rourke, A conjecture on random unfoldings, Technical report JHU-87/20, Johns Hopkins University, Baltimore, MD, 1987.
- [24] M. Sharir and A. Schorr, On shortest paths in polyhedral spaces, SIAM J. Comput., 15:193–215, 1986.
- [25] G. C. Shephard, Convex polytopes with convex nets, Math. Proc. Camb. Phil. Soc., 78:389–403, 1975.
- [26] I. Wegener, Branching programs and binary decision diagrams, Monographs on discrete mathematics and applications, 2000.

Development of Curves on Polyhedra via Conical Existence^{*}

Joseph O'Rourke[†]

Costin Vîlcu[‡]

Abstract

We establish that certain classes of simple, closed, polygonal curves on the surface of a convex polyhedron develop in the plane without overlap. Our primary proof technique shows that such curves "live on a cone," and then develops the curves by cutting the cone along a "generator" and flattening the cone in the plane. The conical existence results support a type of source unfolding of the surface of a polyhedron, described elsewhere.

1 Introduction

Nonoverlapping development of curves plays a role in unfolding polyhedra without overlap [2]. Any result on simple (non-self-intersecting) development of curves may help establishing nonoverlapping surface unfoldings. One of the earliest results in this regard is [7], which proved that the left development of a directed, simple, closed convex curve does not self-intersect. The proof used Cauchy's Arm Lemma. Here we extend this result to a wider class of curves without invoking Cauchy's lemma. Our results support a "source unfolding" based on these curves, described in [5].

Development. Let C be a simple, closed, polygonal curve on the surface of a convex polyhedron \mathcal{P} . For any point $p \in C$, let L(p) be the total surface angle incident to p at the left side of C, and R(p) the angle to the right side. The *left development* of C with respect to $x \in C$ is an isometric drawing $\overline{C_x}$ of C in the plane, starting from x, such that the angle to the left of $\overline{C_x}$ at every point in the plane is L(p). The *right development* is defined analogously. The left and right developments of a curve are different if C passes through one or more vertices of P. And in general the development depends upon the *cut point* x.

Curve Classes. To describe our results, we introduce a number of different classes of curves on convex polyhedra, which exhibit different behavior with respect to living on a cone. Define a curve C to be *convex* (to the left) if the angle to the left is at most π at every point p: $L(p) \leq \pi$; and say that C is a *convex loop* if this condition holds for all but one exceptional *loop point* x, at which $L(x) > \pi$ is allowed. Analogously, define C to be a *reflex curve* if the angle to one side (we consistently use the right side) is at least π at every point p: $R(p) \geq \pi$; and say that C is a *reflex loop* if this condition holds for all but an exceptional loop point x, at which $R(x) < \pi$.

The loop versions of these curves arise naturally in some contexts. For example, extending a convex path on \mathcal{P} until it self-intersects leads to a convex loop.

Summary of Results.

- 1. Every convex curve C left-develops to $\overline{C_x}$ without intersection, for every cut point x. This is a new proof of the result in [7].
- 2. There are convex loops C such that, for some x, the left-development $\overline{C_x}$ self-intersects. However, for every convex loop, there exists a y for which $\overline{C_y}$ left-develops without overlap.
- 3. Every reflex curve C right-develops to $\overline{C_x}$ without intersection, for every cut point x.
- 4. Every reflex loop C whose other side is convex right-develops to $\overline{C_x}$ without intersection, for every cut point x.

These results may be combined to reach conclusions about the left- and right-developments of the same curve: Every convex curve C that passes through at most one vertex, both left-develops, and right-develops without overlap, for every cut point x.

Living on a Cone. Our primary proof technique relies on the notion of a curve C "living on a cone," which is based on neighborhoods of C. An open region N_L is a vertex-free left neighborhood of C to its left if it includes C as its right boundary, and it contains no vertices of \mathcal{P} . In general C will have many vertex-free left neighborhoods, and all will be equivalent for our purposes. We say that C lives on a cone to its left if there exists a cone Λ and a neighborhood N_L so that

^{*}This paper is based largely on [8]

[†]Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu.

[‡]Institute of Mathematics "Simion Stoilow" of the Romanian Academy, P.O. Box 1-764, RO-014700 Bucharest, Romania. Costin.Vilcu@imar.ro.

 $C \cup N_L$ may be embedded isometrically onto Λ , and encloses the cone apex a.

A cone is an unbounded developable surface with curvature zero everywhere except at one point, its *apex*, which has total incident surface angle, called the *cone* angle, of at most 2π . Throughout, we will consider a cylinder as a cone whose apex is at infinity with cone angle 0, and a plane as a cone with apex angle 2π . We only care about the intrinsic properties of the cone's surface; its shape in \mathbb{R}^3 is not relevant for our purposes. So one could view it as having a circular cross section, although we will often flatten it to the plane.

We should remark that the cone on which a curve C lives has no direct relationship (except in special cases) to the surface that results from extending the faces of \mathcal{P} crossed by C.



Figure 1: A 4-segment curve C which lives on cone Λ_L to its left. One possible N_L is shown, and a generator g = ax is illustrated.

To say that $C \cup N_L$ embeds isometrically into Λ means that we could cut out $C \cup N_L$ and paste it onto Λ with no wrinkles or tears: the distance between any two points of $C \cup N_L$ on $(C \cup N_L) \cap \mathcal{P}$ is the same as it is on $(C \cup N_L) \cap \Lambda$. See Figure 1. We say that C lives on a cone to its right if $C \cup N_R$ embeds isometrically on the cone, where N_R is a vertex-free right neighborhood of C such that the cone apex a is inside (the image of) C. We will call the cones to the left and right of C, Λ_L and Λ_R respectively. We will see that all four combinatorial possibilities occur: C may not live on a cone to either side, it may live on a cone to one side but not to the other, it may live on different cones to its two sides, or live on the same cone to both sides.

Cone Generators and Visibility. A generator of a cone Λ is a half-line starting from the apex *a* and lying on Λ . A curve *C* that lives on Λ is visible from the apex if every generator meets *C* at one point. Although it is possible for a curve to live on a cone but not be visible from its apex, when we can establish visibility from the

apex, then cutting C at any point $x \in C$ will develop $\overline{C_x}$ without overlap.

2 Preliminary Tools and Lemmas

C partitions \mathcal{P} into two *half-surfaces*. We call the left and right half-surfaces P_L and P_R respectively, or P if the distinction is irrelevant. We view each half-surface as closed, with boundary C.

Curvature. The *curvature* $\omega(p)$ at any point $p \in \mathcal{P}$ is the "angle deficit": 2π minus the sum of the face angles incident to p. The curvature is only nonzero at vertices of \mathcal{P} ; at each vertex it is positive because \mathcal{P} is convex. The curvature at the apex of a cone is similarly 2π minus the cone angle.

Define a *corner* of curve C to be any point p at which either $L(p) \neq \pi$ or $R(p) \neq \pi$. Let c_1, c_2, \ldots, c_m be the corners of C, which may or may not also be vertices of \mathcal{P} . C "turns" at each c_i , and is straight at any noncorner point. Let $\alpha_i = L(c_i)$ be the surface angle to the left side at c_i , and $\beta_i = R(c_i)$ the angle to the right side. Also let $\omega_i = \omega(c_i)$ to simplify notation. We have $\alpha_i + \beta_i + \omega_i = 2\pi$ by the definition of curvature. These definitions will be used to further detail the relationships among the curve classes in Section 5.

The Gauss-Bonnet Theorem. We will employ this theorem in two forms. The first is that the total curvature of \mathcal{P} is 4π : the sum of $\omega(v)$ for all vertices v of \mathcal{P} is 4π . It will be useful to partition the curvature into three pieces. Let $\Omega_L(C) = \Omega_L$ be the total curvature strictly interior to P_L , Ω_R the curvature to the right, and Ω_C the sum of the curvatures on C (which is nonzero only at vertices of \mathcal{P}). Then $\Omega_L + \Omega_C + \Omega_R = 4\pi$.

The second form of the Gauss-Bonnet theorem relies on the notion of the "turn" of a curve. Define $\tau_L(c_i) =$ $\tau_i = \pi - \alpha_i$ as the left *turn* of curve *C* at corner c_i , and let $\tau_L(C) = \tau_L$ be the total (left) turn of *C*, i.e., the sum of τ_i over all corners of *C*. Thus a convex curve has nonnegative turn at each corner, and a reflex curve has nonpositive turn at each corner. Then $\tau_L + \Omega_L =$ 2π , and defining the analogous term to the right of *C*, $\tau_R + \Omega_R = 2\pi$.

Alexandrov's Gluing Theorem. In our proofs we use Alexandrov's theorem [1, Thm. 1, p. 100] that gluing polygons to form a topological sphere in such a way that at most 2π angle is glued at any point, results in a unique convex polyhedron.

Vertex Merging. We now explain a technique used by Alexandrov, e.g., [1, p. 240]. Consider two vertices v_1 and v_2 of curvatures ω_1 and ω_2 on \mathcal{P} , with $\omega_1 + \omega_2 < 2\pi$, and cut \mathcal{P} along a shortest path $\gamma(v_1, v_2)$ joining v_1 to
v_2 . Construct a planar triangle $T = \bar{v}' \bar{v}_1 \bar{v}_2$ such that its base $\bar{v}_1 \bar{v}_2$ has the same length as $\gamma(v_1, v_2)$, and the base angles are equal to $\frac{1}{2}\omega_1$ and respectively $\frac{1}{2}\omega_2$. Glue two copies of T along the corresponding lateral sides, and further glue the two bases of the copies to the two "banks" of the cut of \mathcal{P} along $\gamma(v_1, v_2)$. By Alexandrov's Gluing Theorem, the result is a convex polyhedral surface \mathcal{P}' . On \mathcal{P}' , the points v_1 and v_2 are no longer vertices because exactly the angle deficit at each has been sutured in; they have been replaced by a new vertex v'of curvature $\omega' = \omega_1 + \omega_2$ (preserving the total curvature). Figure 2(a) illustrates this. Here $\gamma(v_1, v_2) = v_1 v_2$ is the top "roof line" of the house-shaped polyhedron \mathcal{P} . Because $\omega_1 = \omega_2 = \frac{1}{2}\pi$, T has base angles $\frac{1}{4}\pi$ and apex angle $\frac{1}{2}\pi$. Thus the curvature ω' at v' is π . (Other aspects of this figure will be discussed later.)

Note this vertex-merging procedure only works when $\omega_1 + \omega_2 < 2\pi$; otherwise the angle at the apex \bar{v}' of T would be greater than or equal to π .



Figure 2: (a) C = (a, b, c, d) is a convex curve with angle $\frac{3}{4}\pi$ to the left at each vertex. The curvature at v_1 and at v_2 is $\frac{1}{2}\pi$. (b) Cutting along the generator from v' through the midpoint of ad and developing C shows that it lives on a cone with apex angle π at v'. (Base of \mathcal{P} is $3 \times \sqrt{2}$.)

Lemma 1 A curve C that lives on a cone Λ (say, to its left) uniquely determines that cone.

Proof. Sketch. The apex angle of any cone on which C lives must be $\alpha = 2\pi - \Omega_L$, where Ω_L is the total curvature inside and left of C. Imagining rolling out two distinct cones cut along a generator through the same point $x \in C$ leads to isometric unfoldings, showing that the cones are in fact identical. Details are in [5].

3 Convex Curves

The lemma below reproves the result from [7].

Lemma 2 Let C be a convex curve on \mathcal{P} , convex to its left. Then C lives on a cone Λ_L to its left side, whose apex a has curvature Ω_L , and so has cone apex angle $2\pi - \Omega_L$. C is visible from the apex a of Λ . **Proof.** Sketch. By the Gauss-Bonnet theorem, $\tau_L + \Omega_L = 2\pi$. Because $\tau_L \geq 0$ for a convex curve, we must have $\Omega_L \leq 2\pi$. If $\Omega_L < 2\pi$, we continually merge vertices in P_L until only one remains, at which point P_L is a pyramid, and therefore a cone. If $\Omega_L = 2\pi$, a slight alteration of the proof results in C living on a cylinder. Details are in [5].

Example 1. In Figure 2, the two vertices inside C, of curvature $\frac{1}{2}\pi$ each, are merged to one of curvature π , which is then the apex of a cone on which C lives.

Example 2. Figure 3(a) shows an example with three vertices inside C. \mathcal{P} is a doubly covered flat pentagon, and $C = (v_4, v_5, v_4)$ is the closed curve consisting of a repetition of the segment v_4v_5 . C has π surface angle at every point to its left, and so is convex. The curvatures at the other vertices are $\omega_1 = \pi$ and $\omega_2 = \omega_3 = \frac{1}{2}\pi$. Thus $\Omega_L = 2\pi$, and the proof of Lemma 2 shows that C lives on a cylinder. Following the proof, merging v_1 and v_2 removes those vertices and creates a new vertex v_{12} of curvature $\frac{3}{2}\pi$; see (b) of the figure. Finally merging v_{12} with v_3 creates a "vertex at infinity" v_{123} of curvature 2π . Thus C lives on a cylinder as claimed. If we first merged v_2 and v_3 to v_{23} , and then v_{23} to v_1 , the result is exactly the same, although not obviously so.



Figure 3: (a) A doubly covered flat pentagon. (b) After merging v_1 and v_2 . (c) After merging v_{12} and v_3 .

4 Convex Loops

Convex Loops and Cones. We first show that the technique that proved successful for convex curves cannot apply to all convex loops: not every convex loop lives on a cone. Consider the polyhedron \mathcal{P} shown in Figure 4(a), which is a variation on the example from Figure 2(a). Here C = (a, b, b', x, c', c, d) is a convex loop, with loop point x. The cone on which it should live is analogous to Figure 2(b): vertex merging of v_1 and v_2 again produces the cone apex v' whose curvature is π . But C does not "fit" on this cone, as Figure 4(b) shows; the apex a = v' is not inside C.

Overlapping development of convex loop. In light of the preceding negative result, it is perhaps not surpris-



Figure 4: (a) A convex loop C that does not live on a cone. (b) A flattening of the cone on which it should live. (Base of \mathcal{P} is 3×3 .)

ing that there are convex loops C and a point $x \in C$ such that $\overline{C_x}$ left-develops with overlap. Indeed Figure 5 shows an example where x is the loop point.



Figure 5: (a) \mathcal{P} with convex loop C. (b) $\overline{C_x}$ when cut at loop point x.

Visibility Points. Despite the negative result illustrated above, we can show that there always exists some cut point y that develops a convex loop without overlap.

Say that $y \in C$ is a visibility point for C if for every point $z \in C$ there is a shortest path joining y to z that remains interior to C except at its endpoints. The following proof sketch shows, roughly, that a convex loop C lives on the union of two cones (Case I), or on two cones separated by another region (Case II). This suffices to establish a non-overlapping development. The sketch relies at several points on our work on the star unfolding in [4].

Lemma 3 Every convex loop C has a visibility point y different from its loop point x, and $\overline{C_y}$ left-develops without overlap.

Proof. Sketch. Let τ_1 and τ_2 be the tangent directions of C at x, and consider $\mu_i = -\tau_i$.

Case I. Assume first there exists a shortest path $\gamma = xy$ from x to some $y \in C$ whose tangent direction at x lies between μ_1 and μ_2 ; see Figure 6. Then γ splits $P = P_L$ into two convex regions P_i sharing the common boundary point y, and hence (by Lemma 8 in [4])

y "sees" every point in P. Moreover, vertex merging in each P_i produces two cones Λ_i (of apices a_i) with common boundary γ .



Figure 6: Case 1: $\gamma = xy$ is a shortest path.

Claim 1. Cutting each cone along the generator $a_i y$ unfolds the union of cones without overlappings. Consequently, this develops C without overlap.

Case II. Assume now that Case I does not hold. Then P must contain a "fat digon" D, a concept from [4]. This is a region bounded by two shortest paths from x to some $y \in C$ whose angle at x covers all possible "splitting" γ between μ_1 and μ_2 . In this case what remains outside the digon is the union of two convex regions P_i , each visible from y. Moreover, D is itself completely visible from y (see Sec. 4.2 in [4]). Again we perform vertex merging in each P_i to obtain two cones, of apices a_i , which we unfold by cutting along a_iy .

We unfold D by the star unfolding with respect to y, and apply Lemma 7 in [4] to establish that the result lies inside some angle (at x).

Claim 2. We can join the unfoldings without overlappings. Consequently, this develops C without overlap. The proof of Claim 2 follows the one for Claim 1, with the additional fact that the star unfolding of the "fat digon" fits inside a circular sector at \bar{x} .

This result on convex loops is best possible in the sense that there are curves C that are convex except at two exceptional points, and for which $\overline{C_x}$ overlaps for every x.

5 Reflex Curves and Reflex Loops

For each corner c_i of a curve C, $\alpha_i + \omega_i + \beta_i = 2\pi$, where α_i and β_i are the left and right angles at c_i respectively, and ω_i is the curvature at c_i . When C is vertex-free, $\omega_i = 0$ at all corners, and the relationships among the curve classes is simple and natural: the other side of a convex curve is reflex, the other side of a reflex curve is convex. The same holds for the loop versions: the other side of a convex loop is a reflex loop (because $\alpha_m \geq \pi$ implies $\beta_m \leq \pi$, where c_m is the loop point), and the other side of a reflex loop. When C

includes vertices, the relationships between the curve classes are more complicated. The other side of a convex curve is reflex only if the curvatures at the vertices on C are small enough so that $\alpha_i + \omega_i \leq \pi$; C would still be convex even if it just included those vertices inside. The same holds for convex loops.

On the other hand, the other side of a reflex curve is always convex, because nonzero vertex curvatures only make the other side more convex. The other side of a reflex loop is a convex loop, and it is a convex curve if the curvature at the loop point c_m is large enough to force $\alpha_m \leq \pi$, i.e., if $\beta_m + \omega_m \geq \pi$.

This latter subclass of reflex loops—those whose other side is convex—especially interest us, because any convex curve that includes at most one vertex is a reflex loop of that type. All our results in this section hold for this class of curves.

Lemma 4 Let C be a curve that is either reflex (to its right), or a reflex loop which is convex to the other (left) side, with $\beta_m < \pi$ at the loop point c_m . Then C lives on a cone Λ_R to its reflex side, and is visible from its apex a. If $\Omega_R > 2\pi$, then the reflex neighborhood N_R is to the unbounded side of Λ_R , i.e., the apex of Λ_R is left of C; if $\Omega_R < 2\pi$, then N_R is to the bounded side, i.e., the apex of Λ_R is to the right side of C. If $\Omega_R = 2\pi$, $C \cup N_R$ lives on a cylinder.

Proof. Sketch. Because C is convex to its left, we have $\Omega_L \leq 2\pi$. Just as in Lemma 2, merge the vertices strictly in P_L to one vertex a. Let Λ_L be the cone with apex a on which C now lives.

The remainder of the proof alters Λ_L to Λ_R step-bystep with repeated insertions of "curvature triangles" to the left at each corner c_i of C. Each of these triangles is an isosceles triangle of apex angle ω_i , which flattens the surface at c_i without altering $C \cup N_R$. For a detailed proof, see [5]

Example 3. An example of a reflex loop that satisfies the hypotheses of Lemma 4 is shown in Figure 7(a). Here C has five corners, and is convex to one side at each. C passes through only one vertex of the cuboctahedron \mathcal{P} , and so it is reflex at the four non-vertex corners to its other side. Corner c_5 coincides with a vertex of \mathcal{P} , which has curvature $\omega_5 = \frac{1}{3}\pi$. Here $\alpha_5 = \beta_5 = \frac{5}{6}\pi$. Because $\beta_5 < \pi$, C is a reflex loop. We have $\Omega_L = \frac{2}{3}\pi$ because C includes two cuboctahedron vertices, u and v in the figure. $\Omega_C = \omega_5 = \frac{1}{3}\pi$. And therefore $\Omega_R = 3\pi$. The apex curvature of Λ_L is $\Omega_L = \frac{2}{3}\pi$, and the apex curvature of Λ_R is π . N_R lives on the unbounded side of this cone, which is shown shaded in Figure 7(b). Note the apex a is left of C, in accord with the lemma.



Figure 7: (a) A curve C of five corners passing through one polyhedron vertex. C is convex to one side, and a reflex loop to the other, with loop point c_5 , at which $\beta_5 = \frac{5}{6}\pi(=150^\circ) < \pi$. (b) The cone Λ_R with apex a is shaded.

6 Discussion

We summarize the results claimed in the Introduction in a theorem:

Theorem 5 On a convex polyhedron, every convex curve left-develops without overlap, and every reflex curve, and reflex loop whose other side is convex, rightdevelops without overlap, for every cut point. Every convex loop has some cut-point from which it leftdevelops without overlap.

Proving that a curve on a convex polyhedron lives on a cone is a powerful technique for establishing that these polyhedron curves develop without overlap. Even when a curve—such as a convex loop—does not live on a cone, still the cone perspective can help prove nonoverlapping development (Lemma 3).

Many questions remain.

Overlapping Developments. It is not the case that every curve that lives on a cone develops without overlap. Here we show that there exist C such that \overline{C}_x is non-simple for every choice of x. We provide one specific example, but it can be generalized.

The cone Λ has apex angle $\alpha = \frac{3}{4}\pi$; it is shown cut open and flattened in two views in Figure 8(a,b). An open curve $C' = (p_1, p_2, p_3, p_4, p_5)$ is drawn on the cone. Directing C' in that order, it turns left by $\frac{3}{4}\pi$ at p_2, p_3 , and p_4 . From p_5 , we loop around the apex a with a segment $S = (p_5, p_6, p'_5)$, where p'_5 is a point near p_5 (not shown in the figure). Finally, we form a simple closed curve on Λ by then doubling C' at a slight separation (again not illustrated in the figure), so that from p_5 it returns in reverse order along that slightly displaced path to p_1 again. Note that $C = C \cup S \cup C'$ is closed and includes the apex a in its (left) interior.



Figure 8: (a) Open curve $C' = (p_1, p_2, p_3, p_4, p_5)$ on cone of angle α , with cone opened. (b) A different opening of the same cone and curve. (c) Development of curve $\overline{C'}$ self-intersects.

Now, let x be any point on C from which we will start the development \overline{C}_x . Because C is essentially $C' \cup C'$, x must fall in one or the other copy of C', or at their join at p_1 . Regardless of the location of x, at least one of the two copies of C' is unaffected. So \overline{C}_x must include $\overline{C'}$ as a subpath in the plane.

Finally, developing C' reveals that it self-intersects: Figure 8(c). Therefore, \overline{C}_x is not simple for any x. Moreover, it is easy to extend this example to force selfintersection for many values of α and analogous curves. The curve C' was selected only because its development is self-evident.

Slice Curves. There are curves already known to develop without overlap that are not known to live on a cone. One particular class we could not settle are the slice curves. A *slice curve* C is the intersection of \mathcal{P} with a plane. Slice curves in general are not convex. The intersection of \mathcal{P} with a plane is a convex polygon in that plane, but the surface angles of \mathcal{P} to either side along C could be greater or smaller than π at differ-

ent points. Slice curves were proved to develop without intersection, to either side, in [6], so they are good candidates to live on cones. However, we have not been able to prove that they do.

Convex Loops. Although we have shown that there is some cut point from which every convex loop develops without overlap (Lemma 3), we have not determined all the cut points that enjoy this property.

Cone Curves. Finally, we have not obtained a complete classification of the curves on a cone that develop, for every cut point x, as simple curves in the plane. It would equally interesting to identify the class of curves on cones for which there exists at least one cut-point that leads to simple development. Indeed, the same questions for curves on a sphere are also unresolved [3].

References

- Aleksandr D. Alexandrov. Convex Polyhedra. Springer-Verlag, Berlin, 2005. Monographs in Mathematics. Translation of the 1950 Russian edition by N. S. Dairbekov, S. S. Kutateladze, and A. B. Sossinsky.
- [2] Erik D. Demaine and Joseph O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, 2007. http://www.gfalop.org.
- [3] Erik D. Demaine and Joseph O'Rourke. Open problems from CCCG 2009. In Proc. 22nd Canad. Conf. Comput. Geom., pages 83–86, 2010.
- [4] Jin-ichi Itoh, Joseph O'Rourke, and Costin Vîlcu. Star unfolding convex polyhedra via quasigeodesic loops. *Dis*crete Comput. Geom., 44:35–54, 2010.
- [5] Jin-ichi Itoh, Joseph O'Rourke, and Costin Vîlcu. Source unfoldings of convex polyhedra with respect to certain closed curves. Submitted, 2011.
- [6] Joseph O'Rourke. On the development of the intersection of a plane with a polytope. Comput. Geom. Theory Appl., 24(1):3–10, 2003.
- [7] Joseph O'Rourke and Catherine Schevon. On the development of closed convex curves on 3-polytopes. J. Geom., 13:152–157, 1989.
- [8] Joseph O'Rourke and Costin Vîlcu. Conical existence of closed curves on convex polyhedra. http://arxiv.org/ abs/1102.0823, February 2011.

Common Developments of Several Different Orthogonal Boxes

Zachary Abel^{*}

Erik Demaine[†] Martin Demaine[‡]

ine[‡] Hiroa

Hiroaki Matsui[§] (

Günter Rote[¶]

Ryuhei Uehara[∥]

Abstract

We investigate the problem of finding common developments that fold to plural incongruent orthogonal boxes. It was shown that there are infinitely many orthogonal polygons that fold to two incongruent orthogonal boxes in 2008. In this paper, we first show that there is an orthogonal polygon that fold to three boxes of size $1 \times 1 \times 5$, $1 \times 2 \times 3$, and $0 \times 1 \times 11$. Although we have to admit a box to have volume 0, this solves the open problem mentioned in literature. Moreover, once we admit that a box can be of volume 0, a long rectangular strip can be folded to an arbitrary number of boxes of volume 0. We next consider for finding common non-orthogonal developments that fold to plural incongruent orthogonal boxes. In literature, only orthogonal folding lines or with 45 degree lines were considered. In this paper, we show some polygons that can fold to two incongruent orthogonal boxes in more general directions.

1 Introduction

Since Lubiw and O'Rourke posed the problem in 1996 [4], polygons that can fold to a (convex) polyhedron have been investigated. In a book about geometric folding algorithms by Demaine and O'Rourke in 2007, many results about such polygons are given [3, Chapter 25]. Such polygons have an application in the form of toys and puzzles. For example, the puzzle "cubigami" (Figure 1) is developed by Miller and Knuth, and it is a common development of all tetracubes except one (of surface area 16). One of the many interesting problems in this area is that whether there exists a polygon that folds to plural incongruent orthogonal boxes. Biedl et al. answered "yes" by finding two polygons that fold to two incongruent orthogonal boxes [2] (see also [3, Figure



Figure 1: Cubigami.

25.53]). Later, Mitani and Uehara constructed infinite families of orthogonal polygons that fold to two incongruent orthogonal boxes [5]. However, it is open that whether there is a polygon that can fold to three or more boxes.

First, we give an affirmative answer to this open problem, at least in some weak sense. That is, we give a polygon that can fold to three incongruent orthogonal boxes of size $0 \times 1 \times 11$, $1 \times 1 \times 5$, and $1 \times 2 \times 3$. Note that one of the boxes is degenerate, as it has a side of length 0. Such a box is sometimes called a "doubly covered rectangle" (e.g., [1]). For boxes of positive volume, the existence of three boxes with a common unfolding is still open.

The polygon is found as a side effect of the enumeration of common developments of boxes of size $1 \times 1 \times 5$ and $1 \times 2 \times 3$. In the previous result by Mitani and Uehara [5], they randomly generated common developments of these boxes, and they estimated the number of common developments of these boxes at around 7000. However, they overestimated it since their algorithm did not exclude some symmetric cases. We enumerate all common developments of boxes of size $1 \times 1 \times 5$ and $1 \times 2 \times 3$, which can be found on a Web page¹. As a result, the number of common developments of these boxes is 2263. Among 2263 developments, the development in Figure 2 is the only one that can fold to $0 \times 1 \times 11$.

Once we admit that a box can be a doubly covered

^{*}Department of Mathematics, Massachusetts Institute of Technology, MA 02139, USA. zabel@math.mit.edu

[†]Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, MA 02139, USA. edemaine@mit.edu

[‡]Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, MA 02139, USA. mdemaine@mit.edu

[§]School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. s0910058@jaist.ac.jp

[¶]Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany. rote@inf.fu-berlin.de

^{||}School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

¹http://www.jaist.ac.jp/~uehara/etc/origami/net/ all-22.html



Figure 2: A common development of three different boxes. (a) Folding lines to make a $1 \times 1 \times 5$ box. (b) Folding lines to make a $1 \times 2 \times 3$ box. (c) Folding lines to make a $0 \times 1 \times 11$ box.

rectangle, we have a new view of this problem since a doubly covered rectangle seems to be easier to construct than a box of positive volume. Indeed, we show that a sufficient long rectangular strip can be folded to an arbitrary number of doubly covered rectangles.

Next we turn to another approach to this topic. In an early draft by Biedl et al. [2], they showed a common development of two boxes of size $1 \times 2 \times 4$ and $\sqrt{2} \times \sqrt{2} \times$ $3\sqrt{2}$ (Figure 3). In the development, two folding ways to two boxes are not orthogonal. That is, the set of folding lines of a box intersect the other set of folding lines by 45 degrees. This development motivates us to the following problem: Is there any common development of two incongruent boxes such that two sets of folding lines intersect by an angle different from 45 or 90 degrees? We give an affirmative answer to this question.

Figure 3: A common development of two different boxes by Biedl et al. [2]. (a) Folding lines to make a $1 \times 2 \times 4$ box. (b) Folding lines to make a $\sqrt{2} \times \sqrt{2} \times 3\sqrt{2}$ box.

2 Common orthogonal developments of boxes of size $1\times1\times5$ and $1\times2\times3$

For a positive integer S, we denote by P(S) the set of three integers a, b, c with $0 < a \leq b \leq c$ and ab+bc+ca = S, i.e., $P(S) = \{(a, b, c) \mid ab+bc+ca = S\}$. When we only consider the case that folding lines are on the edges of unit squares, it is necessary to satisfy |P(S)| > k to have a polygon of size 2S that can fold to k incongruent orthogonal boxes of positive volumes. The smallest S with $P(S) \ge 2$ is 11 and we have $P(11) = \{(1,1,5), (1,2,3)\}$. In this section, we concentrate at this special case. That is, we consider the developments that consist of 22 unit squares. Mitani and Uehara developed two randomized algorithms that try to find common developments of two different boxes [5]. Both algorithms essentially generate common developments randomly. Using the faster algorithm, they also estimated the number of common developments of the boxes of size $1 \times 1 \times 5$ and $1 \times 2 \times 3$ at around 7000. However, they overestimated it since their algorithm did not exclude some symmetric cases.

We develop another algorithm that tries all common developments of these boxes. For a common development P of the boxes, let P' be a connected subset of P. That is, P' be a set of unit squares and it produces a connected simple polygon. Then, clearly, we can stick P' on these two boxes without overlap. We use the term *common partial development* of the boxes to denote such a smaller polygon. For example, one unit square is the common partial development of the boxes of surface area 1, and a rectangle of size 1×2 is the common development of them of surface area 2, and so on. Let L_i be the set of common partial developments of the boxes of surface area *i*. Then $|L_1| = |L_2| = 1$, and $|L_3| = 2$, and one of our main results is $|L_{22}| = 2263$. The outline of the first algorithm is as follows:

	i	1	2	3	4	5	6	7	8	9
j	L_i	1	1	2	5	12	35	108	368	1283
<i>i</i> -on	ninos	1	1	2	5	12	35	108	369	1285
	i	1	0	1	1	12	2	13		14
l i	L_i	46	00	16	388	574	39	19338	3 60	4269
<i>i</i> -on	ninos	46	55	170	073	636	00	23859	1 90	1971
	i		15			16		17	1	18
j j	L_i	16	328	11	346	69043	51	82945	491	17908
<i>i</i> -ominos		34	265'	76	1307	79255	501	.07909	19262	22052
i	19)		20		21		22		
L_i	2776	413	88	8206	52	13303	37	2263		

Table 1: The number of common partial developments of two boxes $1 \times 1 \times 5$ and $1 \times 2 \times 3$ of surface area *i* with $1 \le i \le 22$. (For $1 \le i \le 18$, we give the number of *i*-ominos, for comparison.)

Input : None;

Output: Polygons that consist of 22 squares and fold to boxes of size $1 \times 1 \times 5$ and $1 \times 2 \times 3$;

1 let L_1 be a set of one unit square;

2 for $i = 2, 3, 4, \dots, 22$ do

3 $L_i := \emptyset;$

4 **for** each common partial development P in L_{i-1} **do**

5	for every polygon P + of size i obtained by
	attaching a unit square to P do
6	check if P^+ is a common partial
	development, and add it into L_i if it is a
	new one;
7	end
8	end

9 end

```
10 output L_{22};
```

We implemented the algorithm and obtain all common developments in L_{22}^2 . One can find all of them at http://www.jaist.ac.jp/~uehara/etc/origami/ net/all-22.html. All the values of L_i with $1 \le i \le 22$ are shown in Table 1. The first main theorem is as follows:

Theorem 1 The number of the common developments of boxes of size $1 \times 1 \times 5$ and $1 \times 2 \times 3$ into unions of unit squares is 2263.

3 Boxes including doubly-covered rectangles

3.1 Three boxes of surface area 22

Among the 2263 developments in Theorem 1, there is only one development that gives an affirmative answer



Figure 4: Tiling by the common development of three different boxes.

to the open problem in [5]:

Theorem 2 There is a common development of three boxes of size $1 \times 1 \times 5$, $1 \times 2 \times 3$, and $0 \times 1 \times 11$. Moreover, the development is a polygon such that (1) it can fold to three boxes by orthogonal folding lines, and (2) it forms a tiling.

Proof. The development is depicted in Figure 2. It is easy to see that all folding lines in Figure 2(a)-(c) are orthogonal. The tiling is given in Figure 4.

In Theorem 2(1), one may complain that some folding lines are not on the edges of unit squares. Then, split each unit square into four unit squares. On the refined development for three boxes of surface area 88, we again have the claims in Theorem 2 for the boxes of size $2 \times 2 \times 10$, $2 \times 4 \times 6$, and $0 \times 2 \times 22$, and all folding lines are on the edges of unit squares.

3.2 A rectangular strip can be folded to an arbitrary number of doubly-covered rectangles

Theorem 3 A rectangular $L \times 1$ paper (L > 1) can be folded into at least

 $2 + \lfloor L \rfloor$

different doubly-covered rectangles in at least

$$1 + \left\lfloor \frac{L}{4} \right\rfloor + \left\lceil \frac{L}{4} \right\rceil + \left\lfloor L \right\rfloor$$

different ways.

Proof. Figure 5a shows how a long ribbon of width 1 can be wrapped by "twisting" it around a rectangular strip. Here we show that we can obtain $\lfloor L \rfloor$ different doubly covered rectangles based on this way. First, we consider the points p_0, q_0, q_1, a, b, c , in Figure 5b). (Without loss of generality, we assume that $q_0b \ge q_1a$.) Let p_1 be the center of bc, and h_i is the point such that p_ih_i is a perpendicular of ab for i = 0, 1. We first observe that p_0a and bc are in parallel, the angles ap_0b and p_0bc are right angles, and p_0 is the center of q_0q_1 .

 $^{^{2}}$ The first program with a naive implementation was too slow. We tuned it with many technical tricks, and now it outputs L_{22} in around 10 hours.



Figure 5: Another way of folding a ribbon to a doublycovered rectangle

Thus, careful analysis tells us that $\triangle q_0 p_0 b$, $\triangle h_0 p_0 b$, and $\triangle h_1 p_1 a$ are congruent. By symmetry, $\triangle q_1 p_0 a$, $\triangle h_0 p_0 a$, and $\triangle h_1 p_1 b$ are also congruent. Hence the points $a p_0 b p_1$ form a rectangle. Therefore, the folding lines in Figure 5a) can be obtained by filling the rectangles like Figure 5b). Let k and w be the number of the rectangles and the length of the diagonal of the rectangle, respectively. Then, to obtain a feasible folding lines, we need $k \ge 1$, kw = L, and $w = ab \ge 1$. Therefore, for each $k = 1, 2, \ldots, \lfloor L \rfloor$, we can obtain a doubly covered rectangle of size $p_0 b$ and $k p_0 a$.

In addition, we have the two ways of folding the ribbon in half along the long axis (leading to a $L \times \frac{1}{2}$ rectangle) or along the short axis (leading to a $(L/2) \times 1$ rectangle).



Figure 6: Folding a ribbon to a doubly-covered rectangle. For better visibility, one side of the ribbon is shaded.

We next turn to another idea of folding. Figure 6a shows how a long ribbon of width 1 can be wrapped by "winding" it around a rectangular strip in such a way that the space between successive windings is equal to the width of the ribbon. By bending it backward at the end, as in Figure 6b–c, one obtains a doubly covered strip. Figure 6d shows the geometric construction: start with a right triangle ABC with the long side $d = BC = \cot \alpha + \tan \alpha$ on a long edge of the ribbon and the right angle A on the opposite edge. When the length L of the ribbon is an even multiple of d ($L = 2n \cdot d$), the folding will close into a doubly covered rectangle.



Figure 7: A different way of folding a ribbon to a doubly-covered rectangle

The minimum possible value of d is 2. d changes continuously with α , and any value of d larger than 2 can be obtained. So n, the number of repetitions, can take all values between 1 and $n_{\max} := \lfloor L/4 \rfloor$. For each n in this range, one can form a right triangle ABC with hypotenuse d = L/(2n) and legs $\frac{1}{2}(\sqrt{d^2 + 2d} \pm \sqrt{d^2 - 2d})$. One can use the longer leg as the wrapping direction, as in Figure 6, or the shorter leg, as in Figure 7. This leads to doubly covered rectangles of dimensions $(n \cdot \frac{1}{2}(\sqrt{d^2 + 2d} + \sqrt{d^2 - 2d})) \times \frac{1}{2}(\sqrt{d^2 + 2d} - \sqrt{d^2 - 2d})$ and $\frac{1}{2}(\sqrt{d^2 + 2d} + \sqrt{d^2 - 2d}) \times (n \cdot \frac{1}{2}(\sqrt{d^2 + 2d} - \sqrt{d^2 - 2d}))$.

For d = 2, the two possibilities coincide. So the total number of possibilities is $\lfloor L/4 \rfloor + \lceil L/4 \rceil - 1$. This equals $2\lfloor L/4 \rfloor$ except when L is a multiple of 4. In this case, we have to subtract 1 to compensate the overcounting for the case d = 2.

But we can see that each doubly covered rectangle by winding can be also obtained by twisting. Hence we obtain $2 + \lfloor L \rfloor$ different doubly covered rectangles in total.

4 Non-orthogonal polygons that fold to two incongruent boxes

Figure 8 shows a common unfolding of a $4 \times 4 \times 8$ box and a $\sqrt{10} \times 2\sqrt{10} \times 2\sqrt{10}$ box. It was obtained by solving an integer programming problem. The integer programming model formulates the problem of selecting a subset of 160 unit squares of the axis-aligned square grid underlying Figure 8, subject to the following constraints.

- 1. They should form a connected set in the plane.
- 2. When folded on the $4 \times 4 \times 8$ box, every square of the surface is covered exactly once. (There are no overlaps.)
- 3. When folded on the $\sqrt{10} \times 2\sqrt{10} \times 2\sqrt{10}$ box, every part of the surface is covered exactly once. Note

that the surface of the $\sqrt{10} \times 2\sqrt{10} \times 2\sqrt{10}$ box can be partitioned into 160 unit squares, which are however not aligned with the edges of the box. These squares result from folding the standard grid onto the box surface as shown in Figure 8. Some of these squares bend across an edge of the box.

The algorithm of Section 2 can be viewed as a systematic incremental way of finding all solutions to this problem.

The dimensions of the boxes were chosen as follows: A $1 \times 1 \times 2$ box has surface area 10, and a $1 \times 2 \times 2$ box has surface area 16. By scaling the first box with the factor 4 and the second box with the factor $\sqrt{10}$, we get two boxes with equal surface areas. A square lattice of side length $\sqrt{10}$ can be embedded on the standard integer grid by choosing the vector $\binom{1}{3}$ as a generating "unit vector".

The alignment of the two box unfoldings, with the symmetric layout of two "central" faces sharing two vertices, was fixed and was chosen by hand.

Figure 9 has been made from Figure 8 in an attempt to conceal the obvious folding directions. Further puzzles along these lines (for printing and cutting out) are given on a web page³.

5 Concluding remarks

It is an open question if a polygon exists that can fold to three or more orthogonal boxes such that all of them have positive volume. We are exploring the possibility to find such examples by our integer programming model of Section 4. If we take the approach in Section 2, the smallest S with $|P(S)| \ge 3$ is given by $P(23) = \{(1,1,11), (1,2,7), (1,3,5)\}$. Thus we need to construct polygons of surface area 46, which is much bigger than 22.

In Section 3.2, we use three different ideas for folding a rectangular ribbon R to a doubly-covered rectangle. It would be interesting to classify *all* ways of folding ribbons into doubly-covered rectangles. In fact, we can generalize the ideas of "twisting" and "winding"; see Figures 10 and 11. These folding ways correspond to a kind of the billiard ball problem on a rectangular table. Hence, to specify all the folding ways in the figures, we have to find all pairs of relatively prime integers p and q with $pq = \lfloor cL \rfloor$ for c = 1, 1/4. The number of such pairs seems to be related to the maximal value of prime divisors of numbers in reduced residue system for $\lfloor cL \rfloor$ ⁴.

⁴http://oeis.org/A051265



Figure 8: A common development of two different boxes. The set of folding lines for one box intersect the other set by neither 90 nor 45 degrees, but at $\arctan 3 \approx 72^{\circ}$.

Acknowledgments

This work was initiated at the 26th Bellairs Winter Workshop on Computational Geometry held February 11–18, 2011 in Holetown, Barbados, co-organized by Erik Demaine and Godfried Toussaint. We thank the other participants of that workshop—Oswin Aichholzer, Greg Aloupis, Prosenjit Bose, Mirela Damian, Vida Dujmović, Robin Flatland, Ferran Hurtado, Anna Lubiw, André Schulz, Diane Souvaine, and Andrew Winslow for helpful comments and for providing a stimulating environment.

References

- J. Akiyama. Tile-Makers and Semi-Tile-Makers. The Mathematical Association of Amerika, Monthly 114:602– 609, August-September 2007.
- [2] T. Biedl, T. Chan, E. Demaine, M. Demaine, A. Lubiw, J. I. Munro, and J. Shallit. Notes from the University of Waterloo Algorithmic Problem Session. September 8 1999.
- [3] E. D. Demaine and J. O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, 2007.

³http://www.inf.fu-berlin.de/~rote/Software/ folding-puzzles/



Figure 9: A common development of two different boxes. This has been obtained from Figure 8 by modi-fying the boundary.

- [4] A. Lubiw and J. O'Rourke. When Can a Polygon Fold to a Polytope? Technical Report Technical Report 048, Department of Computer Science, Smith College, 1996.
- [5] J. Mitani and R. Uehara. Polygons Folding to Plural Incongruent Orthogonal Boxes. In *Canadian Conference* on *Computational Geometry (CCCG 2008)*, pages 39–42, 2008.



Figure 10: A generalization of twist folding to a doubly covered rectangle.



Figure 11: A generalization of wind folding to a doubly covered rectangle.

Edge-Unfolding Orthogonal Polyhedra is Strongly NP-Complete

Zachary Abel^{*}

Erik D. Demaine[†]

Abstract

We prove that it is strongly NP-complete to decide whether a given orthogonal polyhedron has a (nonoverlapping) edge unfolding. The result holds even when the polyhedron is topologically convex, i.e., is homeomorphic to a sphere, has faces that are homeomorphic to disks, and where every two faces share at most one edge.

1 Introduction

An *edge unfolding* of a polyhedron consists of cutting the surface along a subset of its edges in such a way that the surface can be unfolded into one planar piece without overlap.¹ Edge unfoldings have a long history, dating back to Albrecht Dürer in 1525; see [3]. The most famous open question is whether every convex polyhedron has an edge unfolding, but nonconvex polyhedra are even more interesting for practical manufacturing applications. The theoretical study of such unfoldings began at CCCG 1998 [2] and CCCG 1999 [1]. Biedl et al. [2] found some orthogonal polyhedra with no edge unfoldings, but the examples had faces with holes or two faces that shared two edges. Bern et al. [1] found a triangulated polyhedron with no edge unfolding that is homeomorphic to a sphere, implying that the polyhedron is *topologically convex*—has the graph (1-skeleton) of a convex polyhedron. In the journal version of their CCCG 1999 paper [1], they asked for the computational complexity of deciding whether a given triangulated polyhedron has an edge unfolding.

In this paper, we settle the computational complexity of the closely related problem of deciding whether a topologically convex orthogonal polyhedron has an edge unfolding. Specifically, we prove this *Orthogonal Edge Unfolding* problem is strongly NP-complete.

2 Unique Coordinate Square Packing

The Square Packing problem asks, given n squares s_1, \ldots, s_n of side-lengths a_1, \ldots, a_n and a target distance d, whether there is some (non-overlapping) orthogonal packing of the squares s_i into a square of side-length d. This is known to be strongly NP-complete [4]. We first show that we may impose a few simplifying assumptions on the packings produced by this problem:

Definition 1 (Unique Coordinate Square Packing) An instance of the Unique Coordinate Square Packing (UCSP) promise problem has the form $(d, (a_1, ..., a_n))$, where all values are positive integers and $a_i \leq d-2$ for each $1 \leq i \leq n$. In a YES instance, there exists an orthogonal packing of n squares $s_1, ..., s_n$ of side-lengths $a_1, ..., a_n$ into the square $D = [0, d] \times [0, d] \subset \mathbb{R}^2$ satisfying the following additional properties:

- all vertices of all squares in the packing have integer coordinates,
- no two vertices of two different squares have the same x- or y-coordinate, and

• no square in the packing touches the boundary of D.

In a NO instance, there does not exist any orthogonal packing of the s_i into D.

Theorem 2 The Unique Coordinate Square Packing problem is strongly NP-hard.

The simple but technical proof is omitted from this extended abstract.

3 Overview

This section provides an overview of the detailed constructions to follow.

We first consider the problem of unfolding orthogonal polyhedra with boundary in Section 4, proving hardness by reduction from a UCSP instance $(d, (a_1, \ldots, a_n))$. We construct a polyhedron B with boundary (Figure 3) involving n squares b_i with side-lengths a_i (call these "blocks") surrounded by filler material. The polyhedron is designed to force the blocks to unfold inside a "cage" of shape $d \times d$, such that an unfolding exists if and only if there exists a square packing. (In the construction below, the blocks and cage are scaled up by a large factor q.) As the unfolding must remain connected, we use thin "wires" made from the filler material to "wind" around the blocks and connect them to the boundary

^{*}Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. zabel@math.mit.edu. Partially support by an MIT Mathematics Department Levinson Fellowship and an NSF Graduate Research Fellowship.

[†]Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, MA 02139, USA. edemaine@mit.edu. Partially supported by NSF CAREER award CCF-0347776.

¹We allow boundary edges to touch in unfoldings, requiring only that the interior of the cut surface does not overlap itself.



Figure 1: A depiction of an "atom," the polyhedral surface with 9×9 square boundary that enables universal wire unfolding as in Theorem 3. An atom is composed of 125 unit-square faces.

of the cage. The un-needed filler material winds itself out of the way. The universally windable wires are described and proved in Section 4.1, and the details of the unfolding are presented in Sections 4.2, 4.3, and 4.4.

In Section 5 we reduce to orthogonal polyhedra without boundary by extending B into a polyhedron C with the property that C has an unfolding if and only if Bdoes. This is accomplished with two U-shaped polygons (Figure 6) that must be separated from B to avoid overlap, which forces the extra material of C not to interfere with the unfolding of B.

4 Polyhedron With Boundary

In this section we show that the edge unfolding problem for an orthogonal polyhedron *with boundary* is NP-hard.

4.1 Atoms and Universal Wire Unfolding

As described in the overview, we require "winding wires" that can unfold into an arbitrarily chosen orthogonal path. We construct those here.

Define an **atom** as the polyhedral surface with boundary in Figure 1, whose boundary is a 9×9 square. The width of an atom is called the **atomic width**, $w_A = 9$. Atoms are named thus since they are the basic "winding wire" unit, and also due to their tiny size relative to many constructions to follow.

For a finite or infinite grid G of $u \times u$ squares in the xyplane and integers i, j, write $G[i, j]_u$ for the $(i, j)^{\text{th}}$ cell in G, i.e., the $u \times u$ cell positioned at (ui, uj). Similarly, if e is a directed line segment of length ℓ and u evenly divides ℓ , express e as the union of ℓ/u directed segments of length u and let $e[i]_u$ be the i^{th} such segment.

Define a wire W of length k in G as a simple path of connected squares in G: specifically, a collection of distinct squares $c_i = c_i(W)$ $(0 \le i \le k-1)$ in G and distinct, oriented edges $e_i = e_i(W)$ $(0 \le i \le k)$ such that e_i is the common edge of cells c_{i-1} and c_i for each $1 \le i \le k-1$, edge e_0 (the **starting edge**) is an edge of the **starting cell** c_0 , and e_k (the **ending edge**) is an edge of the **ending cell** c_{k-1} . Edge e_i is oriented to trace the boundary of c_{i-1} clockwise, or equivalently, to trace the boundary of c_i counterclockwise. (Use the former condition for e_k and the latter for e_0 .) It is convenient to discuss the **medial path** of a wire that connects the centers of $e_0, c_0, e_1, \ldots, c_{k-1}, e_k$ sequentially. The wire **turns right, straight, or left** at square c_i if the medial path turns right, straight, or left there.

If W is a wire of $w_A \times w_A$ squares in the x, y-plane, we can form the associated **wire of atoms** A(W), a polyhedral surface with boundary, by replacing each square c_i with an atom a_i pointing in the positive z-direction such that atoms a_{i-1} and a_i are connected along the edge corresponding to e_i . Each unit-length edge $e_i[s]_1$ (for $0 \le s \le w_A - 1 = 8$) corresponds to an edge of one or two unit-square faces on A(W).

Define a **flatom**² as a $w_F \times w_F$ square where $w_F = 27$ is the **flatomic width**. We will now show that wires of atoms can be universally unfolded in the following sense: roughly, any wire of k atoms can be unfolded inside any desired wire of k flatoms, while ensuring that the middle of each atom edge unfolds to the center of the corresponding flatom edge (or one unit away from center).

Theorem 3 Let W and W' be any two wires of length k with side-lengths w_A and w_F respectively. Then for each $t \in \{12, 13, 14\}$ there is an edge unfolding of the wire of atoms A(W) that lies inside W' such that $e_0(W)[4]_1$ (the middle unit edge of $e_0(W)$) unfolds to $e_0(W')[t]_1$ (i.e., the middle edge of $e_0(W')$ or one unit away) and $e_k(W)[4]_1$ unfolds to $e_k(W')[u]_1$ for some $u \in \{12, 13, 14\}$. Furthermore, this unfolding can be accomplished so that t and u have the same (resp., different) parity when W and W' together have an even (resp., odd) total number of left and right turns.

Proof. By induction on k, it suffices to prove only the case k = 1. There are thus 27 cases: W turns right, straight or left; W' turns right, straight, or left; and $e_0(W)[4]_1$ unfolds to $e_1(W')[12]_1$, $e_1(W')[13]_1$, or $e_1(W')[14]_1$. We label these unfoldings of an atom by a quadruple [X, Y, t, u], where $X, Y \in \{L, S, R\}^3$ indicate the directions of the turns of wires W and W' respectively, and t and u are as above. We must show that each of the 27 tuples (X, Y, t) appears in some unfolding [X, Y, t, u] with the required parity constraints on tand u. Up to mirror-reflection and direction reversal, only ten unfoldings are required⁴. Three of these are

³These are abbreviations for Left, Straight, and Right turns.

²short for "flat atom"

⁴For example, these ten suffice: [L, L, 13, 13], [L, L, 14, 12],



Figure 2: Three of the ten required unfoldings of an atom inside a flatom. Each unfolding is labeled with [X, Y, t, u] as described in the proof of Theorem 3. Solid black lines indicate cuts, dotted lines are valley folds, and dashed lines are mountain folds. Gray lines are uncreased edges.

illustrated in Figure 2, with the remaining seven to be included in the full version. $\hfill\square$

4.2 The Construction

Here we specify the polyhedron with boundary used in the reduction. The remainder of Section 4 is devoted to proving its correctness.

It will be useful to package atoms into a **molecule**: a 2×2 grid of atoms whose boundary is a $w_M \times w_M$ square, where $w_M = 2w_A = 18$. Much of the reduction below uses a molecule as a basic unit of construction.

Begin with a Unique Coordinate Square Packing instance $(d, (a_1, \ldots, a_n))$. Define $q = 2^5 \cdot 3^4 \cdot nd$ (a large scale factor), and let $q_M = q/w_M$ be the number of molecules that fit across a distance q. Also set t = $(n+1+a_1+\cdots+a_n)q_M$, and $p = 500(4dq_Mt+t^2)+3w_A$; these choices will be explained shortly. Define the polyhedron with boundary $B(d, (a_1, \ldots, a_n))$ as the surface shown in Figure 3, to be described in more detail presently. The diagram is oriented so that the positive x and y directions are right and up respectively, and zis out of the page.

The face F_{floor} in Figure 3a is a single polygon formed by creating a $w_F \times p$ hole, H_{drain} , and a $dq \times dq$ hole, H_{cage} , in a large square of size $\ell = p + dq + w_A$. The two faces F_{pipe}^1 and F_{pipe}^2 , of widths $2w_A$ and w_A respectively, exactly fill H_{drain} . Five (not flat!) polyhedral surfaces T_{bottom} , T_{left} , T_{mid} , T_{right} , T_{top} , shown in detail in Figure 3b, form the sides of the **tower**, T, which connects along the boundary of H_{cage} . The polyhedral surface T_{bottom} , whose boundary is a $dq \times tw_M$ rectangle, is a $dq_M \times t$ grid of molecules facing away from the tower except for the *n* square faces b_1, \ldots, b_n —called **bricks** of side-lengths qa_1, \ldots, qa_n , where brick b_i is positioned at $(q, (i + a_1 + \cdots + a_{i-1})q)$ relative to the bottom-left corner of T_{bottom} . (For T_{bottom} , "right" and "up" refer to the positive x and z directions, respectively, as in Figure 3b.) The parameter t was chosen so that these bricks exactly fit with q separation from each other and from the bottom and top edges. Recall that $a_i \leq d-2$ for each i, so there is at least q separation between each brick and the right edge of T_{bottom} . The other four sides of T, which have dimensions $dq \times tw_M$ or $dq \times dq$, are completely tiled with outward-facing molecules.

A single molecule has surface area 500, so the total surface area of T is strictly less than what the surface area would be if each brick were also tiled with molecules, namely $500(4dq_Mt + t^2) . Further$ more, the height of a molecule (out of the plane of itsboundary) is 7, so the projection of <math>T onto the plane containing F_{floor} extends beyond H_{cage} by only seven units. In particular, this projection lies strictly in the interior of the bounding box of F_{floor} , and is at least $p - 7 > w_A$ units away from the top edge of F_{floor} .

We will show in the next two subsections that $B(d, (a_1, \ldots, a_n))$ has an edge unfolding if and only if $(d, (a_1, \ldots, a_n))$ is a YES instance of UCSP. One direction is straightforward:

Lemma 4 If $(d, (a_1, \ldots, a_n))$ is a UCSP instance and $B(d, (a_1, \ldots, a_n))$ has an edge unfolding, then $(d, (a_1, \ldots, a_n))$ is a YES instance.

Proof. Fix some unfolding of $B = B(d, (a_1, \ldots, a_n))$. Let F_{pipe}^{1*} be the bottom height-1 subrectangle of F_{pipe}^{1} , and similarly for F_{pipe}^{2*} , and consider the polyhedral surface B^* obtained by replacing F_{pipe}^1 and F_{pipe}^2 with F_{pipe}^{1*} and F_{pipe}^{2*} . The unfolding of B induces an unfolding of

 $[\]begin{array}{l} [L,S,12,13], \; [L,S,14,13], \; [S,L,14,13], \; [S,L,12,13], \; [S,S,13,13], \\ [S,S,12,12], \; [R,L,13,13], \; [R,L,14,12]. \end{array}$



(a) The global structure of surface B. Faces $F_{\rm floor}$, $F_{\rm pipe}^1$, and $F_{\rm pipe}^2$ are each a single polygon, but tower T is mostly covered with molecules as detailed in part (b). For ease of viewing, the image here is not drawn to scale: the width of the two pipes is significantly smaller than the width of the tower T, for example.



(b) A detail of the tower, T. Each surface of T is entirely tiled with molecules except for T_{bottom} , which has bricks b_1, \ldots, b_n each a single square face—arranged as shown.

Figure 3: Detailed depiction of polyhedral surface B.

 B^* . In this unfolding of B^* , all of $T \cup \{F_{\text{pipe}}^{1*}, F_{\text{pipe}}^{2*}\}$ unfolds into $H_{\text{cage}} \cup H_{\text{drain}}$: indeed, p was chosen to ensure that the surface area of $T \cup \{F_{\text{pipe}}^{1*}, F_{\text{pipe}}^{2*}\}$ is strictly less than p, so there is not enough material to reach the top of H_{drain} . It follows that each brick b_i unfolds into $H_{\text{cage}} \cup H_{\text{drain}}$, and since H_{drain} is too narrow for the bricks, each b_i unfolds into H_{cage} . So there exists a packing of the b_i (with side-lengths $a_i \cdot q$) into H_{cage} (with side-length $d \cdot q$), which proves the Lemma. \square

4.3 Wiring the Tower

Think of T_{bottom} as a grid of molecules, with origin (0,0)at its lower left corner. In this section we demonstrate how to connect each brick to the bottom-left corner of the tower by a chain of molecules. For convenience, we ensure all such chains have the same length, L.

Brick b_i is positioned $(q, y_i q)$ $^{\rm at}$ where $y_i = i + a_1 + \dots + a_{i-1}$. Let $f_i \ (1 \le i \le n)$ be the lower edge of b_i , oriented left-to-right, and let f_0 be the lower edge of T_{bottom} , also oriented left-to-right. For $1 \leq i \leq n$ define $u_i = T_{\text{bottom}}[6i, 0]_{w_M}$; these are lined along the left of f_0 in T_{bottom} , spaced 6 molecules apart. Also let $v_i = T_{\text{bottom}}[q, y_i - 1]_{w_M}$ be the molecule just under the lower left corner of b_i in T_{bottom} .

Lemma 5 For any permutation σ of $\{1, \ldots, n\}$, there exist n non-overlapping wires W_1, \ldots, W_n of molecules in T_{bottom} such that each wire W_i has length exactly L = $4ndq_M$ and connects $c_0(W_i) = u_i$ to $c_L(W_i) = v_{\sigma i}$, with starting and ending edges along f_0 and $f_{\sigma(i)}$ respectively. Furthermore, no wire touches the two leftmost columns of molecules on T_{bottom} , and finally, the complement of the bricks b_i and wires W_i in T_{bottom} forms a single edge-connected polyomino of molecules.



step, overlap- modified with modified with zig-zags in the ping wires are detours around drawn from u_i bricks to avoid to $v_{\sigma(i)}$ with intersections. just one right turn.

(a) In the first (b) Wires are (c) Finally, each wire W_i is empty $a_{\sigma(i)}q_M/2 \times q_M/2$ grid next to brick $b_{\sigma(i)}$ in order to bring its length up to exactly $L = 4ndq_M$.

Figure 4: The three steps in the construction of molecule wires W_i of Lemma 5. The figures correspond to $\sigma(1) =$ 3, $\sigma(2) = 1$, $\sigma(3) = 2$, and $\sigma(4) = 4$.

Proof. Provisionally define each W_i as the wire that goes straight up from u_i and turns right to $v_{\sigma(i)}$, as in Figure 4a. As defined, these wires may intersect: the horizontal segment of W_i hits the vertical segment of W_i when i < j but $\sigma(i) > \sigma(j)$. To fix these, for each *i*, take all wires W_j that hit the horizontal part of W_i and insert a detour around brick b_i as illustrated in Figure 4b, keeping a 1-molecule gap between two detouring wires, and between these wires and b_i . Because $q_M > 4n$, there is ample room for the detours. Before the detours, each wire had length less than $t+q_M = (n+2+a_1+\cdots+a_n) q_M \leq 2ndq_M$, and each of fewer than *n* detours adds at most $2dq_M$ molecules, so the total length of each W_i is less than $4ndq_M$. Furthermore, by the parity of the positions of u_i and $v_{\sigma(i)}$, W_i has even length.

We now bring the length of each W_i up to exactly $L = 4ndq_M$. The $a_iq_M \times \frac{q_M}{2}$ grid of molecules to the left of b_i is empty, its bottom edge is adjacent to wire W_i , and the top and left edges are not adjacent to any wires. This grid has even width $q_M/2$, and by zig-zagging up and down in this region as shown in Figure 4c, we can add any even number of molecules up to $a_iq_M^2/2 > L$. This indeed allows each wire to reach its destination with total length exactly L. Finally, the left two columns of molecules were not touched by the wires, and the 1-molecule gaps inserted above ensure that the complement of the bricks and wires remains connected.

Now think of T (partially unfolded as in Figure 3b) as a grid of atoms, not molecules. Edges f_0, f_1, \ldots, f_n are as defined above, and let g be the bottom edge of $F_{\text{pipe}}^1 \cup F_{\text{pipe}}^2$ of length $3w_A$, oriented left to right.

Lemma 6 It is possible to write T as an interiordisjoint union of the following pieces:

- bricks b_1, \ldots, b_n ,
- wires X₁,..., X_n of atoms where each X_i has length exactly 4L and connects the bottom-right corner of molecule u_i (with starting edge along f₀) to the top-right corner of molecule v_{σ(i)} (with ending edge along f_{σ(i)}), and
- a wire X₀ connecting (T_{bottom}[1,0]_{w_A}, f₀[1]_{w_A}) to edge g[1]_{w_A} along with its adjacent atom on T_{top}.

Proof. Let wires of molecules W_i be as in Lemma 5. Wire X_i is obtained from W_i by starting at the bottom-right atom in molecule u_i and ensuring that $c_{4k}(X_i), \ldots, c_{4k+3}(X_i)$ are the four atoms in molecule $c_k(W_i)$ for each $0 \le k \le L - 1$. This can be done uniquely, and by parity, this wire X_i will terminate at the top-right atom of molecule $v_{\sigma(i)}$.

It remains to construct X_0 . Let W_0 be the wire of molecules in T that starts at $T_{\text{bottom}}[0,0]_{w_M}$ and traces the left edge of $T_{\rm bottom}$, the bottom, left, and top edges of T_{left} , and the left edge of T_{top} up to its top-left corner. Let X'_0 be the length-four wire of atoms that traces $c_0(W_0)$ as in Figure 5, and define X_0'' as the wire of atoms that follows the rest of W_0 as in the Figure. Let G be the region of T outside of the bricks b_1, \ldots, b_n and wires $X_1, \ldots, X_n, X'_0, X''_0$; by Lemma 5, G forms a connected polyomino of molecules. Lemma 5 guarantees that molecules $G[1,0]_{w_M}$ and $G[1,1]_{w_M}$ are in G, so pick any spanning tree S of the molecules in G in which these two molecules are connected. The desired wire of atoms X_0 is obtained by traversing X'_0 , walking all the way around S to the starting edge of X_0'' , and then following X_0'' . \square



Figure 5: A closeup of the bottom-left corner of T_{bottom} illustrating how to write G as a wire of atoms as in the proof of Lemma 6. This wire, X_0 , is formed by traversing the four atoms of wire X'_0 , then walking around the spanning tree S, and finally following atom-wire X''_0 .

4.4 Unfolding Surface B

We are now able to prove the converse of Lemma 5:

Lemma 7 If $(d, (a_1, \ldots, a_n))$ is a YES instance of UCSP, then $B(d, (a_1, \ldots, a_n))$ has an edge unfolding.

Proof. Think of H_{cage} as a $dq_F \times dq_F$ grid of flatoms, where $q_F = q/w_F$, with origin in the lower-left corner. Let f_0 and f_i $(1 \le i \le n)$ be the bottom edges of H_{cage} and brick b_i respectively, as above. Pick a packing of squares with side-lengths a_1, \ldots, a_n into $[0, d]^2$ with all the guarantees of the YES-promise of UCSP, and say the i^{th} square is positioned at (x_i, y_i) . Scale this up to a packing of bricks b_i into H_{cage} , with b_i positioned at $(x_iq_Fw_F, y_iq_Fw_F)$. Since the bricks b_i do not meet each other or the edges of H_{cage} , there is at least a q_F -flatom separation between them. For $1 \le i \le n$, define the flatoms $h_i = H_{\text{cage}}[4i, 0]_{w_F}$ (along f_0) and $k_i = H_{\text{cage}}[x_iq_F, y_iq_F - 1]_{w_F}$ (just under edge f_i).

Since the coordinates y_1, \ldots, y_n are all different, let σ be the permutation so that $y_{\sigma(1)} > y_{\sigma(2)} > \cdots > y_{\sigma(n)}$. It is possible to construct non-overlapping wires Z_1, \ldots, Z_n of flatoms in $H_{\text{cage}} \setminus \bigcup_{i=1}^n b_i$ where wire Z_i connects flatom h_i with its bottom edge to flatom $k_{\sigma(i)}$ with its top edge, and each wire has length exactly L. This can be accomplished with a method very similar to the proof of Lemma 5, so we omit these details.

Now we can describe the unfolding of $B = B(d, (a_1, \ldots, a_n))$. Using permutation σ defined here, apply Lemma 6 to B to obtain n + 1 wires of atoms X_0, X_1, \ldots, X_n . Each brick b_i will unfold to its position (x_iq, y_iq) in the UCSP unfolding above. For each $1 \leq i \leq n$, wires X_i and Z_i were designed so that their initial edges are centered on the same unit-length segment along f_0 : $e_0(X_i)[4]_1 = f_0[108i + 13]_1 = e_0(Z_i)[13]_1$, and similarly their final edges are centered in the same place

on $f_{\sigma(i)}$: $e_{4L}(X_i)[4]_1 = f_{\sigma(i)}[13]_1 = e_{4L}(Z_i)[13]_1$. Furthermore, wires X_i and Z_i have the same length, 4L, and each has an even number of left and right turns because their initial and final edges are parallel. It is thus possible, by Lemma 3, to unfold wire X_i into the region of H_{cage} described by Z_i while keeping X_i connected to both F_{floor} and $b_{\sigma(i)}$ along edges $f_0[108i + 13]_1$ and $f_{\sigma(i)}[13]_1$ respectively.

It remains to describe the unfolding of X_0 , F_{pipe}^1 and F_{pipe}^2 . In H_{cage} , the wires Z_1, \ldots, Z_n do not intersect leftmost column of modules, so define Z_0 as the wire of flatoms in $H_{\text{cage}} \cup H_{\text{drain}}$ that starts at $H_{\text{cage}}[0,0]_{w_F}$ with its bottom edge and proceeds straight up into H_{drain} with a total length equal to the length of X_0 . By Lemma 3, we may unfold X_0 into Z_0 while keeping the center of its final edge connected to F_{floor} at $f_0[13]_1$ and the center of its final edge connected to $F_{\text{pipe}}^1 \cup F_{\text{pipe}}^2$ along $g[13]_1$. In the unfolding, therefore, $F_{\text{pipe}}^1 \cup F_{\text{pipe}}^2$ simply slides up relative to F_{floor} and partially juts out of the top of H_{drain} .

5 Eliminating the Boundary

With the construction from the previous section, we are ready for the main result:

Theorem 8 The Orthogonal Edge Unfolding problem is strongly NP-complete.

Proof. This problem is in NP because any unfolding has integer coordinates and can thus be checked to be non-overlapping in polynomial time.

For hardness, we reduce from UCSP. For an instance $(d, (a_1, \ldots, a_n))$ of UCSP, define $B = B(d, (a_1, \ldots, a_n))$ as above, whose boundary is a square of side-length ℓ . Define the closed, orthogonal polyhedron $C = C(d, (a_1, \ldots, a_n))$ as specified in Figures 6a and 6b. The tower T (and the molecules on the tower) do not intersect the other faces of C, so this is a simple polyhedron. We will show C has an edge unfolding if and only if $(d, (a_1, \ldots, a_n))$ is a YES instance.

If $(d, (a_1, \ldots, a_n))$ is a YES instance, then by Lemma 7, there is an unfolding of *B* that fits inside the bounding box of F_{floor} except for $F_{\text{pipe}}^1 \cup F_{\text{pipe}}^2$ which sticks above the top edge. Then Figure 6b shows that this unfolding extends to an unfolding of all of *C*.

On the other hand, suppose C has an edge unfolding. Let t_1 be the edge shared by F_{pipe}^1 and U_1 , and similarly for t_2 . The shapes of U_1 and U_2 were chosen to force these two edges to be cut in the unfolding of C: indeed, if t_1 were not cut, then F_{pipe}^1 and U_1 would overlap in the plane; the argument for t_2 is the same. It follows that the unfolding of C induces a connected unfolding of B, so by Lemma 4, $(d, (a_1, \ldots, a_n))$ is a YES instance. \Box



(a) The faces U_1 and U_2 must be cut away from B in any unfolding of C in order to avoid overlapping F_{floor} , F_{pipe}^1 , or F_{pipe}^2 .



(b) A partial unfolding of C showing that any unfolding of B extends to an unfolding of C.

Figure 6: The polyhedron $C = C(d, (a_1, \ldots, a_n))$ without boundary.

References

- M. Bern, E. D. Demaine, D. Eppstein, E. Kuo, A. Mantler, and J. Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory and Applications*, 24(2):51–62, February 2003.
- [2] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceed*ings of the 10th Canadian Conference on Computational Geometry, Montréal, Canada, August 1998.
- [3] E. D. Demaine and J. O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, July 2007.
- [4] J. Y.-T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into a square. *Journal* of Parallel and Distributed Computing, 10(3):271–275, 1990.

A Topologically Convex Vertex-Ununfoldable Polyhedron

Zachary Abel*

Erik D. Demaine[†]

Martin L. Demaine[†]

Abstract

We construct a polyhedron that is topologically convex (i.e., has the graph of a convex polyhedron) yet has no vertex unfolding: no matter how we cut along the edges and keep faces attached at vertices to form a connected (hinged) surface, the surface necessarily unfolds with overlap.

1 Introduction

Polyhedron unfolding has a long history dating back to Albrecht Dürer in 1525; see [3]. In general, the goal is to cut along a one-dimensional subset of the polyhedron's surface to enable the remainder of the surface to unfold into the plane without overlap. An edge unfolding consists of cutting along a subset of the edges of the polyhedron, while keeping the surface interiorconnected; the planar unfolding is then uniquely determined by the development (local unfolding) of the intrinsic metric in the plane. A vertex unfolding consists of cutting along a subset of the edges, typically all of them, while keeping the faces connected together via shared vertices (without any crossing connections at the vertices); the planar unfolding is no longer unique, but rather acts like a hinged dissection, with faces able to rotate around shared vertex hinges.

Vertex unfolding was introduced in [2] as a less restrictive form of edge unfolding. They proved that every triangulated manifold (in any dimension, though we focus here on 2-manifolds in 3D) has a vertex unfolding. This result shows that vertex unfolding is more powerful than edge unfolding, as there are triangulated polyhedra that are *edge-ununfoldable* (have no edge unfolding) [1].

In this paper, we solve the "obvious question left open" by [2]: to what extent is the assumption of triangular faces necessary for vertex unfolding? Specifically, they asked whether every polyhedron with simply connected faces has a vertex unfolding, and whether every polyhedron with convex faces has a vertex unfolding.



Figure 1: The polyhedron P is a union of two identical overlapping triangular prisms, and—with proper dimension choices—has no vertex unfolding. The labeled points have coordinates A = (1,2,1), B = (1,-2,1),C = (5,-2,1), D = (5,2,1), E = (2,1,-1), F =(2,5,-1), G = (0,5,3); the rest can be derived from symmetries around the z-axis and the lines $x = \pm y$ in the xy-plane.

We prove that the answer to the first problem is "no", though the second problem remains open.

More precisely, we construct "topologically convex" vertex-ununfoldable polyhedra, strengthening the CCCG 1999 result of edge-ununfoldable polyhedra [1]. A polyhedron is *topologically convex* if its graph (1skeleton) is the graph of a convex polyhedron, or equivalently by Steinitz's Theorem, it is 3-connected and planar. In terms of the polyhedron's surface, topological convexity is equivalent to requiring that every face is homeomorphic to a disk (as they are in a convex polyhedron), and that every two faces meet at one edge, one vertex, or not at all (as they would in a convex polyhedron). In particular, topological convexity forbids the example of a small box attached in the middle of a face of another box, which is the only previously known vertex-ununfoldable polyhedron [2].

2 Vertex-Ununfoldable Polyhedra

We present two related topologically convex vertexununfoldable polyhedra. Our first example, P, is sim-

^{*}Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. zabel@math.mit.edu. Partially support by an MIT Mathematics Department Levinson Fellowship and an NSF Graduate Research Fellowship.

[†]Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, MA 02139, USA. {edemaine, mdemaine}@mit.edu. Partially supported by NSF CAREER award CCF-0347776.



Figure 2: If faces S_1 and S_3 were hinged at A, they must be in this configuration by Observation 2. But as there are overlaps, this is not allowed.

ply the union of two overlapping, identical triangular prisms, as shown in Figure 1. For concreteness's sake, we have listed coordinates of the labeled vertices, and the rest can be inferred from symmetries. To prove that P has no vertex unfolding, we make two self-evident observations:

Observation 1 If two polygons T_1 and T_2 have two vertices $v_1 \in T_1$ and $v_2 \in T_2$ whose angles add to more than 360°, then these vertices cannot be hinged in the plane without the polygons overlapping.

Observation 2 If the angles at v_1 and v_2 add to exactly 360° , and if these vertices are hinged without overlap in the plane, then they must be oriented to exactly cover the 360° surrounding the hinge.

Notice these obstructions to vertex unfoldings are entirely local in nature, involving only two polygons joined at a vertex.¹ These observations alone are enough to prove our claim:

Theorem 3 Polyhedron P has no vertex unfolding.

Proof. We will show that no lightly shaded face (as in Figure 1) can connect to a dark face in any planar vertex hinging of the faces, and therefore any proposed vertex unfolding is disconnected. Indeed, any light-dark connection must happen at one of the eight central vertices, and as they are all identical under symmetry, we may focus on vertex A. Because $\alpha > 270^{\circ}$ and $\beta = 90^{\circ}$, Observation 1 implies that S_1 and S_2 cannot hinge at A. Because $\alpha + \gamma = 360^{\circ}$, by Observation 2, if S_1 and S_3 were hinged at A then they must be hinged as in Figure 2. But the dimensions were chosen so that these polygons would overlap in this configuration.

By contrast, if the unfolding is allowed to have two connected components, then an *edge* unfolding is possible, as in Figure 3. Also, the use of Observation 2 required more global knowledge than just the vertex angles: the shapes of polygons S_1 and S_3 were crucial. Indeed, if AD (and all symmetric copies) were chosen shorter, then an edge unfolding of P would be possible, as in Figure 4.



Figure 3: An edge unfolding of P into two connected components.



Figure 4: If the prisms were shorter, an edge unfolding would exist, as depicted here.

In fact, Observation 1 alone is sufficient to provide a vertex-ununfoldable polyhedron. Perturb polyhedron P to a new polyhedron P' by increasing γ slightly (while maintaining symmetry) so that $\alpha + \gamma' > 360^{\circ}$; this also increases β slightly to β' . Such a polyhedron P' is shown in Figure 5. Because $\alpha + \beta', \alpha + \gamma' > 360^{\circ}$, it follows by Observation 1 that S'_1 cannot hinge to S'_2 or S'_3 at vertex A, so as before, any vertex unfolding must be disconnected:

Theorem 4 Polyhedron P' has no vertex unfolding.

3 Open Questions

The foremost open question concerning vertex unfolding is to find the largest natural class of polyhedra that always admit vertex unfoldings. We have shown here that topologically convex is too large a class. In fact, topologically convex and star shaped is too large, because both P and P' are star shaped—in both cases, the origin can see the entire polyhedron.

Another natural question, posed in [2], is whether (topologically convex) polyhedra with convex faces admit vertex unfoldings. We conjecture that the answer

 $^{^1\}mathrm{For}$ the related edge-unfolding problem, these are called 1- local obstructions [4].



Figure 5: Polyhedron P' is obtained from P by moving vertex C to C' = (5, -3, 3) and similarly for its symmetric copies. This polyhedron has no vertex unfolding based solely on the fact that $\alpha + \beta' > 360^{\circ}$ and $\alpha + \gamma' > 360^{\circ}$.

is "no", but the methods used in this paper cannot be directly extended. Indeed, any vertex of such a polyhedron with negative curvature must have at least four incident faces, any two of which could potentially remain connected, so the local conclusions are not as strong.

Finally, we echo an open problem implicit in [2] and explicit in [3, Open Problem 22.20]: does every convex polyhedron have a vertex unfolding? This is a weaker form of the famous convex edge-unfolding conjecture [3].

4 Acknowledgments

This work was begun at the 26th Bellairs Winter Workshop on Gomputational Geometry in February, 2011, and we are grateful to the organizers—Godfried Toussaint and the second author—and participants of the Workshop for providing a stimulating and productive atmosphere.

References

- [1] M. Bern, E. D. Demaine, D. Eppstein, E. Kuo, A. Mantler, and J. Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory* and Applications, 24(2):51–62, February 2003. Originally appeared at CCCG 1999.
- [2] E. D. Demaine, D. Eppstein, J. Erickson, G. W. Hart, and J. O'Rourke. Vertex-unfolding of simplicial manifolds. In *Discrete Geometry: In Honor of W. Kuperberg's* 60th Birthday, pages 215–228. Marcer Dekker Inc., 2003. Originally appeared at SoCG 2002.

- [3] E. D. Demaine and J. O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, July 2007.
- [4] B. Lucier. Local overlaps in special unfoldings of convex polyhedra. *Computational Geometry: Theory and Applications*, 42(5):495–504, 2009. Originally appeared at CCCG 2006.

Isoperimetric Triangular Enclosure with a Fixed Angle

Prosenjit Bose^{*}

Jean-Lou De Carufel*

Abstract

Given a set S of n > 2 points in the plane (in general position), we show how to compute in $O(n^2)$ time, a triangle T with maximum (or minimum) area enclosing S among all enclosing triangles with fixed perimeter P and one fixed angle ω . We show that a similar approach can be used to compute a triangle T with maximum (or minimum) perimeter enclosing S among all enclosing triangles with fixed area A and one fixed angle ω .

1 Introduction

The classical isoperimetric problems are

- 1. Of all plane figures of equal area, what is the one with minimum perimeter?
- 2. Of all plane figures of equal perimeter, what is the one with maximum area?

Several related problems and a complete discussion on the foundations and applications of these problems together with the proof of the following result can be found in Polya [7].

Theorem 1 (Isoperimetric Theorem)

- 1. Of all plane figures of equal area, the circle has minimum perimeter.
- 2. Of all plane figures of equal perimeter, the circle has maximum area.
- 3. 1 and 2 are equivalent.
- 4. Let n > 2 be a fixed integer. Of all n-gons of equal area, the regular n-gon has minimum perimeter.
- 5. Let n > 2 be a fixed integer. Of all n-gons of equal perimeter, the regular n-gon has maximum area.

Given a fixed area (respectively perimeter), there is no upper bound (respectively lower bound) on the perimeter (respectively area) a plane figure can have. In this paper, we are interested in figures that enclose a set of at least 3 non-collinear points. Then it is relevant to maximize the perimeter given a fixed area (respectively minimize the area given a fixed perimeter). We refer to these four isoperimetric problems as **FIP**.

Let ω be a fixed angle with $0 < \omega < \pi$. A triangle that has an angle ω is called an ω -triangle. In this paper, we study the **FIP** with two additional constraints: (1) the plane figures we consider are ω -triangles, and (2) they must enclose a given set S of n points.

These problems are a variant of the problems studied in [1, 2, 3, 4, 5, 6, 8, 9]. Most of these problems can be solved in linear time when the input is a convex ngon or in $O(n \log n)$ time when the input is a set of n points because of an interspersing lemma proper to each of these problems. Essentially, an interspersing lemma states that given a local extremum, if we turn clockwise around the convex hull of the set of points, then we will find all the other local extrema also in clockwise order (there is no need to backtrack). So it takes $O(n \log n)$ time to compute the convex hull, then it takes O(n) time to compute one local extremum, then it takes O(n) time to compute all the other local extrema and finally, it takes O(n) time to compute the global extrema. Unfortunately, such a lemma does not hold in the isoperimetric case. Our solution to the **FIP** takes $O(n^2)$ time. We explain in Section 5 why the canonical interspersing lemma does not apply, though we do not have a proof of a quadratic lower bound. The FIP can also be compared to the following problem (see [7], p.180): "Given an angle (the infinite part of a plane between two rays drawn from the same initial point). Find the maximum area cut off from it by a line of given length.". Note that if the fixed perimeter or the fixed area is too small, no solution exists.

2 Preliminary Results

Let $T = \triangle bqc$ be an ω -triangle with $\angle bqc = \omega$. Denote the area and the perimeter of T by A and P respectively. Let x = |bq|, y = |qc| and z = |bc|. Therefore,

$$P = x + y + z ,$$

$$A = \frac{1}{2}xy\sin(\omega) ,$$

$$z^{2} = x^{2} + y^{2} - 2xy\cos(\omega)$$

from which

$$x = \frac{P^2 \sin(\omega) + 4A(1 + \cos(\omega))}{4P \sin(\omega)} \tag{1}$$

^{*}School of Computer Science, Carleton University. This research was partially supported by NSERC (Natural Sciences and Engineering Research Council of Canada). {jit,jdecaruf}@cg.scs.carleton.ca

$$y = \frac{\sqrt{(P^2 \sin(\omega) + 4A(1 + \cos(\omega)))^2 - 32AP^2 \sin(\omega)}}{4P \sin(\omega)}$$
$$y = \frac{P^2 \sin(\omega) + 4A(1 + \cos(\omega))}{4P \sin(\omega)}$$
$$+ \frac{\sqrt{(P^2 \sin(\omega) + 4A(1 + \cos(\omega)))^2 - 32AP^2 \sin(\omega)}}{4P \sin(\omega)}$$
$$z = \frac{P^2 \sin(\omega) - 4A(1 + \cos(\omega))}{2P \sin(\omega)}.$$

Then, from (1), we have

$$A(x) = \frac{Px\sin(\omega)(P-2x)}{4(P-x(1+\cos(\omega)))} ,$$

$$P(x) = \frac{2A+x^2\sin(\omega)}{x\sin(\omega)} + \frac{\sqrt{4A^2+x^4\sin^2(\omega)-4x^2A\sin(\omega)\cos(\omega)}}{x\sin(\omega)}$$

With standard calculus techniques, we can prove the following properties (refer to Subsection 2.1). If P is fixed, then A(x) is defined for all $0 < x < \frac{1}{2}P$. It is increasing for $x \in \left[0, \frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P\right]$ and decreasing for $x \in \left[\frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P, \frac{1}{2}P\right]$. Thus the area is a unimodal function of x. When $x = \frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P$, T is isosceles with x = y. As for P(x), if A is fixed, then it is defined for all x > 0. It is decreasing for $x \in \left[0, \frac{\sqrt{2}\sqrt{A}}{\sqrt{\sin(\omega)}}\right]$ and increasing for $x \in \left[\frac{\sqrt{2}\sqrt{A}}{\sqrt{\sin(\omega)}}, \infty\right]$. Thus the perimeter is a unimodal function of x. When $x = \frac{\sqrt{2}\sqrt{A}}{\sqrt{\sin(\omega)}}$, $x \in \left[0, \frac{\sqrt{2}\sqrt{A}}{\sqrt{\sin(\omega)}}\right]$ and increasing for $x \in \left[\frac{\sqrt{2}\sqrt{A}}{\sqrt{\sin(\omega)}}, \infty\right]$.

From the previous discussion, we see that the **FIP** can be solved by focusing on the length of one of the sides of the ω -triangle. The angle ω of the desired ω -triangle is part of an ω -wedge.

Definition 1 (ω -Wedge) Let q be a point in the plane. Let Δ_1 and Δ_2 be two rays emanating from q such that the smallest angle between Δ_1 and Δ_2 is ω . The closed set formed by q, Δ_1 , Δ_2 and the points between Δ_1 and Δ_2 is an ω -wedge, denoted $W(\omega, q, \Delta_1, \Delta_2)$. The point q is the apex of the ω wedge. An ω -wedge W is said to enclose a convex ngon Q when $Q \subseteq W$ and both Δ_1 and Δ_2 are tangent to Q.

Therefore the vertex q of the desired ω -wedge is on an ω -cloud.

Definition 2 (ω -Cloud) By rotating an enclosing ω wedge around a convex n-gon Q, the apex traces a sequence of circular arcs. This sequence is called an ω -cloud, denoted Ω (refer to Figure 1). The circu-



Figure 1: Ω is the $\frac{1}{2}\pi$ -cloud of Q.

lar arcs of Ω are labelled in clockwise order by Γ_j for $0 \leq j \leq n'-1$. We note that n' = O(n) [3].

The proof of the following Lemma is similar to the proof of Lemma 1 in [1].

Lemma 2 Let A and P be two positive real numbers. Take $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$.

1. Consider the set of ω -triangles $\triangle bqc$ with perimeter P such that $b \in \Delta_1$ and $c \in \Delta_2$. The side bc of these ω -triangles are tangent to a common circle with radius $r_P = \frac{1}{2}P\tan(\frac{1}{2}\omega)$ (refer to Figure 2). We call this circle the perimeter circle of W and we



Figure 2: A $\frac{1}{2}\pi$ -wedge together with $\frac{1}{2}\pi$ -triangles $\triangle bqc$ with perimeter P such that $b \in \Delta_1$ and $c \in \Delta_2$.

denote it by C_P . The center of C_P is on the angle bisector of W and Δ_1 (respectively Δ_2) is tangent to C_P at t_1 (respectively at t_2) where $|qt_1| = \frac{1}{2}P$ (respectively $|qt_2| = \frac{1}{2}P$).

2. Consider the set of ω -triangles \triangle bqc with area Asuch that $b \in \Delta_1$ and $c \in \Delta_2$. The sides bc of all these ω -triangles are tangent to a common hyperbola with asymptotes Δ_1 and Δ_2 (refer to Figure 3). We call this hyperbola the area hyperbola of W and we denote it by \mathcal{H}_A . The center of \mathcal{H}_A is on q.

In this paper, we explain in detail how to find an ω triangle of minimum and maximum area with fixed perimeter. The solution when the area is fixed is almost identical since both A(x) and P(x) are unimodal functions. We use a technique similar to the one of Bose



Figure 3: A $\frac{1}{2}\pi$ -wedge together with $\frac{1}{2}\pi$ -triangles $\triangle bqc$ with area A such that $b \in \Delta_1$ and $c \in \Delta_2$.

and De Carufel in [2]. The main difference is in the lack of interspersing lemma (refer to Section 5). If the input is a set S of $n \ge 3$ non-collinear points, we first compute the convex hull of S. In what follows, we show how to solve the **FIP** when the input is a convex n-gon Q. Moreover, $Q = int(Q) \cup \delta Q$, where int(Q) is the interior of Q and δQ is the boundary of Q.

2.1 Analysis of A(x) and P(x)

Given

$$A(x) = \frac{Px\sin(\omega)(P-2x)}{4(P-x(1+\cos(\omega)))}$$

for $0 < x < \frac{1}{2}P$, we have

$$A'(x) = \frac{P\sin(\omega)(P^2 - 4Px + 2x^2 + 2x^2\cos(\omega))}{4(P - x - x\cos(\omega))^2}$$

Hence, A'(x) = 0 if and only if $x = \frac{2\pm\sqrt{2-2\cos(\omega)}}{2(1+\cos(\omega))}P$. We reject $x = \frac{2+\sqrt{2-2\cos(\omega)}}{2(1+\cos(\omega))}P$ because

$$\frac{2+\sqrt{2-2\cos(\omega)}}{2(1+\cos(\omega))}P = \frac{1+\sqrt{\frac{1-\cos(\omega)}{2}}}{1+\cos(\omega)}P$$
$$= \frac{1+\sin\left(\frac{1}{2}\omega\right)}{1+\cos(\omega)}P$$
$$> \frac{1}{2}P \qquad (0<\omega<\pi).$$

As for $x = \frac{2-\sqrt{2-2\cos(\omega)}}{2(1+\cos(\omega))}P$, it is a maximum because

$$A''(x) = -\frac{P^{3}\sin(\omega)(1-\cos(\omega))}{(P-x-x\cos(\omega))^{3}} < -\frac{P^{3}\sin(\omega)(1-\cos(\omega))}{(P-\frac{1}{2}P-\frac{1}{2}P\cos(\omega))^{3}} = -\frac{8\sin(\omega)}{(1-\cos(\omega))^{2}} < 0 \quad (0 < \omega < \pi).$$

Therefore, if P is fixed, A(x) is increasing for $x \in \left[0, \frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P\right]$ and decreasing for $x \in \mathbb{R}$



(a) An enclosing ω -triant (b) No enclosing ω -triangle exists because $\operatorname{int}(Q) \cap$ gle exists because $\operatorname{int}(Q) \cap$ $\operatorname{int}(\mathcal{C}_P) = \emptyset$. b_- is such that $\operatorname{int}(\mathcal{C}_P) \neq \emptyset$. $x_- = |qb_-|$ is the smallest. b_+ is such that $x_+ = |qb_+|$ is the longest.

Figure 4: In Figure 4(a), an enclosing ω -triangle exists. In Figure 4(b), no enclosing ω -triangle exists.

$$\left[\frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P, \frac{1}{2}P\right[.$$
 The analysis of $P(x)$ is similar.

3 The Solution for a Fixed ω -Wedge

Take a convex polygon Q and an ω -wedge $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$ enclosing Q (refer to Figure 4(a)). The solution for W is based on the following observations.

Observation 1

- 1. An enclosing ω -triangle T with perimeter P can be constructed on W if and only if $int(Q) \cap int(\mathcal{C}_P) = \emptyset$ (refer to Figure 4).
- 2. There exists exactly one T if and only if Q and C_P are tangent.
- 3. Suppose that more than one T exists.
 - (a) We have to compare the ω-triangle △b-qcwith the smallest side x₋ = |qb₋| and the ωtriangle △b₊qc₊ with the longest side x₊ = |qb₊| to find the one with minimum area. These two candidates are such that b₋c₋ and b₊c₊ are tangent to both Q and C_P (refer to Figure 4(a)). Let v₋ (respectively v₊) be the vertex of Q such that b₋c₋ (respectively b₊c₊) is tangent to Q at v₋ (respectively at v₊). We say that v₋ and v₊ are witness vertices. If b₋c₋ (respectively b₊c₊) is flush with an edge e₋ (respectively e₊) of Q, we define v₋ (respectively v₊) as the vertex on e₋ = Q ∩ b₋c₋ (respectively on e₊ = Q ∩ b₊c₊) that is the closest to b₋ (respectively to b₊).

- (b) Any enclosing ω -triangle $\triangle bqc$ with perimeter P strictly between $\triangle b_-qc_-$ and $\triangle b_+qc_+$ is such that bc is tangent to C_P and bc does not touch Q.
- (c) If one of these T's is isosceles, then it has maximum area. One of the T's is isosceles if and only if $x_{-} \leq \frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P \leq x_{+}$. Otherwise, one of $\triangle b_{-}qc_{-}$ and $\triangle b_{+}qc_{+}$ has minimum area and the other one has maximum area by unimodality of A(x).

Hence, if there is no witness vertex, then no ω -triangle T can be constucted on W. If $v_{-} = v_{+}$, then there exists exactly one such T. If $v_{-} \neq v_{+}$, then there exists infinitely many such T's.

For a given ω -wedge W and a given vertex v of Q, it takes O(1) time to decide whether or not $v = v_{-}$ and whether or not $v = v_{+}$ (refer to Subsection 3.1). Thus, for a given ω -wedge W, it takes O(n) time to compute $v_{-}, v_{+}, x_{-}, x_{+}, \Delta b_{-}qc_{-}$ and $\Delta b_{+}qc_{+}$.

3.1 Decide Whether a Vertex is a Witness Vertex

Let $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$ be a fixed ω -wedge enclosing a convex *n*-gon Q and v be a vertex of Q. Denote by Γ the circular arc of Ω such that $q \in \Gamma$. In this subsection, we explain how to decide whether v is a witness vertex of W.

Without loss of generality, Γ is the locus of points q such that $\angle v_i q v_j = \omega$, where v_i and v_j are two vertices of Q (refer to Figure 5). Hence we can take $v_i = (0, 0)$,



Figure 5: A formula for the center (h, k) of C_P .

and $v_j = (2r\sin(\omega), 0)$, where r is the radius of Γ . Let $\theta = \angle v_j v_i q$. In this setting, the radius of C_P is $r_P = \frac{1}{2}P\tan\left(\frac{1}{2}\omega\right) = \frac{P\sin(\omega)}{2(1+\cos(\omega))}$ by Lemma 2-1 and the center (h, k) of C_P is such that

$$h = \frac{1}{2(1 + \cos(\omega))} \times (6r\cos^2(\omega)\sin(\theta)\cos(\theta) + 4r\cos(\omega)\sin(\theta)\cos(\theta) +6r\sin(\omega)\cos(\omega)\cos^2(\theta) - P\cos(\omega)\cos(\theta)$$

$$-2r\sin(\omega)\cos(\omega) + P\sin(\omega)\sin(\theta) +2r\sin(\omega)\cos^{2}(\theta) - 2r\sin(\theta)\cos(\theta) -P\cos(\theta) + 2r\sin(\omega)) ,$$

$$k = -\cot(\theta)h -\frac{P - 4r\cos(\omega)\sin(\theta) - 4r\sin(\omega)\cos(\theta)}{2\sin(\theta)} .$$

These formulas can be obtained by analytic geometry and Lemma 1 in the following way. By geometry and trigonometry, we have $q = (2r\cos(\theta)\sin(\theta + \omega), 2r\sin(\theta)\sin(\theta + \omega))$. Since (h, k) is on the angle bisector of W and $r_P = \frac{P\sin(\omega)}{2(1+\cos(\omega))}$, then the distance between (h, k) and Δ_1 and the distance between (h, k) and Δ_2 are equal to $\frac{P\sin(\omega)}{2(1+\cos(\omega))}$. Hence, we can find the equation of Δ_1 , Δ_2 and the angle bisector of W. From these equations, we deduce the formulas for h and k.

Let e and e' be the two edges adjacent to v. Denote by Δ_e and $\Delta_{e'}$ the lines through e and e' respectively. If $e \cap \operatorname{int}(\mathcal{C}_P) \neq \emptyset$ or $e' \cap \operatorname{int}(\mathcal{C}_P) \neq \emptyset$, then v is not a witness vertex. Moreover, no enclosing ω -triangle can be constructed on W. It follows from Observation 1-1. Suppose that $e \cap \operatorname{int}(\mathcal{C}_P) = \emptyset$ and $e' \cap \operatorname{int}(\mathcal{C}_P) = \emptyset$.

- If $C_P \cap \Delta_e = \emptyset$ and $C_P \cap \Delta_{e'} = \emptyset$, then v is not a witness vertex.
- If $C_P \cap \Delta_e \neq \emptyset$ or $C_P \cap \Delta_{e'} \neq \emptyset$, then v is a witness vertex.

It all follows from Lemma 2-1 and Observation 1-3a. Since we supposed that W is fixed, then r, ω and θ are fixed. So all these tests can be done in O(1) time.

4 Turning Around the ω -Cloud

Let v_{-} and v_{+} be the two witness vertices (possibly $v_{-} = v_{+}$) of an ω -wedge $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$ enclosing Q. Let Γ_j be the circular arc of Ω such that $q \in \Gamma_i$. As q moves on Γ_i , $\triangle b_-qc_-$ and $\triangle b_+qc_+$ move continuously around Q. Moreover, there is a circular arc $\Gamma'_i \subseteq \Gamma_j$ (that can be reduced to a single point) for which the witness vertices remain v_{-} and v_{+} . We say that v_{-} and v_{+} are persistent witness vertices of Γ'_{i} . Among all the enclosing ω -triangles that can be constructed on the enclosing ω -wedges having their apex on Γ'_i , we find the one with the smallest $x_- = |b_-c_-|$ (respectively with the longest $x_+ = |b_+c_+|$). We denote this triangle by $T_{\min} = \triangle q b_{\min} c_{\min}$ (respectively by $T_{\rm max} = \triangle q b_{\rm max} c_{\rm max}$ and we let $x_{\rm min} = |b_{\rm min} c_{\rm min}|$ (respectively $x_{\max} = |b_{\max}c_{\max}|$). By continuity, for all x such that $x_{\min} \leq x \leq x_{\max}$, there exists an enclosing ω wedge with apex on Γ'_j such that an enclosing ω -triangle $\triangle bqc$ can be constructed with x = |bc|. Therefore, on Γ'_j ,

• the minimum enclosing area ω -triangle that can be constructed is either $\triangle q b_{\min} c_{\min}$ or $\triangle q b_{\max} c_{\max}$.

- - If $x_{\min} \leq \frac{2-\sqrt{2}\sqrt{1-\cos(\omega)}}{2(1+\cos(\omega))}P \leq x_{\max}$, then an isosceles ω -triangle with perimeter P has maximum area.
 - Otherwise, one of $\triangle q b_{\min} c_{\min}$ and $\triangle q b_{\max} c_{\max}$ has minimum area and the other one has maximum area.

Given an enclosing ω -wedge $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$ with $q \in \Gamma_j$, it takes O(n) time to compute $v_-, v_+, x_-, x_+, \Delta b_- qc_-$ and $\Delta b_+ qc_+$ (refer to Section 3). Then it takes O(1) time to compute $\Gamma'_j \subseteq \Gamma_j$ such that Γ'_j has persistent witness vertices v_- and v_+ (refer to Subsection 4.1). Then it takes O(1) time to compute $x_{\min}, x_{\max}, T_{\min}$ and T_{\max} (refer to Subsection 4.2).

Once we solved the **FIP** for Γ'_j , we need to compute the next circular arc together with its persistent witness vertices. Denote the next circular arc by Γ''_j Note that since $\Gamma'_j \subseteq \Gamma_j$, then either $\Gamma''_j \subset \Gamma_j$ or $\Gamma''_j \subseteq \Gamma_{j+1}$. Two different events can happen:

Event 1 Γ''_i has no persistent witness vertex

Event 2 or Γ''_j has persistent witness vertices and at least one of v_- and v_+ is different from the persistent witness vertices of Γ'_j .

If Γ''_j has persistent witness vertices, then by continuity, these persistent witness vertices are adjacent to or equal to the persistent witness vertices of Γ'_j . So there are 8 pairs of vertices to test for persistence (recall that at least one of v_- and v_+ is not the same compared to Γ'_j). If none of these 8 pairs is a pair of persistent witness vertices for Γ''_j , then Γ''_j has no persistent witness vertex. Therefore, both **Event 1** and **Event 2** can be detected in O(1) time (refer to Subsection 4.1).

With this technique, we subdivide the circular arcs Γ_j of Ω into subarcs that either have persistent witness vertices or not. Given a subarc Γ'_j , we explained how to solve the **FIP** on Γ'_j in O(n). Given the witness vertices of Γ'_j , we also explained how to solve the **FIP** on every next subarc in O(1). How many of these subarcs are there? In Section 5, we present an example that shows that Γ_j can be subdivided into a linear number of subarcs. However, we do not know whether a constant number or a linear number of circular arcs of Ω can be subdivided into a linear number of subarcs. So the lower bound on the time of computation of the solution to the **FIP** remains an open question.

4.1 Computing Γ'_i

Given two vertices v_{-} and v_{+} and a circular arc Γ_{j} of Ω , we explain how to find $\Gamma'_{j} \subseteq \Gamma_{j}$ such that Γ'_{j} has v_{-} and v_{+} as persistent witness vertices. From the discussion of Subsection 3.1, Lemma 2-1 and Observation 1, we need to find the values of θ such that Δ_{e} is tangent to \mathcal{C}_{P} and the values of θ such that $\Delta_{e'}$ is tangent to C_P . We explain how to find the values of θ such that Δ_e is tangent to C_P (the values of θ such that $\Delta_{e'}$ is tangent to C_P can be found in a similar way). Take $C_P: (x-h)^2 + (y-k)^2 = r_P^2$ and $\Delta_e: y = \mu x + \lambda$. From analytic geometry, Δ_e is tangent to C_P if and only if

$$k \pm \frac{r_P}{\sqrt{\mu^2 + 1}} = \mu \left(h - \frac{\mu r_P}{\sqrt{\mu^2 + 1}} \right) + \lambda \quad . \tag{2}$$

Since μ , λ , P, ω and r are constant, (2) is an equation of degree 2 in $\sin(\theta)$ and $\cos(\theta)$. Therefore, it can be transformed into an equation of degree 4 in $\sin(\theta)$. If (2) has no solution in θ or if the solutions are not sound with respect to Γ_j , then there is no $\Gamma'_j \subseteq \Gamma_j$ that has v_- and v_+ as persistent witness vertices. Thus it can be solved exactly in O(1) time.

4.2 Compute b_{\min} and b_{\max}

As we did in Section 3.1, let $W = \mathcal{W}(\omega, q, \Delta_1, \Delta_2)$ be a fixed ω -wedge enclosing a convex *n*-gon Q. Let Γ be a circular arc such that v is a persistent witness vertex of Γ . Without loss of generality, Γ is the locus of points qsuch that $\angle v_i q v_j = \omega$, where v_i and v_j are two vertices of Q (refer to Figure 5). Hence we can take $v_i = (0, 0)$, and $v_j = (2r \sin(\omega), 0)$, where r is the radius of Γ . Let $\theta = \angle v_j v_i q$.

Let $b = (\alpha, \alpha \tan(\theta)) \in \Delta_1$ be a point such that $\triangle bqc$ has the prescribed perimeter, where c is the intersection point of the line through bv and the line through qv_j . Therefore, the line $\Delta : y = \mu x + \lambda$ through bv satisfies (2) from the discussion of Subsection 4.1. For a fixed θ , it is an equation in α . Hence, in order to find b_{\min} or b_{\max} , we need to optimize |qb| subject to (2). It leads to a polynomial equation in $\sin(\theta)$ of high degree. This can be done with numerical methods. For a given fixed error tolerance, it takes O(1) time to compute b_{\min} or b_{\max} .

5 An Interspersing Lemma

If we translate the interspering lemmas of [1, 2, 3, 6] in terms of the **FIP**, we get the following statement: "as q turns clockwise around Ω , v_{-} and v_{+} turn clockwise around Q." This statement implies that the time of computation of the solution to the **FIP** is O(n) (when the input is a convex n-gon) since we only need to go around once. Unfortunately, this statement is false in the current setting. In this section, we construct an example where q turns clockwise around Ω and v_{+} turns counter-clockwise around Q. Because of this example, the time of computation of our algorithm is $O(n^2)$. Indeed, this example suggests that all circular arcs Γ_j of Ω could be subdivided into a linear number of subarcs (refer to Section 4). Consider the example of Figure 6 where $\omega = \frac{1}{2}\pi$. Four vertices of Q appear, namely $v_i, v_k, v_{i'}$ and $v_{i'+1}$. The circular arc Γ_j of Ω is built over v_i and v_k , and we consider an enclosing $\frac{1}{2}\pi$ -wedge $W = \mathcal{W}(\frac{1}{2}\pi, q, \Delta_1, \Delta_2)$ where $q \in \Gamma_j$. $T_+ = \Delta b_+ qc_+$ is such that b_+c_+ is flush



Figure 6: As q turns clockwise around Ω , v_+ turns counter-clockwise around Q.

with the edge $e_{i'} = v_{i'}v_{i'+1}$ of Q and b_+c_+ is tangent to C_P at $t \in e_{i'}$. Therefore, the witness vertex v_+ of T_+ is $v_+ = v_{i'+1}$.

Let $W' = \mathcal{W}(\frac{1}{2}\pi, q', \Delta'_1, \Delta'_2)$ be an enclosing $\frac{1}{2}\pi$ wedge obtained by a clockwise rotation of q around Γ_j and such that $v_{i'} \notin \Delta'_2$. $T'_+ = \Delta b'_+ q' c'_+$ is such that $b'_+ c'_+$ touches Q at $v_{i'}$ and $b'_+ c'_+$ is tangent to \mathcal{C}'_P at $t' \notin Q$. Therefore, $v'_+ = v_{i'}$. Hence, this is an example where q turns clockwise around Ω and v_+ turns counterclockwise around Q.

Using the same strategy, we can make v_+ turn counter-clockwise around Q and visit m vertices for any $m \geq 1$. Let $q_0 = q, q_1, ..., q_{m-1} = q' \in \Gamma_j$ be a sequence of m different points from q to q'. For each q_l ($0 \leq l \leq m-1$), consider the wedge $W_l =$ $\mathcal{W}(\frac{1}{2}\pi, q_l, \Delta_{l,1}, \Delta_{l,2})$ and $T_{l,+} = \Delta b_{l,+}q_ic_{l,+}$. Put a vertex $v_{i'-l}$ on $b_{l,+}c_{l,+}$ such that $b_{l,+}c_{l,+}$ is flush with the edge $e_{i'-l} = v_{i'-l}v_{i'-l+1}$ and $v_{i'-l}$ is strictly between $v_{i'-l+1}$ and $c_{l,+}$. This way, $v_{l,+} = v_{i'-l+1}$ so as q turns clockwise around Ω , v_+ turns counter-clockwise around Q and visits m vertices for any $m \geq 1$.

This proves that the canonical interspersing lemma for the **FIP** does not stand. However, it does not prove that the lower bound on the the time of computation of the solution to the **FIP** is $\Omega(n^2)$. The construction we presented works for Γ_j , but we do not know if it is possible to do such a construction on all the circular arcs of Ω simultaneously. This question remains open.

6 Conclusion

We explained in detail how to find an ω -triangle of minimum and maximum area with fixed perimeter. Our solution takes $O(n^2)$. If one fixes the area rather than the perimeter, a similar solution exists by switching the word "perimeter" with "area", "minimum" with "maximum", and "perimeter circle" with "area hyperbole".

Two main questions remain open about the **FIP**. Is $\Omega(n^2)$ the lower bound on the time of computation of the solution to the **FIP**? Is it possible to simplify the polynomial equations involved in the computation of b_{\min} and b_{\max} ? As for more general open questions related to the **FIP**,

- 1. What is the time of computation of the solution to the **FIP** when there is no angle constraint?
- 2. What is the time of computation of the solution to the **FIP** when we consider shapes with curved boundary?
- 3. What is the solution in three dimensions?

References

- B.K. Bhattacharya and A. Mukhopadhyay. On the minimum perimeter triangle enclosing a convex polygon. In *JCDCG*, pages 84–96, 2002.
- [2] P. Bose and J.-L. De Carufel. Minimum enclosing area triangle with a fixed angle. In *CCCG*, pages 171–174, 2010.
- [3] P. Bose, M. Mora, C. Seara, and S. Sethia. On computing enclosing isosceles triangles and related problems. *Int. J. Comput. Geometry Appl.*, 21(1):303– 318, 2011.
- [4] E.A. Melissaratos and D.L. Souvaine. Shortest paths help solve geometric optimization problems in planar regions. SIAM J. Comput., 21(4):601–638, 1992.
- [5] J.S. Mitchell and V. Polishchuk. Minimumperimeter enclosures. *Inf. Process. Lett.*, 107(3-4):120–124, 2008.
- [6] J. O'Rourke, A. Aggarwal, S.R. Maddila, and M. Baldwin. An optimal algorithm for finding minimal enclosing triangles. J. Algorithms, 7(2):258–269, 1986.
- [7] G. Polya. Mathematics and Plausible Reasoning Vol. I. Induction and Analogy in Mathematics. Princeton University Press, Princeton, New Jersey, 1954.
- [8] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.
- [9] Y. Zhou and S. Suri. Algorithms for a minimum volume enclosing simplex in three dimensions. *SIAM* J. Comput., 31(5):1339–1357, 2002.

Robust approximate assembly partitioning

Elisha Sacks *

Victor Milenkovic[†]

Yujun Wu[‡]

Abstract

We present a robust approximate assembly partitioning algorithm for polyhedral parts. We achieve robustness by applying our controlled linear perturbation strategy to Minkowski sums of polyhedra and to arrangements of great circle arcs. Our algorithm is far faster than a prior robust algorithm based on exact computational geometry. Its error is small even on degenerate input.

1 Introduction

We present a robust approximate assembly partitioning algorithm. Given a set of polyhedral parts, the task is to find a direction in which a subset of the parts can translate unboundedly without touching the other parts. Assembly partitioning is a key step in the larger task, called assembly planning, of devising a sequence of coordinated part translations and rotations that builds an assembly from a set of parts. An efficient assembly partitioning algorithm is crucial because assembly planning is computationally intractable. Halperin [7] presents a real RAM algorithm for generic input. Actual input is typically degenerate because useful parts usually have symmetric features. The *robustness* problem is how to implement the algorithm accurately, efficiently, and for any input.

Fogel [4] uses exact computational geometry [9] to implement Halperin's algorithm. Error is avoided by exactly evaluating polynomials in the input parameters, called predicates, whose signs determine the output. Although most predicates can be evaluated quickly via floating point filtering [1], near-zero predicates require expensive rational arithmetic. Typical assembly partitioning tasks have many such predicates, which makes Fogel's approach slow. Degenerate (zero value) predicates require explicit handling, which complicates the algorithm. Exact computation also increases bit complexity, hence memory use, which is the computational bottleneck for large inputs.

We [8] advocate an alternate robustness strategy, called controlled linear perturbation (CLP), based on approximate computation with floating point arithmetic. CLP uses differential calculus to compute a



Figure 1: Assembly (a) and directional blocking graphs for x (b) and y (c).

small input perturbation that makes the output accurate. The running time is insensitive to near-zero predicates, degeneracy handling is avoided, and the bit complexity is low. We use CLP to implement the assembly partitioning algorithm (Sec. 2). The computational geometry steps are Minkowski sums, using our prior algorithm [8], and arrangements of great circle arcs, using a plane sweep algorithm (Secs. 3–4). We demonstrate that our algorithm is far faster than its exact counterpart (Sec. 5). Its error is small even on degenerate input. We conclude with a discussion of the two robustness strategies (Sec. 6).

2 Assembly partitioning algorithm

The input to the algorithm is n disjoint polyhedral parts, $A = \{P_1, \ldots, P_n\}$. Let $P_i + v = \{p + v | p \in P_i\}$ denote the translation of P_i by the vector v. A motion direction is represented by a unit vector. The direction d is free for $\langle P_i, P_j \rangle$ if $(P_i + kd) \cap P_j = \emptyset$ for every $k \ge 0$; otherwise d is blocked for $\langle P_i, P_j \rangle$. Part P_i can translate unboundedly along a free d without hitting P_j , but not along a blocked d. We seek a proper subset, $S \subset A$, and a direction, d, that is free for every $\langle P_i, P_j \rangle$ with $P_i \in S$ and $P_j \notin S$. In Fig. 1a, \hat{x} is free for $\langle P_2, P_1 \rangle$, and blocked for $\langle P_1, P_2 \rangle$ and $\langle P_2, P_3 \rangle$. One solution is $S = \{P_3\}$ and $d = \hat{x}$; another is $S = \{P_1, P_2\}$ and $d = \hat{y}$.

The algorithm for a fixed d is combinatorial. Form the directional blocking graph with a node for each part and with a link from P_i to P_j if d is blocked for $\langle P_i, P_j \rangle$. If

^{*}Department of Computer Science, Purdue University, eps@cs.purdue.edu

[†]Department of Computer Science, University of Miami

[‡]Department of Computer Science, Purdue University

the graph is strongly connected, there is no solution because every $S \subset A$ has a link from some $P_i \in S$ to some $P_j \notin S$. Otherwise, any component without outgoing links is a solution. In our example, the \hat{x} graph components are $\{\{P_1\}, \{P_2\}, \{P_3\}\}$ and the \hat{y} graph components are $\{\{P_1, P_2\}, \{P_3\}\}$ (Fig. 1).

The assembly partitioning algorithm computes a subdivision of the unit sphere such that all the directions in each face have the same graph. It traverses the faces of the subdivision, analyzes their graphs, and returns the first solution or reports failure. The subdivision is computed in two steps. 1) Partition the unit sphere into free and blocked faces for each $\langle P_i, P_j \rangle$. The graph has a link from P_i to P_j for d in the blocked faces of $\langle P_i, P_j \rangle$. 2) Compute the overlay of the partitions.

We illustrate the algorithm on an assembly comprised of ring P_1 , ring P_2 , and cone P_3 with axis z = (0, 0, 1)(Fig. 2). Faces e-g of the overlay are in the northern hemisphere, h straddles the equator, and the southern hemisphere is symmetric. Face e has solutions with S = $\{P_1\}$ (Fig. 2c). Face f has one more link, from P_2 to P_3 , and also has solutions with $S = \{P_1\}$. Face g has no solutions (Fig. 2d). Face h has one less link, from P_2 to P_1 , and no solutions.

Figure 3 illustrates step 1 of the algorithm in 2D. A direction, d, is blocked for $\langle P_i, P_j \rangle$ if the ray kd intersects the Minkowski sum

$$M_{ij} = P_j \oplus -P_i = \{a - b | a \in P_j, b \in P_i\},\$$

which comprises the vectors, v, such that $P_i + v$ intersects P_j . Let Q_{ij} denote the projection of M_{ij} onto the unit sphere: a vector, v, projects to $\hat{v} = v/||v||$. The projection is defined because the parts are disjoint, so $(0,0,0) \notin M_{ij}$. The connected components of Q_{ij} are the blocked faces of $\langle P_i, P_j \rangle$. We compute them for i < j and handle $Q_{ji} = -Q_{ij}$ by symmetry.

Figure 4 illustrates projection. The boundary of Q is a subset of the projected silhouette edges of M. Let e = ab denote an edge of M with tail a and head b; its twin is the edge with tail b and head a. Let e have tangent u, and faces to the left and right with outward normals m and n. If $a \cdot m > 0$ and $a \cdot n < 0$, e is a silhouette edge. Project it to the arc, \hat{e} , with tail \hat{a} and head \hat{b} (Fig. 4a,c) on the great circle defined by the plane with normal $\widehat{a \times b}$. Label the silhouette arcs positive and label their twins negative. Compute the induced subdivision of the unit sphere. A face is in Q if its boundary contains a positive edge or if the ray kd intersects M for any point, d, in its interior. The blocked faces of Q are bounded by the edges that bound Q, but whose twins do not.



Figure 2: Ring assembly (a), overlay (b), and directional blocking graphs for faces e (c) and g (d).



Figure 3: Partition: (a) parts; (b) Minkowski sum; (c) projection.



Figure 4: Projection: (a) M; (b) Q; (c) projected edges with silhouette edges drawn thickly.



Figure 5: Arrangement algorithm: (a) input arcs; (b) split points; (c) faces.

3 Arrangement algorithm

We compute arrangements of great circle arcs for projection and for overlay. The algorithm is a plane sweep, which splits the arcs at their intersection points to obtain the vertices and edges of the arrangement, followed by a traversal of the vertex/edge graph, which derives the faces. Fig. 5 illustrates on the Fig. 4 example.

Preprocessing Split the input arcs (Fig. 5a) at z turning points (a in Fig. 5b). An arc, e = ab, with normal n has a turning point if $u_z v_z < 0$ with $u = n \times a$ and $v = n \times b$ the tangents at a and b. If $u_z > 0$, e has a maximum at \hat{p} with $p = \operatorname{sign}(n_z)(-n_x, -n_y, 1/n_z - n_z)$; if $u_z < 0$, e has a minimum at $-\hat{p}$. Splitting the arcs yields z-monotone edges. Split the edges at intersection points, q, with the great circle with normal (0, 1, 0) such that $q_x < 0$ (g and h in Fig. 5b).

Place the incident edges of each vertex, a, in clockwise order around the outward normal. An edge, e = ab, is forward or backward if $a_z < b_z$ or $b_z < a_z$. Forward edges precede backward edges. An edge with normal m precedes one with normal n if $a \cdot (u \times v) < 0$ with $u = \widehat{m \times a}$ and $v = \widehat{n \times a}$ the tangents at a. This predicate is identically zero if a is a z turning point because u = -v. Instead, the positive edge precedes the negative edge for $a_z n_z > 0$ and vice versa for $a_z n_z < 0$.

Sweep Sweep a plane along the z axis from z = -1 to z = 1. The plane intersects the unit sphere in a circle. The sweep list consists of the forward edges that intersect this circle in counterclockwise order. The events are the input vertices and the intersection vertices (*b*-*f* in Fig. 5b). The z order is calculated by comparing vertex z coordinates, except that a always precedes or follows b when a is a z minimum or maximum of e = ab.

An input vertex is handled by removing the twins of its backward edges from the sweep list, recording the edge that follows it in the sweep list, inserting its forward edges, and checking if any newly adjacent edges intersect. Two edges cannot intersect if they come from the same input arc. Otherwise, edges e = ab with normal m and f = cd with normal n intersect at an intersection point, $p = \pm \widehat{m \times n}$, of their great circles if their tangents at $p, \ \widehat{m \times p}$ and $\widehat{n \times p}$, have positive zcomponents and $\max(a_z, c_z) < p_z < \min(b_z, d_z)$. The intersection vertex is handled by splitting e into ap and pb, splitting f into cp and pd, placing the p edges in the order (pd, pb, pc, pa), replacing e by pd and f by pb in the sweep list, and checking the newly adjacent edges.

We represent the sweep order as a linear order on $[-\pi,\pi]$. The transitions between $-\pi$ and π occur at vertices because of the preprocessing. If a transition occurs at a, e = ab is inserted at the start or the end of the sweep list when $b_y < 0$ or $b_y > 0$. Otherwise, e is inserted by repeatedly comparing it to an edge, f = cd, in the list. If a = c, the sweep order is the counterclockwise order around a. Otherwise, compute the sweep order of a and the intersection point, p, of f with the sweep plane $z = a_z$. Edge e precedes f if $a_y < 0$ and $p_y > 0$ or if $a_y p_y > 0$ and $a_x p_y - a_y p_x > 0$.

Graph traversal Mark the edges as untraversed. Visit each vertex in sweep z order and trace an edge loop starting at each of its untraversed edges. While the current edge, e = ab, is untraversed, mark it as traversed and replace it by the successor of its twin among the edges incident on b. For the first vertex or for a vertex with an edge that was traversed before it was visited, each edge loop defines a face. The six faces in Fig. 5c are

generated in this manner in numerical order. Otherwise, the first loop is added to the enclosing face and the other loops define faces. The enclosing face is bounded by the following edge of the vertex, if defined, or by the first edge of the previous vertex.

4 Robustness

A direct floating point implementation of the arrangement algorithm is not robust. Even tiny computation errors can cause a predicate to be assigned the wrong sign, which can create a combinatorial error in the algorithm output. For this to occur, the predicate must be *unsafe*, meaning that its value is on the order of the computation error. The main cause of unsafe predicates is degeneracy. A degenerate input manifests itself as a predicate that evaluates to zero, so approximate computation assigns it an unsafe value.

We prevent unsafe predicates with our controlled linear perturbation (CLP) algorithm [8]. CLP assigns signs, s_i , to a sequence of predicates, $f_i(x)$, with input values x = a. It picks a random unit vector, v, and computes a $\delta \ge 0$ such that $s_i f_i(p) > \epsilon$ with $p = a + \delta v$ and with ϵ a safety threshold that depends on f and on a. If $|f_i(p)| > \epsilon$ with the current δ , $s_i = \text{sign}(f_i(p))$; otherwise, $s_i = \text{sign}(w)$ with $w = \nabla f \cdot v$ and with ∇f the gradient, and δ is increased by $(s\epsilon - f(p))/w$ to the minimum value that makes f_i safe based on its linear Taylor series.

We employ the backward error metric: the error in a computation is the minimum distance from the input to a perturbed input for which the output is correct. For a CLP algorithm, the input is a, a perturbed input is p, and the error is at most $||p - a|| = \delta$. We assume that the signs, s_i , are correct at p, which holds when the safety thresholds exceed the predicate rounding error. The rounding error in a single arithmetic operation is bounded by the rounding unit of $\mu \approx 10^{-16}$. The error in a sequence of n operations is exponential in n in the worst case, but is essentially constant in practice. We employ a safety threshold of $\epsilon = 100\mu$, which is conservative by numerical analysis standards given that n < 50 in our algorithm.

The sign assignment algorithm performs poorly on singular predicates ($\nabla f = 0$). Singularity is much rarer than degeneracy because both f and ∇f must be zero. Yet a single singular predicate can invalidate the arrangement computation by increasing δ unacceptably. The sweep has singularities when vertices coincide with z turning points. We prevent this by sweeping along a random axis. This strategy suffices for the arrangement algorithm. We discuss a general strategy in Sec. 6.



Figure 6: Star puzzle: (a) one part, (b) two parts, (c) all six parts.

5 Performance

We tested our algorithm on Fogel's star puzzle example (Fig. 6). The assembly has six parts that are rotational images of each other. Each part has 14 boundary triangles. The Minkowski sums and the arrangements are degenerate because the parts are symmetric, the pairs consist of isometric parts, and the assembly contains many isometric pairs. Nevertheless, the backward error is only $\delta = 10^{-12}$. The running time, on one core of an Intel Core 2 Duo with 4 GB RAM, is 0.024 seconds with 82% for Minkowski sums, 8% for projection, and 7% for overlay. This is about 100 times faster than Fogel's best time of 5.2 seconds, since our CPU is about 50% faster.

We also tested our algorithm on two engineering examples. The first is the ring assembly (Fig. 2). Each ring has 2068 boundary triangles and the cone has 160. The running time is 3.2 seconds with 94% for the three Minkowski sums. The error is $\delta = 4 \times 10^{-9}$. The second



Figure 7: Hinge assembly.

is a hinge assembly comprised of a bolt and two plates (Fig. 7). The bolt has 160 boundary triangles, the inner plate has 596, and the outer plate has 572. The running time is 1.1 seconds with 85% for the three Minkowski sums. The error, $\delta = 10^{-7}$, is larger than before because the parts have many parallel boundary triangles, each of which causes multiple degeneracies.

6 Discussion

The performance of our assembly partitioning algorithm supports our thesis that the CLP robustness strategy is accurate, is far faster than exact computation, and avoids degeneracy handling. One reason for the speedup is that floating point arithmetic is faster than rational arithmetic. A second reason is that we compute Minkowski sums, which are the dominant cost in assembly planning, via a convolution algorithm that is output sensitive in practice. Fogel decomposes the polyhedra into convex pieces by hand, computes the piece sums, and forms their union. Since the pieces have many noninput features, the complexity far exceeds the output size in typical examples. The same is true of a later algorithm that automates the decomposition [6]. We attribute the lack of a robust exact convolution algorithm to the daunting degenerate cases, including collinear faces, identical faces, and edges on faces.

Clearance Fogel's algorithm has the theoretical advantage that it can find solutions where parts have zero clearance, meaning their boundaries intersect and their interiors are disjoint, by examining the degenerate faces of the overlay. We cannot compute faces whose diameter is less than δ . Since the maximal part clearance for directions on a face is proportional to its diameter, we are limited to solutions whose clearance exceeds δ .

We see no practical significance to this limitation. Parts are subject to manufacturing variation and assembly mechanisms are subject to motion variation. An assembly plan must handle all parts and mechanisms of a specified accuracy. A plan with a clearance of 10^{-6} is unsafe for any conceivable accuracy, whereas δ is always far smaller. The standard planning strategy is to replace each ideal part by an expanded part that bounds its shape variation. The simplest and most common replacement is the Minkowski sum of the ideal part and an *s*-sphere centered at the origin. The existence of an exact solution for s = k implies the existence of an approximate solution for $s = k + \delta$. The solutions are equivalent in practice because δ is negligible with respect to k.

Designers sometimes consider ideal parts before modeling part variation. Assembly plans with zero clearance are then of interest. For example, the star puzzle is more elegant when the parts fit together perfectly. We can approximate zero clearance solutions by dilating the parts by r_1 (subtracting a sphere of radius r_1) until an approximate solution is found, expanding them by r_2 until they overlap, and performing binary search on $[-r_1, r_2]$ for the smallest parts that yield an approximate solution. We implemented this procedure for the star puzzle, but using scaling instead of dilation and expansion. Scaling by 99% yields a solution, scaling by 101% makes the parts overlap, and 7 iterations yield an approximate solution with $\delta = 10^{-12}$ in 0.17 seconds, versus 5.2 seconds for Fogel's fastest run.

Correctness CLP algorithms can fail due to extreme rounding error, whereas exact algorithms cannot. On the other hand, exact algorithms effectively halt when they run out of memory, which already occurs on modest size Minkowski sums. CLP is correct assuming the same empirical bounds on rounding error that underlie every numerical library in the scientific computing community. We have never observed a CLP failure despite extensive consistency checking of every Minkowski sum and spherical arrangement that we compute. We aim to replace this empirical evidence with a rigorous, yet practical error analysis. One option is to adjust the floating point precision to match the worst case rounding error, using an arbitrary precision floating point library, such as MPFR [5]. Another option is to derive probabilistic error bounds by comparing the predicate signs due to several perturbations.

Algorithm design We conclude with a comparison of algorithm design using CLP versus exact computation. An exact algorithm has to address degeneracy, whereas a CLP algorithm does not. Explicit degeneracy handling appears impractical in most 3D algorithms. The alternative to explicit handling is symbolic perturbation [2, 3], which yields predicate signs that are correct for an arbitrarily small input perturbation. Symbolic perturbation further increases the computational complexity of exact computation. Neither CLP nor exact computation with symbolic perturbation can solve degenerate problems. We have illustrated that degenerate assembly partitioning problems can be solved approximately with CLP, but the process is not automated.

A CLP algorithm has to address singularity, whereas an exact algorithm with explicit degeneracy handling does not, since singularity is a special case of degeneracy. An exact algorithm with symbolic perturbation has to address singularity because it computes the first non-vanishing derivative of degenerate predicates. The computational complexity rises sharply with the degree of singularity.

We classify singularities as artifacts, coincidences, and special cases. Artifacts occur in algorithms that impose extra structure on the input, such as the z order in our sweep algorithm. They can be avoided by randomization. Coincidences occur when combinatorially distinct elements are numerically equal, for example $a \cdot (b \times c)$ with a = b = c. We replace $u = b \times c$ by \hat{u} . A generalization of this strategy handles any rank deficient determinant predicate. Special cases occur when the parameters of a predicate are related. We exploit the parameter relationship to derive an equivalent regular predicate, such as the clockwise edge order at a z turn (Sec. 3). We have employed this strategy in several complicated 3D algorithms and aim to automate it.

Acknowledgment

Milenkovic supported by NSF grant CCF-0904707. Sacks supported by NSF grant CCF-0904832.

References

- [1] Exact computational geometry. http://cs.nyu.edu/exact.
- [2] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Transactions on Graphics, 9(1):66–104, 1990.
- [3] I. Emiris, J. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19(1–2):219–242, 1997.
- [4] E. Fogel and D. Halperin. Polyhedral assembly partitioning with infinite translations or the importance of being exact. In *Eighth International Workshop on* the Algorithmic Foundations of Robotics, pages 417– 432, 2009.
- [5] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33, 2007.

- [6] P. Hachenberger. Exact minkowski sums of polyhdra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica*, 55:329–345, 2009.
- [7] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.
- [8] V. Milenkovic, E. Sacks, and M.-H. Kyung. Robust minkowski sums of polyhedra via controlled linear perturbation. In *Solid Modeling*, pages 23– 30. Springer, 2010.
- [9] C. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, chapter 41, pages 927–952. CRC Press, Boca Raton, FL, second edition, 2004.

Approximation Algorithms for a Triangle Enclosure Problem

Karim Douïeb*

Matthew Eastman^{*}

Anil Maheshwari*

Michiel Smid*

Abstract

Given a set S of n points in the plane, we want to find a triangle, with vertices in S, such that the number of points of S enclosed by it is maximum. A solution can be found by considering all $\binom{n}{3}$ triples of points in S. We show that, by considering only triangles with at least 1, 2, or 3 vertices on the convex hull of S, we obtain various approximation algorithms that run in $o(n^3)$ time.

1 Introduction

Let S be a set of n points in the plane. A triangle $\triangle pqr$, with vertices $p, q, r \in S$, is defined to be *optimal* if the number of points of S enclosed by it is maximum. Eppstein *et al.* [1] have shown that this optimal triangle can be computed in $O(n^3)$ time: They present an algorithm that preprocesses the set S in $O(n^2)$ time so that, for any triple (p, q, r) of points in S, the number of points enclosed by $\triangle pqr$ can be computed in O(1) time. By considering all $\binom{n}{3}$ triples, we find an optimal triangle in $O(n^3)$ time.

Since it is not known if an optimal triangle can be computed in $o(n^3)$ time, we consider the problem of approximating it. That is, we will present several subcubic algorithms that compute triangles with vertices in S that enclose at least 1/c times as many points as an optimal triangle with vertices in S, for some approximation ratio c.

Our main approach is based on the simple fact that if a triangle \triangle can be covered by *c* triangles, then one of them is a *c*-approximation of \triangle .

We show that, by considering only triangles that contain at least 1, 2, or 3 vertices on the convex hull of S, we obtain approximation algorithms, for various values of c, that run in $o(n^3)$ time. Let h denote the number of vertices on the convex hull of S. A summary of our results is given in Table 1.

2 Preliminaries

We will assume that no three points in S are collinear and that no two points have the same y-coordinate.

vertices on the	approximation	runtime
$convex \ hull$	ratio	
≥ 1	2	$O(n^2)$
≥ 2	3	$O(nh^2\log n)$
≥ 2	4	$O(n\log^2 n)$
3	4	$O(nh^2\log h)$
3	8	$O(n\log^2 h)$
3	$3\log h$	$O(n\log h)$

Table 1: Summary of results.

The number of points *enclosed* by a triangle $\triangle pqr$ is the number of points contained in the interior of $\triangle pqr$. We say that $\triangle pqr$, with $p, q, r \in S$, is *optimal* if the number of points of S enclosed by it is maximum.

A triangle \triangle is a *c*-approximation of a triangle $\triangle pqr$ if \triangle encloses at least 1/c times as many points as $\triangle pqr$.

Observation 1 If a triangle $\triangle pqr$ can be covered by a set of c triangles then at least one of these triangles is a c-approximation of $\triangle pqr$.

In order to show that an algorithm gives a c-approximation of a triangle $\triangle pqr$ it is enough to show that the algorithm counts the number of points enclosed by each of the c triangles that cover $\triangle pqr$.

Let l(p,q) denote the directed line through points pand q, and let \overline{pq} denote the line segment between p and q. Define the *wedge* of a vertex p in a triangle $\triangle pqr$ as the area bounded by the lines l(q,p) and l(r,p) opposite the interior angle $\angle rpq$.

Lemma 1 The three wedges of an optimal triangle with vertices in S cannot contain any points of S.

Proof. Let $\triangle pqr$ be an optimal triangle with vertices in *S*. Assume that the wedge of *p* contains a point *p'* as in Figure 1. Then the triangle $\triangle p'qr$ encloses more points than $\triangle pqr$, as it encloses all of the points enclosed by $\triangle pqr$ in addition to the point *p*, giving a contradiction.

We refer to the three wedges of an optimal triangle as the *empty regions* of the optimal triangle.

3 Counting points in triangles with two fixed vertices on the convex hull

In order to approximate an optimal triangle in $o(n^3)$ time we need to be able to count the number of points

^{*}School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada. This work was supported by the Natural Sciences and Engineering Research Council of Canada. Emails: kdouieb@ulb.ac.be, {meastma2,anil,michiel}@scs.carleton.ca.



Figure 1: The wedge of p cannot contain any points. The shaded regions denote the empty regions of $\triangle pqr$.

in a set of triangles in $o(n^3)$ time. Fixing two vertices of every triangle on the convex hull of S allows us to count the number of points enclosed by these triangles in $O(n \log n)$ time, or $O(n \log h)$ time if we only consider triangles with the third vertex on the convex hull.

Lemma 2 Given two points t_i and t_j on the convex hull of S we can count the number of points enclosed by every triangle $\Delta t_i t_j s$, $s \in S$, in $O(n \log n)$ time.

Proof. Without loss of generality assume that t_i is below t_j . Let S_L be the set of points of S lying to the left of $l(t_i, t_j)$ and let S_R be the set of points of S lying to the right of $l(t_i, t_j)$.

The following algorithm counts the number of points enclosed by every triangle $\Delta t_i t_j s$, $s \in S_L$. Counting the number of points enclosed by every triangle $\Delta t_i t_j s$, $s \in S_R$, is symmetric.

For each point $s \in S_L$, let s' be the intersection between the horizontal line through s and $l(t_i, t_j)$. Let $S_L^$ be the set of points in S_L lying below the horizontal line through t_i and let S_L^+ be the set of points lying above the horizontal line through t_i .

Let T be an initially empty balanced binary search tree such that every node in T stores the size of its subtree. Rotate a line anchored at t_i clockwise over the set S_L^- . When this line intersects a point $s \in S_L^-$ insert s into T using its y-coordinate as the key. The number of points enclosed by $\Delta t_i ss'$ is the number of successors of s in T immediately after inserting s.

To see why this is true let u be a successor of s in T found immediately after inserting s into T. Since u was inserted before, s the angle $\angle ut_i s'$ is less than $\angle st_i s'$. Since u is a successor of s in T, u is higher than s. Therefore u is enclosed by $\triangle t_i ss'$ (see Figure 2).

The number of points enclosed by every triangle $\Delta t_i ss', s \in S_L^+$, is found using the same technique, except that the line is rotated counter-clockwise over S_L^+ and the number of points in each $\Delta t_i ss', s \in S_L^+$, is the number of predecessors of s in T immediately after inserting s.

Counting the number of points enclosed by every triangle $\Delta t_i ss', s \in S_L$, is symmetric.

For each point $s \in S_L$ let $a_{i,s}$ be the number of points enclosed by $\Delta t_i ss'$ and let $a_{j,s}$ be the number of points enclosed by $\Delta t_j ss'$. Then the number of points enclosed



Figure 2: Point u is enclosed by $\Delta t_i ss'$.

by $\Delta t_i t_j s$ is either (1) $-a_{i,s} + a_{j,s}$ if s is below t_i , (2) $a_{i,s} - a_{j,s}$ if s is above t_j , or (3) $a_{i,s} + a_{j,s}$ otherwise. These cases are shown in Figure 3.



Figure 3: The three cases encountered when calculating the number of points enclosed by $\Delta t_i t_j s$.

It takes $O(n \log n)$ time to sort the points by angle about t_i and t_j . Inserting each point into the binary search tree takes $O(\log n)$ time. Since the binary search tree keeps track of the size of each subtree we can calculate the number of predecessors or successors of a point in the tree in $O(\log n)$ time. The total runtime is $O(n \log n)$.

If we fix two vertices on the convex hull of S we can count the number of points enclosed by every triangle containing these two vertices, with the third vertex on the convex hull, without sorting the entire set S. This lets us count the number of points enclosed by every such triangle in $O(n \log h)$ time.

Lemma 3 Given two points t_i and t_j on the convex hull of S we can count the number of points enclosed by every triangle $\Delta t_i t_j t_k$ where t_k , $1 \le k \le h$, is a point on the convex hull of S, in $O(n \log h)$ time.

Proof. Without loss of generality assume that t_i is below t_j . Let S_L be the set of points of S lying to the left of $l(t_i, t_j)$ and let S_R be the set of points of S lying to the right of $l(t_i, t_j)$.

The following algorithm counts the number of points enclosed by every triangle $\Delta t_i t_j t_k$, where $t_k \in S_L$ is a point on the convex hull between t_i and t_j . Counting the number of points enclosed by every triangle $\Delta t_i t_j t_k$, where $t_k \in S_R$ is a point on the convex hull, is symmetric. The number of points enclosed by $\Delta t_i t_j t_k$, with $t_k \in S_L$, is found by subtracting the number of points in S_L lying to the left of $l(t_i, t_k)$, or to the right of $l(t_j, t_k)$, from the number of points in S_L .

A point $s \in S_L$ lies to the left of $l(t_i, t_k)$ if the line $l(t_i, s)$ intersects the convex hull between t_i and t_k . Similarly, s lies to the right of $l(t_j, t_k)$ if $l(t_j, s)$ intersects the convex hull between t_k and t_j (see Figure 4).



Figure 4: Lines through t_i and the points lying to the left of $l(t_i, t_k)$ intersect the convex hull between t_i and t_k . Lines through t_j and points lying to the right of $l(t_i, t_k)$ intersect the convex hull between t_k and t_j .

Let $a_{i,k}$ be the number of lines $l(t_i, s)$, $s \in S_L$, that intersect the edge $\overline{t_k t_{k+1}}$ of the convex hull and let $a_{j,k}$ be the number of lines $l(t_j, s)$, $s \in S_L$, that intersect the edge $\overline{t_k t_{k+1}}$ of the convex hull.

Let $b_{i,k}$ be the total number of lines $l(t_i, s)$, $s \in S_L$, that intersect the convex hull between points t_i and t_k and let $b_{j,k}$ be the total number of lines $l(t_j, s)$, $s \in S_L$, that intersect the convex hull between t_k and t_j .

The number of points enclosed by triangle $\triangle t_i t_j t_k$ is $|S_L| - (b_{i,k} + b_{j,k} - 1).$

The sets S_L and S_R are found in O(n) time. The convex hull can be found in $O(n \log h)$ time and the intersection of a line and the convex hull can be found in $O(\log h)$ time by performing a binary search on the edges of the convex hull. Then the *a*-variables are computed in $O(n \log h)$ time and the *b*-variables are computed in O(h) time. The total runtime is $O(n \log h)$. \Box

4 Triangles with one fixed vertex on the convex hull

Lemma 4 Let z be the lowest point in S. Let x and y be points in S such that $\triangle xyz$ encloses the maximum number of points of S. Then $\triangle xyz$ is a 2-approximation of an optimal triangle with vertices in S.

Proof. Let $\triangle pqr$ be an optimal triangle with vertices in *S*. Draw a line from *z* to each vertex of $\triangle pqr$. By Lemma 1 the point *z* cannot lie in any of the empty regions of $\triangle pqr$. Then one of the lines from *z* must cross an edge of $\triangle pqr$.

Without loss of generality assume that \overline{zp} crosses the edge \overline{qr} . Then the two triangles $\triangle pqz$ and $\triangle rpz$ cover

 $\triangle pqr$ (see Figure 5). By Observation 1 one of these triangles is a 2-approximation of $\triangle pqr$.



Figure 5: Triangles $\triangle pqz$ and $\triangle rpz$ cover $\triangle pqr$.

Theorem 5 A 2-approximation of an optimal triangle with vertices in S can be found in $O(n^2)$ time.

Proof. Let z be the lowest point in S. Count the number of points enclosed by every triangle containing vertex z and return the triangle found that encloses the most points.

There are $\binom{n}{2}$ triangles containing vertex z so this takes $O(n^2)$ time using the data structure from [1]. The approximation ratio follows from Lemma 4.

5 Triangles with at least two vertices on the convex hull

In this section we consider triangles with at least two vertices on the convex hull of S.

Lemma 6 Let \triangle be a triangle, with vertices in S, such that at least two of its vertices are on the convex hull of S, that encloses the maximum number of points of S. Then \triangle is a 3-approximation of an optimal triangle with vertices in S.

Proof. Let $\triangle pqr$ be an optimal triangle with vertices in *S*. Assume that none of the vertices of $\triangle pqr$ lie on the convex hull of *S*. Then there exist edges $\overline{t_i t_{i+1}}$, $\overline{t_j t_{j+1}}$ and $\overline{t_k t_{k+1}}$ of the convex hull that cross the empty regions of $\triangle pqr$. Figure 6 shows how we can use the end points of two of these edges, and one vertex of $\triangle pqr$, to cover $\triangle pqr$ with three triangles. By Observation 1 one of these triangles is a 3-approximation of $\triangle pqr$. \Box

This approximation factor is tight. Figure 7 shows an example of a set of points where $\triangle pqr$ encloses three times as many points as any triangle \triangle , with vertices in S, with at least two vertices on the convex hull of S. There is no such triangle \triangle that covers more than one of the shaded regions in Figure 7. If we put mpoints in each of these regions then $\triangle pqr$ will enclose 3m points while any triangle with at least two vertices on the convex hull of S can enclose at most m+1 points.



Figure 6: Three triangles that cover $\triangle pqr$.



Figure 7: A set S, with an optimal triangle $\triangle pqr$, such that there are no triangles with at least two vertices on the convex hull of S that enclose more than 1/3 times as many points as $\triangle pqr$. Symmetric cases are not shown.

Theorem 7 A 3-approximation of an optimal triangle with vertices in S can be found in $O(\min(n^2 + nh^2, nh^2 \log n))$ time.

Proof. Count the number of points enclosed by every triangle with at least two vertices on the convex hull of S and return the triangle found that encloses the most points. There are $(n - h) \binom{h}{2}$ such triangles so this takes $O(n^2 + nh^2)$ time using the data structure from [1] or $O(h^2 n \log n)$ time using the algorithm presented in Lemma 2. The approximation ratio follows from Lemma 6.

Theorem 8 A 4-approximation of an optimal triangle with vertices in S can be found in $O(n \log^2 n)$ time.

Proof. Consider the following algorithm: Sort the points of S clockwise by angle about the lowest point z in S. Let s_m be the median of S by angle and let $\overline{t_i t_{i+1}}$ be the edge of the convex hull that intersects $l(z, s_m)$. Count the number of points enclosed by every triangle $\triangle z t_i s$ and $\triangle z t_{i+1} s$, $s \in S$, using the algorithm in Lemma 2. Let S_L be the set of points lying to the left of $l(z, s_m)$ and let S_R be the set of points lying to the right of $l(z, s_m)$. Recursively run the algorithm on the sets S_L and S_R and return the triangle found that encloses the most points.

To prove the approximation ratio, let $\triangle pqr$ be an optimal triangle with vertices in S. Let x and y be points

in S such that $\triangle xyz$ encloses the maximum number of points of S. From Lemma 4 $\triangle xyz$ is a 2-approximation of $\triangle pqr$.

Consider the recursive call where x and y lie on opposite sides of the line $l(z, s_m)$. At least one of t_i and t_{i+1} must lie above l(x, y), otherwise $\overline{t_i t_{i+1}}$ wouldn't be an edge of the convex hull. If t_i lies above l(x, y) then $\triangle xyz$ is covered by triangles $\triangle zxt_i$ and $\triangle yzt_i$ (as in Figure 8). Otherwise if t_{i+1} lies above l(x, y) then $\triangle xyz$ is covered by triangles $\triangle zxt_{i+1}$ and $\triangle yzt_i$. By Lemma 1 one of these triangles is a 2-approximation of $\triangle xyz$ and, therefore, a 4-approximation of $\triangle pqr$.



Figure 8: Triangles $\triangle zxt_i$ and $\triangle yzt_i$ cover $\triangle xyz$.

Sorting the points by angle takes $O(n \log n)$ time. Finding the edge of the convex hull that intersects the line through z and the median takes $O(\log h)$ time if we perform a binary search on the precomputed edges of the convex hull. Counting the number of points enclosed by every triangle $\triangle zt_is$ and $\triangle zt_{i+1}s$, $s \in S$, takes $O(n \log n)$ time using the algorithm presented in Lemma 2. The total amount of work done at each step is $O(n \log n)$. S_L and S_R each contain half of the points of S so the complexity of this algorithm satisfies the equation $T(n) = 2T(n/2) + O(n \log n)$ which solves to $O(n \log^2 n)$.

6 Triangles with three vertices on the convex hull

In this section we consider triangles with three vertices on the convex hull of S.

Lemma 9 Let \triangle be a triangle, whose vertices are on the convex hull of S, that encloses the maximum number of points of S. Then \triangle is a 4-approximation of an optimal triangle with vertices in S.

Proof. Let $\triangle pqr$ be an optimal triangle with vertices in *S*. Assume that none of the vertices of $\triangle pqr$ lie on the convex hull of *S*. Then there exist edges $\overline{t_i t_{i+1}}$, $\overline{t_j t_{j+1}}$ and $\overline{t_k t_{k+1}}$ of the convex hull that cross the empty regions of $\triangle pqr$. Figure 9 shows how we can use the end points of these edges to find a set of at most four triangles that cover $\triangle pqr$. By Lemma 1 one of these triangles is a 4-approximation of $\triangle pqr$.


Figure 9: Four triangles that cover $\triangle pqr$.

This approximation factor is tight. Figure 10 shows an example where $\triangle pqr$ encloses four times as many points of S as any triangle \triangle , whose vertices are on the convex hull of S. There is no such triangle \triangle that covers more than one of the four shaded regions in Figure 10. If we put m points in each of these regions then $\triangle pqr$ will enclose 4m points while any triangle whose vertices are on the convex hull of S can enclose at most m + 1points.



Figure 10: A set S, with an optimal triangle $\triangle pqr$, such that there are no triangles, whose vertices are on the convex hull of S, that cover more than 1/4 times as many points as $\triangle pqr$. Symmetric cases are not shown.

Theorem 10 A 4-approximation of an optimal triangle with vertices in S can be found in $O(\min(n^2 + h^3, h^2 n \log h))$ time.

Proof. Count the number of points enclosed by every triangle with three vertices on the convex hull of S and return the triangle found that encloses the most points. There are $\binom{h}{3}$ such triangles so this takes $O(n^2 + h^3)$ time using the data structure in [1] or $O(h^2 n \log h)$ time using the algorithm presented in Lemma 3. The approximation ratio follows from Lemma 9.

Theorem 11 An 8-approximation of an optimal triangle with vertices in S can be found in $O(n \log^2 h)$ time.

Proof. Consider the following algorithm: Let $t_1 \ldots t_h$ be the vertices of the convex hull of S given in clockwise order starting at the lowest point $z = t_1$ and let t_m be the median of the convex hull. Count the number of points enclosed by every triangle containing vertices z and t_m , with the third vertex on the convex hull of S, using the algorithm described in Lemma 3. Let S_L be the set of points of S lying on or to the left of $l(z, t_m)$ and

let S_R be the set of points of S lying on or to the right of $l(z, t_m)$. Recursively run the algorithm on the sets S_L and S_R and return the triangle found that encloses the most points.

To prove the approximation ratio, let $\triangle pqr$ be an optimal triangle with vertices in S. Let x and y be points in S such that $\triangle xyz$ encloses the maximum number of points of S. From Lemma 4 $\triangle xyz$ is a 2-approximation of $\triangle pqr$.

Assume that x and y are not on the convex hull. Then there exist edges $\overline{t_i t_{i+1}}$ and $\overline{t_k t_{k+1}}$ that cross the empty regions of $\triangle xyz$. Let t_j be any point on the convex hull between t_{i+1} and t_k . Figure 11 shows how we can use the points z, t_i , t_{i+1} , t_j , t_k and t_{k+1} to construct four triangles that cover $\triangle xyz$. By Lemma 1 one of these triangles is a 4-approximation of $\triangle xyz$ and, therefore, an 8-approximation of $\triangle pqr$.



Figure 11: Four triangles that cover $\triangle xyz$.

Consider the recursive call where x and y lie on opposite sides of $l(z, t_m)$. When this occurs t_m is on the convex hull between t_{i+1} and t_k . Thus, in the previous argument, we can take $t_j = t_m$. Then in this call we count the number of points in triangles $\Delta z t_j t_{i+1}$ and $\Delta z t_j t_k$.

In another recursive call either t_i or t_{i+1} is the median of the convex hull and we count the number of points enclosed by the triangle $\triangle zt_it_{i+1}$. Similarly there is a recursive call where either t_k or t_{k+1} is the median and we count the number of points enclosed by $\triangle zt_kt_{k+1}$.

The convex hull of S can be found in $O(n \log h)$ time [2] and does not need to be computed at each step. Each step requires O(n) time to find S_L and S_R and $O(n \log h)$ time to count the number of points enclosed by every triangle with vertices z and t_m , with the third vertex on the convex hull of S, by Lemma 3. When we recursively call the algorithm on the sets S_L and S_R the size of the convex hulls of S_L and S_R are half the size of the convex hull of S and the total number of points in S_L and S_R is the number of points in S. The complexity of this algorithm satisfies the equation $T(h,n) = T(h/2, n_1) + T(h/2, n - n_1) + O(n \log h)$ for some $1 \leq n_1 < n$. The solution to this equation is $O(n \log^2 h)$. We can obtain an $O(\log h)$ -approximation of the optimal triangle with vertices in S in $O(n \log h)$ time by triangulating the convex hull of S and choosing the triangle in this triangulation that encloses the maximum number of points of S.

Let $T = \emptyset$ be an initially empty set of triangles. Initialize $R = r_1, r_2, \ldots, r_h$ to the points of the convex hull of S given in clockwise order. For each point $r_i \in R$ such that i is odd add the triangle $\triangle r_i r_{i+1} r_{i+2}$ to Tand remove r_{i+1} from R. Renumber the elements of Ras r_1, r_2, \ldots and repeat the previous steps until R has less than 3 points. This gives a triangulation T of the convex hull of S (see Figure 12). At each iteration we remove half of the points in R, so T is constructed in O(h) time after constructing the convex hull of S in $O(n \log h)$ time [2].



Figure 12: Triangulation of the convex hull of a set of points.

Lemma 12 Any line crosses at most $2 \log h$ triangles of T.

Proof. Let R_i denote the sequence of points in R at the *i*th iteration of the triangulation algorithm.

Observe that R_i and R_{i+1} are convex polygons and that any triangle added to T in the *i*th iteration has edges in R_i and R_{i+1} only (see Figure 13). Then any line can intersect at most two of the triangles of T added during the *i*th iteration of the triangulation algorithm.

There are $\log h$ iterations of the algorithm, so any line crosses at most $2 \log h$ triangles in T.

Lemma 13 Any triangle \triangle , with vertices in S, can be covered by at most $3 \log h$ triangles in T.

Proof. Observe that any triangle in T that partially covers \triangle must cross at least two edges of \triangle , since every triangle in T has vertices on the convex hull of S. By Lemma 12 each edge of \triangle can cross at most $2 \log h$ triangles in T. Then the edges of \triangle can cross at most $6/2 \log h$ different triangles in T. Therefore \triangle can be covered by at most $3 \log h$ triangles in T. \Box

Theorem 14 A $3 \log h$ -approximation of an optimal triangle with vertices in S can be found in $O(n \log h)$ time.



Figure 13: Any line can cross at most two of the triangles added during the *i*th iteration of the algorithm. The shaded regions denote triangles added to T during the *i*th iteration.

Proof. For each point $s \in S$ we can find the triangle in T enclosing s in $O(\log h)$ time: Start with the innermost triangle $\Delta t_i t_j t_k$. If s is in this triangle we are done. Otherwise s lies to the left of one of the lines $l(t_i, t_j)$, $l(t_j, t_k)$ or $l(t_k, t_i)$. Without loss of generality let s lie to the left of $l(t_i, t_j)$. Repeat the previous steps with the triangle immediately to the left of the line $l(t_i, t_j)$. At each step we remove 2/3 of the triangles. Since there are O(h) triangles it takes $O(\log h)$ time to find the triangle of T that encloses s. Therefore it takes $O(n \log h)$ time to find the triangle in T that encloses the maximum number of points of S. The approximation ratio follows from Observation 1 and Lemma 13.

7 Conclusion

It is not known whether the $O(n^3)$ time algorithm used to find the triangle enclosing the most points is optimal. Similarly it is unclear if the runtimes of our approximations are optimal.

Eppstein *et al.* [1] studied the more general problem of finding a convex k-gon that is optimal for some weight function, for example the minimum or maximum number of points, or the minimum perimeter. Their algorithm runs in $O(kn^3)$ time. It would be interesting to see if any of our results can be applied to these problems.

References

- D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.*, 7(1):45–58, 1992.
- [2] D.G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm. SIAM Journal on Computing, 15(1):287–299, 1986.

Finding the Maximum Area Parallelogram in a Convex Polygon

Kai Jin*

Kevin Matulef*

Abstract

We consider the problem of finding the maximum area parallelogram (MAP) inside a given convex polygon. Our main result is an algorithm for computing the MAP in an *n*-sided polygon in $O(n^2)$ time. Achieving this running time requires proving several new structural properties of the MAP, and combining them with a rotating technique of Toussaint [10].

We also discuss applications of our result to the problem of computing the maximum area centrallysymmetric convex body (MAC) inside a given convex polygon, and to a "fault tolerant area maximization" problem which we define.

1 Introduction

A common problem in computational geometry is that of finding the largest figure of one type contained in a given figure of another type. Over the last 30 years researchers have looked at several instances of this problem, such as finding the largest convex polygon contained in an arbitrary polygon [2], the largest axisparallel rectangle in an arbitrary polygon [3], the largest triangle inscribed in a convex polygon [4], the largest kgon in a convex polygon [1], or the largest square in a convex polygon [7].

In this work, we consider the problem of finding the maximum area *parallelogram* (MAP) inside a convex polygon. Our main result is the following:

Theorem 1 There is an algorithm for computing the MAP in a convex polygon with n sides in $O(n^2)$ time.

As we shall see, achieving an $O(n^2)$ running time is not straightforward; it requires proving several structural properties of the MAP. We discuss the challenges involved, and our techniques for overcoming them, in Section 1.2.

1.1 Applications

The MAC. One reason why the parallelogram case is of special interest is because parallelograms are the

simplest polygons that are "centrally-symmetric" (i.e. for which there exists a "center" such that every point on the figure, when reflected about the center, produces another point on the figure). It is natural to ask whether we can, in general, compute the Maximum Area Centrally-symmetric convex body (MAC) inside a given convex polygon or convex curve. Although it seems difficult to compute the area of the MAC exactly, it is known that the MAP serves as an approximation:¹

Theorem 2 [5, 8] For a convex curve Q, the area of the MAP inside it is always at least $\frac{2}{\pi} \approx 0.6366$ times the area of Q, Moreover, this bound is tight; the worst case is realized when the given convex curve is an ellipse.

Theorem 2 follows from two results.² The first result of Dowker [5] says that for any centrally-symmetric convex body K in the plane, and any even $n \ge 4$, among the inscribed (or contained) convex n-gons of maximal area in K, there is one which is centrally-symmetric. The second result of Sas [8] says that for convex bodies in \mathbb{R}^d , the hardest to approximate with inscribed n-gons are exactly the ellipsoids.

By combining Theorem 2 with our Theorem 1, we get the following corollary.

Corollary 3 There is a $\frac{2}{\pi}$ -approximation algorithm for computing the area of the MAC in $O(n^2)$ time.

Fault Tolerant Area Maximization. Consider the following general problem: you are allowed to place k points inside a polygon P, then an adversary removes j of them (where j < k). Your goal is to maximize the area of the convex hull of the remaining points. We call this the Fault Tolerant Area (FTA) Maximization Problem.

Let FTA(k, j) be the maximum area you can achieve in the worst case. It is easy to see that FTA(k, 0) is equivalent to finding the maximum area k-gon inside P. Boyce et. al. give a clever algorithm for solving this in $O(knlgn + nlg^2n)$ time [1]. However, when j > 0, the problem seems much less trivial. Perhaps the simplest

^{*}IIIS, Tsinghua University, cscjjk@msn.com and matulef@gmail.com. Supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, and the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

¹We may give the *simplest* credit to squares in some sense, but with only one constrain of been centrally-symmetric, parallelograms are *simpler* (more flexible) than squares in the lessconstrains (flexible) sense. As a result, parallelograms are more suitable for approximating the MAC than squares.

 $^{^2 \}rm An$ earlier version of this paper contained an alternative proof of Theorem 2, see http://itcs.tsinghua.edu.cn/zh/kaijin/

non-trivial case is FTA(4, 1). In this case, we show that computing FTA(4, 1) reduces to the problem of computing both the maximum area *triangle* (which can be done using Boyce et. al.'s algorithm) and the MAP. Thus, we get the following corollary to our main theorem (due to space limitations, we present the proof of this corollary in the Appendix).

Corollary 4 (Reduction) Computing FTA(4,1) in a convex polygon P can be done in $O(n^2)$ time.

1.2 Techniques

We start by proving the relatively simple fact that the MAP inside a convex polygon P must have all of its corners on the perimeter of P. This suggests the possibility of an algorithm that works by enumerating all 4-tuples of edges of P, and for each 4-tuple finding the largest parallelogram with one corner on each edge. Such an algorithm would, at best, run in $O(n^4)$ time.

To reduce the search space, we further prove that the MAP must be *anchored* on P. In other words, it must have at least one corner on a vertex of P. We prove this via a lemma we call the "hyperbola lemma" which may be of independent interest (see Section 2.2). We then divide the computation of the MAP into two cases: one where the MAP has two opposite, non-anchored corners, and one where it has two adjacent, anchored corners.

For the first case, we prove that for every pair of edges of P, finding the MAP with opposite non-anchored corners on those edges involves checking only O(n) possibilities for the placement of the other corners. As there are $O(n^2)$ pairs of edges, in total this yields an $O(n^3)$ algorithm. In order to speed it up further, we employ a rotating technique of Toussaint [Tou83]. The main idea is to show that if the pairs of edges are processed in the right order, the amortized cost of computing the best placement for the other corners is only O(1). Proving this requires proving additional structural properties of the MAP (see Section 3.1).

For the second case, when the MAP has two adjacent corners anchored on P, the algorithm is slightly more complicated, but uses similar ideas and still has running time $O(n^2)$ (see Section 3.2).

1.3 Related Work

In [2], Chang and Yap gave an algorithm for the "potato peeling" problem, or the problem of finding the largest convex polygon Q inside a given simple polygon P with n sides. They showed that this problem is computable in polynomial time, by giving algorithms computing the maximum area Q in $O(n^7)$, and the maximum perimeter Q in $O(n^6)$. Their investigation led them to define the general notion of "inclusion" problems for arbitrary classes of polygons \mathcal{P} and \mathcal{Q} . The goal of the inclusion problem on \mathcal{P} and \mathcal{Q} is to find the largest polygon from \mathcal{Q} inside a given polygon from \mathcal{P} (here "largest" can be with respect to area, perimeter, or other measures). Chang and Yap surveyed several results on the inclusion problem for specific \mathcal{P} and \mathcal{Q} , although to date no unified solution exists. For different \mathcal{P} and \mathcal{Q} , it seems different techniques must be employed. The problem we solve in this work is the specific case where \mathcal{P} is the set of convex polygons, and \mathcal{Q} is the set of parallelograms.

For the case where \mathcal{P} is the set of convex polygons, the inclusion problem has been studied for several different \mathcal{Q} . For example, given a convex polygon P with n vertices, Shamos [9] gave an algorithm for finding the diameter of P in linear time (this corresponds to \mathcal{Q} being the set of "one-edge" polygons). Dobkin and Snyder [4] gave a linear time algorithm for finding the maximum area triangle; Boyce, Dobkin, Drysdale and Guibas [1] gave an algorithm for finding maximum area/perimeter k-gons in time $O(knlgn + nlg^2n)$. De Pano Ke and O'Rourke [7] gave an algorithm for finding the largest inscribed square in time $O(n^2)$; Fekete [6] gave an algorithm for finding all anchored squares $O(n \log^2 n)$ time; For the case where \mathcal{P} is the set of all simple polygons, Daniels, Milenkovic and Roth [3] gave an algorithm for finding the maximum area axis-parallel rectangle in time $O(n \log^2 n)$.

2 Preliminaries

2.1 Basic notations and lemmas

We will use symbols A, B, A', B' to denote the four corners of a parallelogram Q = ABA'B' (the pairs A, A' and B, B' denote opposite corners). We will use the symbol E to denote the center of Q.

Definition 5 (Inscribed) We say a parallelogram Q is **inscribed** on a polygon P if and only if all four corners of Q are on the boundary of P.

Definition 6 (Anchored) We say a parallelogram Q is **anchored** on a polygon P if it is inscribed on P and at least one of its corners lies on a vertex of P.

Definition 7 (Narrow side & Broad side)

Suppose b, b' are two nonparallel edges of P. They divide the other edges of P into two sets, the edges in the **Narrow side** (where the extended lines of b and b' intersect) and the edges in the **Broad side** (where band b' are further apart), illustrated in Figure 1.

Lemma 8 The parallelogram inside P with the maximum area must be inscribed on P.

Proof. We prove this by contradiction. Suppose Q = ABA'B' is a parallelogram which has maximal area in



Figure 1: Broad side and Narrow side

P but is not inscribed in *P* (See Figure 2). Without loss of generality, assume *A* is a vertex which is not on the boundary of *P*. First, we slide segment *AB* along direction \vec{BA} for a sufficiently small distance to create A_1B_1 . Next we slide it along direction $\vec{B'A_1}$ for a sufficiently small distance to create A_2B_2 where A_2 and B_2 are still inside *P*. It's easy to see that $Area(ABA'B') < Area(A_2B_2A'B')$. □



Figure 2: Illustration of Theorem 8

Lemma 8 says if a parallelogram Q = ABA'B' is the MAP of a polygon P, then all its corner must lie on the boundary of P. For points A, B, A', B' that do not lie on vertices of P, we will use the lowercase letters a, b, a', b' respectively to denote the edges of P they lie on.

Lemma 9 (Two Lines Lemma) Given two nonparallel lines b, b' and one point E not on them, there is exactly one segment connecting b and b' with midpoint E. Moreover, the endpoints of this segment, denoted as B and B', can be computed in constant time.

This lemma is simple; we omit its proof. Next we introduce the segment version of Lemma 9, it will be used many times in our algorithm.

Suppose there are two segments $b = B_1B_2$ and $b' = B_3B_4$ which, when extended, intersect at point O. For $1 \le i \le 4$, let M_i be the midpoint of OB_i (see Figure 3). We draw a parallelogram P(b,b') such that one pair of sides is parallel to b and crossing M_3 and M_4 , and the other pair of sides is parallel to b' and crossing M_1 and M_2 .

Lemma 10 (Two Segments Lemma) There is a segment connecting b and b' with midpoint E, if and only if E is inside of P(b, b').

The proof of Lemma 10 is also simple; due to space constraints we omit it.



Figure 3: Two lines lemma and two segments lemma

Note that for a parallelogram Q with center E and opposite corners B and B' located on b and b' respectively, we know $E \in P(b, b')$ because E is the midpoint of the diagonal BB'.

2.2 The Hyperbola Lemma

Definition 11 For two nonparallel lines b_1, b_2 intersecting at O, and a point A strictly in-between them, there is a unique hyperbola asymptotic to b_1 and b_2 and intersecting A, denoted as $h_A^{b_1,b_2}$ (or h_A for short). Let $C_A^{b_1,b_2}$ (or C_A for short) denote the distance from O to the nearest point on h_A .

Lemma 12 (Hyperbola Lemma) Suppose b, b' are two nonparallel lines which intersect at origin O. Let h_1 and h_2 be two hyperbolas which are both asymptotic to b and b'. Then all parallelograms Q = ABA'B', where A, B, A', B' lie on h_1, b, h_2, b' respectively, have the same area.



Figure 4: Hyperbola Lemma (orthogonal case)

Proof. We will prove the lemma in the case when b and b' are orthogonal. The general case follows from a linear transformation.

Build a Cartesian coordinate system with origin Oand let b', b be the x-axis and y-axis (see Figure 4). Suppose the coordinates of A and A' are (x_1, y_1) and (x_2, y_2) respectively. The center E is the midpoint of AA', and thus has coordinates $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$. By Lemma 9, the coordinates of B and B' are uniquely determined, and are easily verified to have coordinates $(0, y_1 + y_2)$ and $(x_1 + x_2, 0)$. Thus, we can compute $area(Q) = x_2y_2 - x_1y_1$. Since h_1 and h_2 are hyperbolas asymptotic to b and b', this means $x_1y_1 = C_A^2/2$ and $x_2y_2 = C_{A'}^2/2$. Hence, $area(Q) = C_{A'}^2/2 - C_A^2/2$, which is invariant. \Box

We will apply the hyperbola lemma with b and b' equal to *extensions* of edges of the original polygon P. Note that to find the maximum area parallelogram in P with one vertex on b and another on b', we should choose A and A' so as to maximize $C_{A'}$ and minimize C_A . However, this is trickier than it seems, since if we are allowed to choose A and A' arbitrarily, the resulting B and B' may not actually lie on the original edges of the polygon (which are just *segments*, not lines). We discuss this complication further in Section 3.1.

Definition 13 Let b, b' be two nonparallel edges, and let c be an edge in the broad side. Then we use X_c to denote the intersection point of b and the extended line of c, Y_c to denote the intersection point of b' and the extended line of c, and Z_c to denote the midpoint of X_cY_c .

Lemma 14 Suppose D is a point on segment X_cY_c . Then C_D increases while D goes from X_c to Z_c , and while D goes from Y_c to Z_c .

Proof. We only need to prove it in the case when *b* and *b'* are orthogonal, for the same reason used in Lemma 12. Without loss of generality, assume *b*, *b'* are on the *x*, *y*-axis respectively, $X_c = (x_0, 0), Y_c = (0, y_0)$. Assume $D = (x, y_0 - x(y_0/x_0))$. It's not hard to show that $C_D = \sqrt{2 * x * [y_0 - x(y_0/x_0)]}$. Note $x * [y_0 - x(y_0/x_0)]$ is a quadratic equation maximized when $x = \frac{x_0}{2}$.

2.3 The Anchor Theorem

Theorem 15 (Anchor Theorem) The MAP in P must be anchored on P.

Proof. Suppose Q = ABA'B' is a parallelogram inscribed but not anchored on P. We will show that Qis not the MAP in P. First, assume neither pair a, a'nor b, b' is parallel to each other, otherwise the theorem is trivial to prove. Assume a is in the narrow side. We can construct a new parallelogram as follows. Since Ais not on an endpoint of a, we can move A a little bit along a so that C_A decreases (see Lemma 14). We keep the position of A' so that $C_{A'}$ doesn't change. Afterward we replace the new center E by the midpoint of segment AA'. Then according to Lemma 9, B and B' can be computed since b', b', and E are all fixed. We can make sure that B, B' will still be inside segments b, b' respectively by only moving A for a sufficiently small distance. We know that the area of this new parallelogram is larger than the area of Q according to Lemma 12. Hence Q is not the MAP in P.

Theorem 15 leads one to wonder whether the MAP must always be *double-anchored* on P (that is, whether the MAP must have *two* of its corners on vertices of P). Unfortunately, this is not the case. Figure 9 in the Appendix illustrates an example where the double-anchored MAP is smaller than the actual MAP.

To design our algorithm for finding the MAP, we divide anchored parallelograms into two cases, described by the following definitions:

Definition 16 We say that a parallelogram Q is adjacent-double-anchored on a polygon P if it is inscribed on P and two adjacent corners lie on the vertices of P.

Definition 17 We say that a parallelogram Q is **opposite-free-anchored** on a polygon P if it is anchored on P but has two opposite corners which are not anchored.

Note that for a parallelogram Q anchored on P, it must either be adjacent-double-anchored, or opposite-free-anchored; it cannot be both.

3 The Algorithm

In this section we describe our algorithm for finding the MAP in a convex polygon. Our general algorithm will actually consist of two algorithms, one to handle the case when the MAP is opposite-free-anchored, and the other to handle the case when the MAP is adjacent-double-anchored. Both algorithms use similar ideas, and have running time $O(n^2)$.

3.1 The Opposite-Free-Anchored Case

First we give an algorithm for finding the MAP when the MAP is opposite-free-anchored. Without loss of generality, assume that B, B' are not anchored, and are inscribed on b, b' respectively. Let A be the vertex in the narrow side and A' in the broad side. Let M_{b_1,b_2} (M for short) be the point in P such that $C_M^{b_1,b_2} = max\{C_V^{b_1,b_2}|V \in P\}$. For fixed b and b', the following corollary of Lemma 14 helps us compute the optimal placement of M.



Figure 5: Either $M = Z_c$ (pictured at left), or M is on a vertex of P (pictured at right)

Corollary 18 There is at most one edge c in the broad side such that Z_c actually lies on c. When there is such an edge, then $M = Z_c$. When there is not such an edge, then M is on a vertex formed by two edges denoted c and d. The point Z_c lies to the right of M, and the point Z_d lies to the left of M (see Figure 5).

We are now ready to prove one of the main theorems behind our algorithm in the opposite-free-anchored case. Suppose B and B' are non-anchored vertices on fixed edges b and b'. The following theorem reduces the computation of the optimal placement of A and A' to a finite set of possibilities:

Theorem 19 Suppose Q is the opposite-free-anchored MAP on P, where B and B' are located on (nonendpoints) of b and b', A is in the narrow side and A' is in the broad side. Then A' = M, and A is anchored on P.

Proof. From Corollary 18, we know that if a dynamic point D goes from one end in the broad side to another, C_D will increase before D reaches M, and decrease after D reaches M. So if $A' \neq M$, there exists an $A^* \in P$ near A' such that $C_{A^*} > C_{A'}$. Then, as in the proof of Theorem 15, we should be able to slightly adjust B and B' (still on b and b') to construct a new parallelogram with vertices A^* , A, and two vertices on b, b', which has area bigger than that of Q. This is a contradiction.

Similarly, suppose A is not anchored on P. There exists a point A^* near A such that $C_{A^*} < C_A$. Again, this means we should be able to slightly adjust A, and then B, B', to construct a new parallelogram with vertices A^*, A' and two vertices on b, b' with area bigger than that of Q. This is also a contradiction. \Box

Theorem 19 suggests a simple enumerative algorithm in the opposite-free-anchored case: for each pair of edges (b, b'), compute the optimal A' and A by cycling through all possibilities. This is described in Algorithm 1.

If implemented naively, the time complexity of Algorithm 1 is $O(n^3)$. In order to speed up it further, we employ a rotating technique of Toussaint [10]. The main idea is to show that it isn't necessary to spend O(n) time calculating A and A' for every pair of edges (b, b'). In fact, it can be done in *amortized* O(1) time.

1	for each $edge \ b \in P, b' \in P$ do
2	foreach vertex $V \in P$, V not on b or b' do
3	$A' \leftarrow V$ if $C_V > C_{A'}$.
4	\mathbf{end}
5	for each $edge \ c \in P, c \neq b, c \neq b'$ do
6	$A' \leftarrow Z_c \text{ if } C_{Z_c} > C_{A'} \text{ and } Z_c \in P.$
7	\mathbf{end}
8	foreach vertex $A \in P$, A not on b or b' do
9	$E \leftarrow$ the midpoint of AA' .
10	Compute B, B' by Lemma 9.
11	$Q \leftarrow ABA'B'$ if $Area(ABA'B') > Area(Q)$
	and $ABA'B'$ is inside P .
12	end
13	end
	Algorithm 1: opposite free anghored

Algorithm 1: opposite-free-anchored

Suppose we fix an edge b, and consider the sequence of pairs $(b, b'_1), (b, b'_2), (b, b'_3), \ldots$, where the edge sequence b'_1, b'_2, b'_3, \ldots is formed by walking counter-clockwise along the boundary of P. Let A_i and A'_i be the optimal values computed for the pair (b, b'_i) . Then it is possible to show that the sequences A_1, A_2, A_3, \ldots , and A'_1, A'_2, A'_3, \ldots , also move counter-clockwise along the boundary of P. Thus, for a fixed b, we only spend O(n)time calculating all values of A_i and A'_i . Repeating with all other edges in place of b yields an $O(n^2)$ time algorithm.

To prove this, there are two stages in Algorithm 1 that need to be analyzed carefully. For every pair (b, b'), the first stage (lines 2-7) takes O(n) time to find A', the second stage (lines 8-12) also takes O(n) time to enumerate all the vertices in the narrow side to find A.

To show that the first stage can be made to have small amortized cost, we cite the following lemma:

Lemma 20 (A monotone property of A')

Suppose b is fixed, and b' moves counter-clockwise along P. Then the distances between A' and b are non-decreasing. In other words, A' can also only move counter-clockwise around P (see Figure 7 for an example).

The proof of Lemma 20 is simple. We omit it due to space limitations.

For the second stage, let $P_{A'}(b, b')$ be a 2-scaling of P(b, b') around point A'. We claim that $A \in P_{A'}(b, b')$, because $E \in P(b, b')$ and E is the midpoint of AA' (see Figure 6).

Lemma 21 The parallelograms $P_{A'_1}(b, b'_1)$, $P_{A'_2}(b, b'_2)$, $P_{A'_3}(b, b'_3)$,... are all non-overlapping. Additionally, their distance to line b is decreasing (see Figure 7 for an example).



Figure 6: $P_{A'}(b, b')$ is the region where A might lie.

Proof. First, $P_{A'_i}(b, b'_i)$ are parallelograms, two sides of which are parallel to b. While b' is shifting, the distances between $P(b, b'_i)$ and line b is decreasing, and the distances between A'_i and line b is non-decreasing, so the distances between $P_{A'_i}(b, b'_i)$ and b is decreasing. Thus, they do not overlap with each other.

In line 8 of Algorithm 1, if we replace " $A \in P$ " with " $A \in P_{A'}(b, b')$ ", then for a fixed b each vertex A will be enumerated at most once. Hence the this stage can be reduced to O(1) time on average, and therefore Algorithm 1 can be implemented in $O(n^2)$ time total.



Figure 7: Illustration of Algorithm 1

3.2 The Adjacent-Double-Anchored Case

Next we give an algorithm for finding the MAP when the MAP is double-adjacent-anchored.

While it is tempting to just run Algorithm 1 and hope that it works in this case too, unfortunately it does not. The reason is that Theorem 19 crucially assumes that B and B' are flexible on b and b', in order to reduce the space of possible values for A and A'. Without the guarantee that B and B' are not anchored, it is possible that the best choice of A' is not equal to M, or that the best choice of A is not on a vertex of P.

Nevertheless, when we assume the MAP has two adjacent anchored vertices, we can still prove some constraints on the placement of the other vertices. This allows us to develop an algorithm similar to Algorithm 1 that enumerates over all choices of anchored vertex Band opposite edge b'. If implemented correctly, this algorithm can be made to run in $O(n^2)$ by showing an amortized analysis similar to the one we used before.

Due to space limitations, we present the full details of our algorithm in the adjacent-double-anchored case in the Appendix.

Acknowledgments

The authors are grateful to the anonymous reviewers of an earlier version of this paper for pointing out the references [5, 8] and for other helpful comments.

The authors would also like to thank Xiaoming Sun, Tiancheng Lou, and Zhiyi Huang for taking part in fruitful discussions.

References

- J. E. Boyce, D. P. Dobkin, R. L. Drysdale, III, and L. J. Guibas. Finding extremal polygons. In *Proc. of* the 14th annu. ACM symp. on Theory of comp., STOC '82, pages 282–289, New York, NY, USA, 1982. ACM.
- [2] J. Chang and C. Yap. A polynomial solution for the potato-peeling problem. *Discrete and Computational Geometry*, 1:155–182, 1986. 10.1007/BF02187692.
- [3] M. Daniels and Roth. Finding the largest area axisparallel rectangle in a polygon. CGTA: Computational Geometry: Theory and Applications, 7, 1997.
- [4] D. Dobkin and L. Snyder. On a general method for maximizing and minimizing among certain geometric problems. In *Proceedings of the 20th Annual Symposium* on FOCS, pages 9–17. IEEE Computer Society, 1979.
- [5] C. Dowker. On minimum circumscribed polygons. Bull. Amer. Math. Soc, 50:120–122, 1944.
- [6] S. P. Fekete. Finding all anchored squares in a convex polygon in subquadratic time. In Proc. 4th Canad. Conf. Comput. Geom., pages 71–76, 1992.
- [7] J. O. N. Adlai De Pano, Yan Ke. Finding largest inscribed equilateral triangles and squares. In *Proc. Aller*ton Conf., pages 869–878, 1987.
- [8] E. Sas. über ein extremumeigenschaft der ellipsen. Compositio Math, 6:468–470, 1939.
- [9] M. Shamos. Computational geometry. 1978.
- [10] G. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, pages 1–4. Citeseer, 1983.

Illumination problems on translation surfaces with planar infinities

Nikolay Dimitrov *

Abstract

In the current article we discuss an illumination problem proposed by Urrutia and Zaks. The focus is on configurations of finitely many two-sided mirrors in the plane together with a source of light placed at an arbitrary point. In this setting, we study the regions unilluminated by the source. In the case of rational- π angles between the mirrors, a planar configuration gives rise to a surface with a translation structure and a number of planar infinities. We show that on a surface of this type with at least two infinities, one can find plenty of unilluminated regions isometric to unbounded planar sectors. In addition, we establish that the non-bijectivity of a certain circle map implies the existence of unbounded dark sectors for rational planar mirror configurations illuminated by a light-source.

1 Introduction

Consider a planar domain with a light reflecting boundary. Place a source of light at a point inside the domain. Assume that the source emits rays in all directions. Each ray follows a straight line and whenever it reaches the boundary it is reflected according to the rule that the angle of incidence equals the angle of reflection. A point from the domain is considered *illuminated* by the source whenever there is a ray that reaches the point either directly or after a series of reflections. In this setting, one can ask the following questions, also known as *illumination problems*.

Question 1 If we place the source of light at any point in the domain, will all of the domain be illuminated? If not, what could be said about the non-illuminated regions?

Question 2 Is there a point from which the light source can illuminate the entire domain?

These problems are often attributed to E. Straus who posed them sometime in the early fifties and first published by V. Klee in 1969 [5]. Some famous examples and interesting results are Penrose's room [1], Tokarsky's example [5] as well as the article [3] by Hubert, Schmoll and Troubetzkoy on illumination on Veech surfaces. In 1991, J. Urrutia and J. Zaks proposed the following problem [6]. Assume we are given a finite number of disjoint compact line segments in the plane each representing a mirror that reflects light on both sides (a two-sided mirror). Let p_0 be any point on the plane not incident to any of the segments. Then, the complement of the set of mirrors is an unbounded domain with lightreflecting boundary and if we place a source of light Sat p_0 we can pose questions 1 and 2. Figure 1a depicts an example of a two-sided mirror configuration with a light emitting source S. The convex hull of the mirrors is a polygon. If S is in the convex hull, one can construct a triangle P unilluminated by S, like the shaded one on figure 1b. To do that, it is sufficient for a mirror segment to be an edge of the convex hull.



In this paper we are interested in finite two-sided mirror configurations with the following property: any pair of lines determined by the mirror segments are either parallel or intersect at an angle which is a rational multiple of π . We will call such a configuration a rational mirror configuration and the domain obtained as a complement of the mirrors will be called rational mirror domain. For those, we will find conditions that will guarantee the existence of unbounded unilluminated sectors in the plane (see definition 2).

A rational mirror domain can be "unfolded" into a surface that carries a flat metric with conical singularities and trivial holonomy group (see section 3 or [2, 4]). This means that the surface has a special atlas, called a *translation atlas*, with the property that *away from* the cone points, the transition maps between two charts from the atlas are Euclidean translations (section 3 or [2, 4]). As a result, the piecewise linear trajectory of a light ray in the original domain becomes a smooth

^{*}Department of Mathematics and Statistics, McGill University, dimitrov@math.mcgill.ca

geodesic on the flat surface. Thus, one can think of a light source placed at a nonsingular point on the surface, emitting geodesic rays in all directions. Any other point is considered *illuminated* if there is a smooth geodesic connecting the source to the point. In this way, one can ask questions 1 and 2 for the surface. Notice that there are regions on it isometric to complements of compact sets in the plane. We will call a surface with such a geometry a translation surface with planar infinities.

A translation surface with planar infinities gives rise to a pair (X, ω) where X is a closed surface with a complex structure and ω is a meromorphic differential on X with only double poles and zero residues. The zeroes of ω are the cone points of the flat structure [2, 4], and around each pole the surface looks like the complement of a compact set in the plane. The converse is also true. A pair (X, ω) of a closed Riemann surface and a meromorphic differential with only double poles and zero residues induces a translation structure on X with planar infinities. We have provided more details, definitions and constructions in section 3. For a good introduction to the theory of polygonal billiards and translation surfaces, we recommend [2] and [4].

Definition 1 The pair (X, ω) is called a translation surface with planar infinities whenever the following conditions hold:

- (1) X is a closed surface with a complex structure;
- (2) ω is a meromorphic differential on X;

(3) Every pole of ω is of order exactly 2 and the residue at that pole is zero. We will refer to the poles of ω as planar infinities.

In this study we would like to show non-illumination of a special type of domains both on a translation surface with planar infinities and in the plane.

Definition 2 a) Let l_1 and l_2 be two half-lines in the plane both starting form a point p_0 and going to infinity. Let θ be the angle between l_1 and l_2 at the vertex p_0 , measured counterclockwise from l_1 to l_2 . Then, the open region C bounded by l_1 and l_2 , whose internal angle at p_0 is θ , is called an infinite sector of angle θ (see figure 2a).

b) An open subdomain C of a translation surface with planar infinities (X, ω) is called an infinite sector of angle θ whenever there exists a chart from the translation atlas of (X, ω) that maps C isometrically to a planar infinite sector of angle θ like the one defined in point a.

On any translation surface (X, ω) one can always find an orientable foliation \mathcal{F}_{ω} with singularities, whose leaves are geodesics. Indeed, let us foliate the Euclidean plane into horizontal straight lines, oriented as usual from left to right. Since each transition map between

118

two charts is a Euclidean translation, it sends horizontal lines to horizontal lines (line orientation preserved). Thus, pulling back onto the surface the planar horizontal foliation from all translation charts defines globally the desired foliation \mathcal{F}_{ω} . Moreover, the singularities of \mathcal{F}_{ω} are the cone points of the surface (X, ω) , i.e. the zeroes of the differential ω . We call \mathcal{F}_{ω} the horizontal foliation of the surface and its leaves - the horizontal geodesics of the surface. At each non-singular point p_0 of (X, ω) the oriented horizontal geodesic $l_{p_0}(0)$ from \mathcal{F}_{ω} defines a positive horizontal direction at p_0 . The counterclockwise angle α between $l_{p_0}(0)$ and an arbitrary oriented geodesic $l_{p_0}(\alpha)$ through p_0 is called the direction of $l_{p_0}(\alpha)$ at p_0 (see figure 2b). From now on, $l_{p_0}(\alpha)$



Figure 2:

denotes the geodesic ray on (X, ω) starting from $p_0 \in X$ and going in the direction of angle α . It is important to emphasize that, since we are working with a translation surface, the intersection of the geodesic $l_{p_0}(\alpha)$ with any other horizontal geodesic $l_q(0)$ will always form the same angle α , as shown locally on figure 2b. In other words, just like in the plane, a geodesic on (X, ω) does not changes its angle with respect to the horizontal direction. Since a direction at any non-singular point $p \in X$ is defined as an angle $\alpha \in \mathbb{R} \mod 2\pi$, we can identify the set of all directions at p with the unit circle $S^1 = \{z \in \mathbb{C} : |z| = 1\}$. The point $1 \in S^1$ gives the horizontal direction $\alpha = 0$.

2 Results

It is natural to ask questions about the behavior of the geodesics on a surface. The first question we will address is the following. On a translation surface with planar infinities, where do most geodesics emanating from a nonsingular point go? As it turns out, almost all of them fall onto the poles of the surface. Same is true for any rational mirror configuration in the plane.

Theorem 1 The following two statements are true:

(1) Let (X, ω) be a translation surface with planar infinities and let $p_0 \in X$ be non-singular. Then the set of all directions $\alpha \in S^1$, for which the geodesic passing through p_0 in direction α goes to one of the poles of ω , is open and dense in the circle S^1 ;

(2) Assume we are given a rational mirror configuration in the plane and let p_0 be a point not lying on any of the mirrors. Then the set of all directions $\alpha \in S^1$, for which the piece-wise linear reflected trajectory starting from p_0 in direction α goes to infinity, is open and dense in the circle S^1 .

The next result establishes the existence of infinite unilluminated sectors and large unbounded regions on translation surfaces with more than one planar infinity.

Theorem 2 Let (X, ω) be a translation surface with at least two planar infinities. Then, for any point p_0 on $X \setminus (zeroes(\omega) \cup poles(\omega))$ there exists an infinite sector C on (X, ω) unilluminated by p_0 , i.e. for any point $p \in C$ there is no smooth geodesic on (X, ω) that connects p_0 to p. Moreover, there exists a region on (X, ω) consisting of unilluminated, non-overlapping infinite sectors of total angle $2\pi(k-1)$, where k is the number of poles of ω .

The main ideas used in the proof of theorem 2 can be adjusted to the study of illumination problems for rational mirror configurations in the plane. For instance, an interesting question put in an every day language, is the following. How big of an object can be hidden from a stationary observer in a rational mirror domain? Can we hide a car? A whole parking lot of cars? Precisely speaking, we would like to find a basic condition that will ensure the existence of an infinite unilluminated sector for a light source placed at a point inside a rational mirror domain.

Let D be a rational mirror domain and let $p_0 \in D$. Draw a large enough circle K, so that its interior contains the mirrors from the configuration and the light source at the point p_0 . Denote by U_{p_0} the open dense set of all directions which go to infinity, provided by theorem 1. For an angle $\alpha \in U_{p_0} \subset S^1$ follow the straight line $l_{p_0}(\alpha)$ starting form p_0 in direction of α . Whenever



Figure 3:

the line reaches a mirror it is reflected, changing its direction. In this way, a piecewise linear trajectory is formed, which at some point leaves the disc bounded by K never to come back to it. Denote by $f_{p_0}(\alpha)$ the angle between the horizontal direction of \mathbb{C} and the portion of the trajectory that is outside the circle K. As a result, we obtain a map $f_{p_0}: U_{p_0} \longrightarrow S^1$. For a picture of the construction of f_{p_0} see figure 3. The map f_{p_0} is defined almost everywhere on the unit circle. In fact, its domain U_{p_0} is open and dense in S^1 . Moreover, f_{p_0} is a rotation when restricted to any connected component of U_{p_0} . Our hope is that finding ways to study the combinatorial properties of f_{p_0} may facilitate the search for unbounded unilluminated sectors in rational mirror domains.

Theorem 3 Assume we are given a rational mirror configuration. For an arbitrary point p_0 not on any of the mirrors, consider the circle map f_{p_0} (see figure 3). If f_{p_0} is not injective, then there exists an infinite sector in the plane unilluminated by p_0 .

3 Translation surfaces.

In the current section we discuss translation surfaces and show how to construct one from a rational mirror configuration. To illustrate the idea better, we apply the procedure to an example.

Various descriptions. A translation surface is a closed surface X with a finite set of points $\Sigma \subset X$, called singularities, and a cover of $X \setminus \Sigma$ by open charts $\{(W_a, \varphi_a) \mid W_a \subseteq X \setminus \Sigma, \varphi_a : W_a \to \mathbb{C}\}$ having the property that whenever $W_a \cap W_b \neq \emptyset$ the transition map between the two charts (W_a, φ_a) and (W_b, φ_b) is a Euclidean translation, i.e. $z_b = \varphi_b^{-1} \circ \varphi_a(z_a) = z_a + c$. In our study, Σ partitions into two subsets Σ_0 and Σ_{∞} . Each point from Σ_0 has a cone angle of $2\pi N$, where N is a positive integer. Each point p_{∞} form Σ_{∞} has an open neighborhood $W' \subset X$ with a map $\varphi_{\infty} : W' \setminus \{p_{\infty}\} \to \mathbb{C}$ such that $(W' \setminus \{p_{\infty}\}, \varphi_{\infty})$ is a translation chart from the atlas and the set $\mathbb{C} \setminus \varphi_{\infty}(W' \setminus \{p_{\infty}\})$ is compact. Thus, the collection Σ_{∞} contains all planar infinities on the surface.

Since translations are holomorphic maps, the translation atlas induces a complex structure on X (for details see [2] and [4]). Moreover, the differential dz_a in each $\varphi(W_a) \subset \mathbb{C}$ can be pulled back as a holomorphic differential $\omega_a = \varphi_a^* dz_a$ in the corresponding W_a . But if $z_b = \varphi_b^{-1} \circ \varphi_a(z_a) = z_a + c$ then $dz_b = dz_a$. Hence, $\omega_a = \omega_b$ in any intersection $W_a \cap W_b \neq \emptyset$ which gives rise to a global holomorphic differential ω on $X \setminus \Sigma$. Moreover, ω extends to the singular set Σ so that Σ_0 becomes the set of zeroes of ω and Σ_{∞} becomes the set of all poles of ω . The latter are all double and with residue 0. So we see that a translation surface with planar infinities induces a pair (X, ω) of a compact Riemann surface without boundary together with an appropriate meromorphic differential.

To recover the translation atlas from a pair (X, ω) , one can cover $X \setminus (\operatorname{zeroes}(\omega))$ with topological discs W_a . On each of them define the chart $\varphi_a(p) = \int_{p_a}^p \omega$, where $p_a \in W_a$ is fixed and p varies in W_a . As ω is either holomorphic or meromorphic with a double pole and residue 0 inside the topological disc W_a , the path of integration in $W_a \setminus \operatorname{poles}(\omega)$ is arbitrary. If $W_a \cap W_b \neq \emptyset$ then $z_b = \int_{p_b}^p \omega = \int_{p_a}^p \omega + \int_{p_b}^{p_a} \omega = z_a + c$ for $p \in W_a \cap W_b$. Thus, we have obtained the desired translation atlas. As we can see, the description of a translation surface with planar infinities which we gave in the beginning of the current section is equivalent to definition 1.

The horizontal foliation \mathcal{F}_{ω} on X, mentioned in the introduction, is defined as follows. Let $\mathcal{F}_{\mathbb{C}}$ be the foliation of horizontal lines $\{z \in \mathbb{C} | \operatorname{Im}(z) = s\}, s \in \mathbb{R}$ in \mathbb{C} oriented from left to right (see figure 2b). Define the pulled-back local foliation $\mathcal{F}_a = \varphi_a^* \mathcal{F}_{\mathbb{C}}$ in each W_a . Observe that $\mathcal{F}_{\mathbb{C}}$ is invariant with respect to any translation, i.e. the translations map any horizontal line to a horizontal line. Hence, $\mathcal{F}_a = \mathcal{F}_b$ on each $W_a \cap W_b \neq \emptyset$. Thus, all local foliations fit together in a global foliation \mathcal{F}_{ω} on X with geodesic leaves and singularities Σ . The oriented leaves of \mathcal{F}_{ω} determine globally a horizontal direction on (X, ω) . Since translations are Euclidean isometries, the Euclidean metric on $\mathbb C$ induces a Euclidean metric on $X \setminus \Sigma$. In this metric geodesics that do not go through singularities are isometric to straight lines in \mathbb{C} . The notion of a direction at a non-singular point $p \in X$ is as defined in the introduction. It is the counterclockwise angle between the horizontal leaf and an oriented geodesic both passing through p. Finally, an oriented geodesic always forms the same angle with any horizontal leaf it intersects, so it never self-intersects, except possibly to close up.



Figure 4:

Construction. Assume we have a configuration of disjoint compact line segments $I_1, ..., I_m$ in the plane \mathbb{C} , which we regard as two-sided mirrors. The angle between any two of them is a rational-multiple of π . Observe that if one of the mirrors forms a rational- π angle with the rest of the mirrors, then immediately follows

that any pair of mirrors forms a rational- π angle. This is a consequence of the fact that in an Euclidean triangle the angles at the vertices sum up to π .

To understand better the construction that follows, one could have a simple toy-example in mind. Let us have two perpendicular mirrors I_1 and I_2 in the plane \mathbb{C} like the ones depicted on figure 4.

Begin by slicing \mathbb{C} along the segments $I_1, ..., I_n$ to obtain a closed slitted domain D^* in which every mirror segment I_k is doubled in order to obtain two parallel copies I_k^+ and I_k^- that form the boundary component of the surface D^* around the slit I_k . For an intuitive geometric picture of D^* in the case of the toy-example, look at figure 4. Then D^* is homeomorphic to a oncepunctured sphere with n disjoint open discs removed, as shown on figure 5 for the case of two orthogonal mirrors. In particular, $\partial D^* = \bigsqcup_{k=1}^n (I_k^+ \cup I_k^-)$.



Figure 5:

For each segment I_k , fix the line $l_k \subset \mathbb{C}$ through $0 \in$ \mathbb{C} parallel to I_k . Denote by σ_k the reflection of \mathbb{C} in l_k . The group G generated by all σ_k , k = 1, ..., n is a finite group. If α_1 is a generic direction in \mathbb{C} , then $G(\alpha_1) = \{g(\alpha_1) | g \in G\} = \{\alpha_1, ..., \alpha_m\}$ is an orbit of maximal length $m \leq n$. In our example $G \cong \mathbb{Z}_4$ and a generic orbit has 4 elements. Pick m copies D_i^* of D^* each with a choice of a direction α_i in it. If you prefer more formally, let $D_j^* = (D^*, \alpha_j)$. On figure 5, in the case of the toy-example, we can see a topological model of these four slitted planes with a choice of direction on each of them. We glue D_i^* to D_i^* if and only if there is a segment $I_k \subset \mathbb{C}$ whose corresponding reflection σ_k satisfies $\sigma_k(\alpha_i) = \alpha_i$. The gluing is done in the following way. Take D_i^* and $\sigma_k(D_j^*)$. Glue the edge $I_k^+ \subset D_i^*$ to the edge $\sigma_k(I_k^+) \subset \sigma_k(D_i^*)$ and the edge $I_k^- \subset D_i^*$ to the edge of $\sigma_k(I_k^-) \subset \sigma_k(D_i^*)$. On figure 4 of the toyexample, we have chosen i = 1 and j = 2. The upper edge $I_1^+ \subset D_1$ of the cut I_1 is glued to the lower edge $\sigma_1(I_1^+) \subset \sigma_1(D_2^*)$ of the cut $\sigma_1(I_1)$. Analogously, the lower edge I_1^- from D_1^* is glued to upper edge $\sigma(I_1^-)$ from $\sigma_1(D_2^*)$.

Both D_i^* and $\sigma_k(D_j^*)$ are naturally translation surfaces with piecewise geodesic boundaries, global coordinates z_i and z_j , and differentials dz_i and dz_j respectively. Segments I_k and $\sigma_k(I_k)$ are equal and parallel, hence the gluing map is a translation $z_j = z_i + c$ (see the gluing of the shaded pieces on figure 4). Therefore the resulting surface made out of D_i^* and $\sigma_k(D_j^*)$ has a translation structure. Moreover, $dz_j = dz_i$ along the gluing locus, so there is a well-defined holomorphic differential on the new surface which extends meromorphically to both of its infinity points.

Now, follow the described gluing procedure for all cuts on the pieces D_j^* , where j = 1, ..., m. The final result is a closed Riemann surface X and a meromorphic differential ω with only double poles and zero residues, as well as simple zeroes with cone angle 4π . For the example of the two orthogonal mirrors, figure 5 illustrates how the four pieces $D_1^*, ..., D_4^*$ fit together to form a compact torus X with a complex structure and a meromorphic differential ω on X. There are eight simple zeroes of ω and four double poles. The zeroes are obtained from identifying pairs of black vertices on the segments I_k form figure 4. The cone angle at each zero is 4π and the residue at each pole is 0 as desired.

4 Proofs

Proof of theorem 1. From now on (X, ω) is an arbitrary translation surface with planar infinities and $p_0 \in X \setminus (\operatorname{zeroes}(\omega) \cup \operatorname{poles}(\omega))$ any fixed point. The idea is to cut out a rectangle around each pole $\infty_j \in \operatorname{poles}(\omega)$ and replace it by a one-handle. Indeed, choose a small





topological disc W around ∞_i and map it to \mathbb{C} by $\varphi(p) = \int_{q_0}^p \omega$ where p varies in W and $q_0 \in W$ is fixed. Notice, φ is well defined as the residue at ∞_i is 0, so the path of integration is irrelevant. The image $\varphi(W) \subset \mathbb{C}$ is the complement of a compact set (the total shaded region on figure 6 stretching to infinity). Draw a rectangle $Q \subset \varphi(W)$ as shown on figure 6 and remove its exterior (the darker region). On the surface, we remove the darker rectangular domain containing ∞_i . Then glue together the lower horizontal edge of Q to the upper and the left to the right, like gluing a torus. The gluing maps are clearly a vertical and a horizontal translation respectively. Therefore we obtain a handle with a translation structure compatible with the structure on the rest of the surface (see figure 6). By doing this for each ∞_i , we obtain a compact translation surface $(X, \tilde{\omega})$ of genus(\tilde{X}) = genus(X) + \sharp (poles(ω)), where $\tilde{\omega}$ is now holomorphic (has no poles). A lot is known about the behavior of the geodesics on such surfaces [2], [4], [7], so we use this knowledge in our advantage. Let $\tilde{\Lambda}_{p_0}$ be the set of all directions $\theta \in S^1$ for which the geodesic $\tilde{l}_{p_0}(\theta)$ on \tilde{X} is closed or hits a zero of $\tilde{\omega}$. Also, let $\tilde{\Xi}$ be the set of all directions $\theta \in S^1$ for which the geodesic flow of $(\tilde{X}, \tilde{\omega})$ in direction of θ is minimal [4] (e.g. an ergodic flow is minimal [2],[4]). Then $\tilde{\Lambda}_{p_0}$ is countable but dense in S^1 (see [7]) and $\tilde{\Xi}$ is dense and of full measure in S^1 (see [4], [2]). As a result, the set $\tilde{\Theta}_{p_0} = \tilde{\Xi} \setminus \tilde{\Lambda}_{p_0}$ consists of all $\theta \in S^1$ for which the geodesic ray $\tilde{l}_{p_0}(\theta)$ is dense in \tilde{X} . Moreover, $\tilde{\Theta}_{p_0}$ is dense and of full measure in S^1 . Therefore, for any $\theta \in \tilde{\Theta}_{p_0}$ the corresponding geodesic ray $l_{p_0}(\theta)$ on the original surface (X, ω) hits a pole of ω .

Let $U_{p_0} \subset S^1$ be the set of all directions $\theta \in S^1$ with the property that the geodesic ray $l_{p_0}(\theta)$ on (X, ω) in the direction of θ reaches a pole of ω . Since the geodesic flow on (X, ω) depends continuously on the initial point and direction, the condition that a geodesic ray reaches a planar infinity is open. Therefore, for each $\theta \in U_{p_0}$ there exists an open circular interval $(\alpha, \beta) \subset U_{p_0}$ that contains θ and for any $\theta' \in (\alpha, \beta)$ the ray $l_{p_0}(\theta')$ also reaches the same infinity. Hence, U_{p_0} is open in S^1 . Moreover, the dense set of full measure $\tilde{\Theta}_{p_0}$ is contained in U_{p_0} . Therefore, U_{p_0} is open and dense set of full measure in S^1 .

The second part of theorem 1 follows from the first one. If we are given a rational mirror configuration, unfold it into a translation surface with planar infinities (X, ω) as described earlier. Then, the infinity of the mirror domain lifts to the set of poles of ω on X and we apply the first part of the theorem.

Proof of theorem 2. As an open dense subset of S^1 , the constructed U_{p_0} is a countable disjoint union of open circular intervals $(\alpha_j, \beta_j) \subset S^1$, i.e. $U_{p_0} = \bigsqcup_{j=1}^{\infty} (\alpha_j, \beta_j)$. By construction, the geodesic rays $l_{p_0}(\theta)$ emitted from p_0 in all directions $\theta \in (\alpha_i, \beta_i)$ go to the same pole of ω . Fix some j and take a subinterval $(\alpha^*, \beta^*) \subseteq (\alpha_i, \beta_i)$ (it may even be convenient to choose $(\alpha^*, \beta^*) = (\alpha_i, \beta_i)$. Choose (α^*, β^*) so that its measure is less than π . Notice, that for every $\theta \in (\alpha^*, \beta^*)$, each ray $l_{p_0}(\theta)$ on X goes to the same $\infty^* \in \text{poles}(\omega)$. In particular, $\infty^* = \infty_3$ on figure 7. As $\sharp(\text{poles}(\omega)) \geq 2$, take another $\infty \in \text{poles}(\omega) \setminus \{\infty^*\}$ and call it ∞_1 just like on our picture below. Choose a "small" topological disc W around ∞_1 with the property $W \cap (\operatorname{zeroes}(\omega) \cup \operatorname{poles}(\omega)) =$ $\{\infty_1\}$. Define the translation chart $\varphi(p) = \int_{a_0}^p \omega$, where p varies in W and $q_0 \in W$ is fixed. The zero residue at ∞_1 guaranties independence of the integral on the path between q_0 and p in W. On figure 7 we have also provided an analogous chart ψ around p_0 . From now on, we use the same notations in W as the ones in $\varphi(W)$. Thus, we identify W with $\varphi(W)$. In \mathbb{C} the domain W looks like the complement of a compact set (the shaded region on figure 7). Let $K \subset W$ be a Euclidean circle in \mathbb{C} centered at O and containing $\mathbb{C} \setminus W$ in its interior. Abusing notation, let α^* and β^* be the two points on the circle K such that the counter-clockwise angles between the positive horizontal line through O in \mathbb{C} and the radii $O\alpha^*$ and $O\beta^*$ are respectively α^* and β^* . Let points T_1 and T_2 on K be such that counter-clockwise $\angle \alpha^* OT_1 = \angle T_2 O\beta^* = \frac{\pi}{2}$. Draw the lines t_1 and t_2 tangent to circle K at T_1 and T_2 respectively. Then they bound an infinite sector C, depicted on figure 7 as a darker shaded region.





We claim that that $C \subset X$ is not illuminated by p_0 . Assume that for some point $p \in C$ there exists $\theta \in S^1$ such that the geodesic $l_{p_0}(\theta) \subset X$ staring from p_0 in the direction of θ passes through p. Then, clearly $l_{p_0}(\theta)$ goes to ∞_1 . As already commented in the introduction, any smooth geodesic on a translation surface forms the same angle with the horizontal direction at every point it passes through. In particular, the angle between $l_{p_0}(\theta)$ and the horizontal direction in the chart W as well as near the point p_0 is always θ . By looking at the picture of the chart W on figure 7, we see that $\theta \in (\alpha^*, \beta^*)$ in W. Hence $\theta \in (\alpha^*, \beta^*) \subset S^1$ at the point p_0 as well. By the choice of the circular interval (α^*, β^*) , the geodesic ray $l_{p_0}(\theta)$ should go to $\infty^* \neq \infty_1$. But a geodesic ray can only reach one pole of ω , so we get to a contradiction. Therefore, the infinite sector Con (X, ω) is not illuminated by $p_0 \in X$.

To conclude the proof, notice that for each circular interval $(\alpha^*, \beta^*) \subset U_{p_0}$ the unilluminated sector C near ∞_1 can be also constructed around any other pole $\infty \neq \infty^*$ of ω , i.e. there are k-1 unilluminated copies of C. Partition U_{p_0} into disjoint subintervals for which we can apply the construction of unilluminated infinite sectors from the preceding two paragraphs. Thus, the the total sum of the angles of all unilluminated sectors constructed on (X, ω) is k-1 times the total measure of $U_{p_0} \subset S^1$ which is 2π . Hence, the total angle is $2\pi(k-1)$.

Proof of theorem 3. Let $D \subset \mathbb{C}$ be a rational mirror domain and $p_0 \in D$ (see figure 1 or 3). Recall the finite group G generated by all reflections in the lines through $0 \in \mathbb{C}$ parallel to the mirrors. It acts on S¹ by

rotations. Let $f_{p_0}: U_{p_0} \to S^1$ be the map described at the end of subsection "Main results" (see also figure 3) and assume it is not injective. Then, there are $\theta_1 \neq \theta_2$ from U_{p_0} such that $f_{p_0}(\theta_1) = f_{p_0}(\theta_2)$. Take the finite orbit $G(\theta_1) = \{g(\theta) \in S^1 \mid g \in G\}$. Then $\theta \in G(\theta_1)$ if and only if $f_{p_0}(\theta) \in G(\theta_1)$ so $\theta_2 \in G(\theta_1)$. Hence, the restriction $f_{|_{G(\theta_1)}}:G(\theta_1)\to G(\theta_1)$ is not bijective and there is $\theta^* \in G(\theta_1)$ such that $\theta^* \in U_{p_0} \setminus f_{p_0}(U_{p_0})$. Since f_{p_0} is a restriction of a rotation on each connected component of U_{p_0} , there is $(\alpha^*, \beta^*) \ni \theta^*$ such that $(\alpha^*, \beta^*) \subset U_{p_0} \setminus f_{p_0}(U_{p_0})$. Remember the circle K from figure 3 that encompasses the mirrors and p_0 . Using the circular interval (α^*, β^*) , we can carry out absolutely the same construction as the one in the chart W described in the proof of theorem 2. For a picture of this construction look at the rightmost large shaded area W on figure 7. Observe that the notations of the current proof match the picture's notations so that we can use it directly, thinking that the set of mirrors is in the little white elliptic region containing the center O. We claim that the infinite sector C (the darker shaded area) is not illuminated by the source $p_0 \in D$. Indeed, assume there is a light ray emitted by p_0 that reaches some $p \in C$. Then, from the picture, the direction of this ray is $\theta \in (\alpha^*, \beta^*)$. But the light ray started from p_0 in some direction $\theta_0 \in S^1$, so $\theta = f_{p_0}(\theta_0)$ which is a contradiction.

References

- Croft, H. T.; Falconer, K. J.; and Guy, R. K., "Unsolved Problems in Geometry", New York, Springer-Verlag, 1991
- Hubert, P.; Schmidt, T. A., An introduction to Veech surfaces, Handbook of dynamical systems.
 Vol. 1B, Elsevier B. V., Amsterdam, 2006, pp. 501-526
- [3] Hubert, P.; Schmoll, M.; Troubetzkoy, S., Modular fibers and illumination problems, Int. Math. Res. Not. IMRN 2008, no. 8, Art. ID rnn011, 42 pp.
- [4] Masur, H., Ergodic theory of translation surfaces, Handbook of dynamical systems. Vol. 1B, Elsevier B. V., Amsterdam, 2006, pp. 527-547
- [5] Tokarsky, G. W., Polygonal rooms not illuminable from every point, Amer. Math. Monthly 102, 1995, pp. 867–879.
- [6] Urrutia, J., Open problems on mirrors, personal website of Jorge Urrutia, URL: http://www.matem .unam.mx/~urrutia/openprob/Mirrors/
- [7] Vorobets, Y., Planar structures and billiards in rational polygons: the Veech alternative, Russ. Math. Surv. 51, 1996, no. 5, pp. 779-817,

Detecting VLSI Layout and Connectivity Errors in a Query Window

Ananda Swarup Das^{*}

Prosenjit Gupta[†]

Kannan Srinathan[‡]

Abstract

The VLSI layout designing is a highly complex process and hence a layout is often subjected to Layout Verification that includes (a) Design Rule Checking to check if the layout satisfies various design rules and (b) Connectivity Extraction to check if the components of the layout are properly electrically connected. In this work we study two geometric query problems which have applications in the above layout verification phase.

1 Introduction

A VLSI chip consists of millions of transistors. Often for the ease of fabrication, these transistors are grouped together to form blocks. Each block has pins on its periphery. Each pin is supposed to carry a signal which is denoted by a net id associated with it. All the pins of the same net id should be connected by wires which is done in the routing phase of VLSI physical design life cycle. The routing phase is again divided into two phases namely (a) global and (b) detailed. In the global routing phase the regions through which the routing is to be carried out are decided. The actual wiring is done in the detailed routing phase. The detailed routing phase is again of two types namely (i) detailed unrestricted routing and (ii) detailed restricted routing. In the detailed restricted routing phase, either channels or switch-boxes are used for routing. In this work, we will assume that channels are used for routing. However, our work has applications even when switch-boxes are used. In fact our first problem has applications even when detailed unrestricted routing is used.

Channels are basically routing regions and are abstracted as rectangles with two sides being open. We present a pictorial representation of channels in Figure 1. In a channel, pins are placed on the either side of the boundaries. The dotted horizontal lines in Figure 1 are the tracks. The thick black horizontal segments on the tracks are the trunks. Pins are connected to the trunks using branches and two trunks on two different tracks are connected using doglegs.



Figure 1: The figure depicts a channel. Pins are on either of the boundaries of the channel.

It should be noted that no two pins of different signal ids should be connected to the same trunk. Also, no two trunks being connected to pins of different signal ids should be connected. As stated before, all the pins of same net id should be electrically connected. Consider Figure 1. Let all the pins are of same signal id and hence they are connected. Once the routing phase is over, a designer is often interested to find (a) if there exist two pins which should have been connected but are not (b) if two pins of different signal ids are connected to the same trunk and (c) no two trunks connected to pins of different signal ids are connected by a dogleg. While working with a layout editor, a designer often zooms into a particular section of the layout to find errors within his/her window of interest. In this work we design data structures using which a designer can efficiently decide the presence of any of the three conditions within the query of interest.

2 Assumptions

In an abstract sense, each pin can be considered as a node of a graph. Two nodes of the graph share an undirected edge if they are directly connected by wires. Assume a, b, c to be the nodes of a graph where a, b share an edge and b, c share an edge. We consider that the nodes a, b, c all are connected as a signal originating from the pin a can reach c through b. See Figure 2 (1).

Each graph is like a connected component. It can be noticed that if two pins are not connected, they will belong to two different connected components. Each connected component can be assigned a unique id. In our work, we assume that (a) each pin is given to us as a point in plane, its net id as *sid* and the id of the

^{*}International Institute of Information Technology , Hyderabad, India, anandaswarup@gmail.com

[†]Heritage Institute of Technology, Kolkata, India , prosenjit_ gupta@acm.org

[‡]International Institute of Information Technology , Hyderabad, India, srinathan@iiit.ac.in



Figure 2: (1) The black nodes are the pins. The pins a, b, c are connected, so are the pins d, e, f. (2) G_1 is the graph with the nodes a, b, c and G_2 is the graph corresponding to d, e, f.

connected component to which it belongs as the gid, (b) each trunk is given as a horizontal segment and is assigned a color which is equal to the sid of one of the pins it is connected to, (c) each branch is given as a vertical segment and is assigned a color which is equal to the sid of the only pin it is connected to and (d) each dogleg is given as a vertical segment and is assigned a color which is equal to the sid of one of the two trunks it is connected to.

3 Problem Definitions

In the rest of the paper, we refer to the following condition as *Condition 1:*

Condition 1: $sid(p_1) = sid(p_2)$, but $gid(p_1) \neq gid(p_2)$.

In this work, we study the following two problems

Problem 3.1 We are given a set S of n points in \mathbb{R}^2 . Each point $p \in S$ has two colors, namely, sid(p) and gid(p), associated with it along with its coordinates. We need to preprocess them into a data structure such that given a query rectangle $q = [a, b] \times [c, d]$ we can decide if there exist two points $(p_1, p_2) \in S \cap q$ such that $sid(p_1) = sid(p_2)$ but $gid(p_1) \neq gid(p_2)$.

Problem 3.2 Let H and V be respectively the sets of horizontal and vertical segments in \mathbb{R}^2 . Each horizontal segment $h \in H$ (resp. $v \in V$) has a color namely, sid(h) (resp. sid(v)) associated with it. The sid(h) (resp sid(v)) is not necessarily unique. We need to preprocess H and V into a data structure such that given a query rectangle $q = [a, b] \times [c, d]$, we can efficiently decide if there exists a pair of horizontal-vertical segments (h, v) such that $h \cap v \cap q \neq \emptyset$ and sid $(h) \neq sid(v)$.

4 Solutions for the Problem 3.1

4.1 Solution 1: A Simple Idea

4.1.1 Preprocessing

We divide the points of the set S into subsets S_1, \ldots, S_k where the subset S_i contains the points with *sid* equal to *i*. Next, we sort the points in S_i according to their gids. For every pair of points $p, m \in S_i$, we create a 4-d point (p_x, p_y, m_x, m_y) if $gid(p) \neq gid(m)$ and store it into a data structure D for range searching in \mathbb{R}^4 .

4.1.2 Query Algorithm

Given a query rectangle $q = [a, b] \times [c, d]$, we search the data structure D with the query $q' = [a, b] \times [c, d] \times$ $[a, b] \times [c, d]$. If we find any point in q', we return "YES", else we return "No".

Lemma 1 Using the data structure of [8] for range searching in \mathbb{R}^4 , a data structure of size $O(n^2(\frac{\log n}{\log \log n})^3)$ can be constructed such that given a query rectangle q we can decide in $O(\frac{\log^2 n}{\log \log n})$ time, if there is any instance of Condition (1) inside q.

4.2 Solution 2: Improving the Storage Space While Trading-off Query Time

4.2.1 Preprocessing:

For each point $p \in S$ we create two points namely p_1 and p_2 . We set the coordinates of both the points to (p_x, p_y) . We then color p_1 with the color sid(p). The point p_2 is colored with a composite color which uniquely represents the chromatic pair < sid(p), gid(p) >. We store the points p_1 in a set S_L and the points p_2 in S_B . We preprocess the points in S_L and S_B into two data structures D_L and D_B respectively. D_L and D_B are instances of generalized two-dimensional orthogonal range counting.

4.2.2 Query Algorithm:

Given a query rectangle q, we first find the distinct colors of the points of the set S_L that are present in q. This is done by querying D_L with q. Let the number of distinct colors of the points of the set S_L present in qbe n_L . Next, we find the distinct colors of the points of the set S_B that are present in q by searching the data structure D_B . We call this value as n_B . If $n_L < n_B$, we return "YES". Else, we return "No".

Lemma 2 There exists an instance of Condition (1) inside the query rectangle iff $n_L < n_B$.

Hence we have the following result

Lemma 3 There exists a data structure ([1]) of size $O(n^2 \log^2 n)$ such that given a query rectangle q we can decide in $O(\log^2 n)$ time, if there is any instance of Condition (1) inside q. A space-time trade of data structure ([3]) with a space bound of $O((n/r)^2 \log^6 n + n \log^4 n)$ and a query time of $O(r \log^7 n)$ such that $1 \le r \le n$ is also possible.

4.3 Solution 3: Further Improving the Storage Space



Figure 3: The arrow marks indicate that each of the rays are extending towards ∞ . The right ray will be allocated to all the nodes marked black. The down ray will be allocated to the nodes marked 1, 2, 3 and 4

In this section, we propose a solution which needs $O(n^2)$ storage space. Using the data structure, we can solve the Problem 3.1 in $O(\log^2 n)$ time.

4.3.1 Preprocessing Stage $1 \rightarrow$ Assignment of rays:

- 1. Consider the point set S. From each point $p \in S$, we shoot four rays namely two horizontal rays, one traveling $-\infty$, the other traveling $+\infty$ and two vertical rays one traveling $-\infty$, the other traveling $+\infty$.
- 2. We call the horizontal rays traveling towards $-\infty$ as *left rays*, horizontal rays traveling towards $+\infty$ as *right rays*, vertical rays traveling towards $+\infty$ as *up rays* and the vertical rays traveling $-\infty$ as *down rays*. See Figure 3.
- 3. Let us sort the points of the set S in terms of their x coordinates. Construct a balanced binary search

tree T_x whose leaf nodes corresponds to the elementary intervals being induced by the x coordinates of the points of the set S.

- 4. Each internal node $\mu \in T_x$ stores an interval $Int(\mu)$ which is union of the elementary intervals being stored in the leaf nodes of the subtree rooted at μ .
- 5. Now consider all the right rays, up rays and the down rays. To the node $\mu \in T_x$, we allocate the right ray emanating from the point p if $Int(\mu) \cap [p_x, \infty) \neq \emptyset$ where p_x is the x coordinate of the point p. Refer to Figure 3.
- 6. For each up ray (resp. down ray), we first find the leaf node storing the x coordinate of the up ray or the down ray. Then, starting from the leaf node τ , we allocate it to all the ancestors of the leaf node.

4.3.2 Preprocessing Stage 2 \rightarrow Classification of rays at the node w :



Segments allocated to node w

Figure 4: In this figure, seg 1 is a right ray completely covering Int(w). seg 2 is a right ray partially overlapping with Int(w). seg 3 is an up ray and seg 4 us a down ray.

- 1. Consider a node $w \in T_x$ and a ray assigned to the node w. Let the x coordinate of the end point of the ray be p_x . With the concerned ray, the following are the possibilities:
 - Int(w) ⊆ [p_x,∞) that is the interval of node w is completely contained in the semi-infinite interval [p_x,∞), the horizontal projection of the ray.
 - $Int(w) \cap [p_x, \infty) \neq \emptyset$, that is the interval Int(w) is not completely contained in $[p_x, \infty)$ but they are overlapping.

- The ray is an up ray or a down ray.
- 2. Let $L_{w,F}$ is the list of rays whose horizontal projection completely contains Int(w). Let $L_{w,P}$ is the list of rays whose horizontal projections are partially overlapping with Int(w). Let $L_{verti,w}$ be the list of up and down rays allocated to w or in other words whose x coordinates are stored in the leaf nodes of the subtree rooted at w.

See Figure 4. In that figure, the horizontal ray denoted by seg 1 belongs to $L_{w,F}$, seg 2 belongs to $L_{w,P}$. The up ray and the down ray will belong to $L_{verti.w}$.

4.3.3 Preprocessing Stage 3 \rightarrow Creation of 2-d and 3-d points:

- 1. Consider any horizontal ray $h \in L_{w,F}$. Let the coordinates of the end point of the h be (h_x, h_y) .
- 2. Check if there is any vertical ray $v \in L_{verti,w}$ such that sid(h) = sid(v) but $gid(h) \neq gid(v)$.
- 3. Let there be some vertical rays v. We will denote the coordinates of the end points of v as (v_x, v_y) .
 - Among all the down rays (respectively up rays) select the one whose v_y is just above (respectively just below) h_y . Let that v_y be denoted as $v_{y,1}$ for the down ray and $v_{y,2}$ for the up ray.
 - We create two 3-d points $(h_x, h_y, v_{y,1}),$ $(h_x, v_{y,2}, h_y).$
- 4. Similarly, for each horizontal ray $h \in L_{w,P}$ which is partially overlapping with Int(w),
 - check if there is any vertical ray $v \in L_{verti,w}$ such that sid(h) = sid(v) but $gid(h) \neq gid(v)$ and $h \cap v \neq \emptyset$.
 - Among all the down rays (respectively up rays) select the one whose v_y is just above (respectively just below) h_y . We will denote that as $v_{y,1}$ for the down ray and $v_{y,2}$ for the up ray.
 - We create two 2-d points $(h_y, v_{y,1}), (v_{y,2}, h_y)$.

See Figure 4. In that figure, the seg 1 will contribute to 3-d points and the seg 2 will contribute to 2-d points.

4.3.4 Preprocessing Stage 4 \rightarrow Storing the 2-d and 3-d points:

- 1. We store the 3-d points in a 3-d dominance reporting data structure $D_{3,w}$ of [5].
- 2. The 2-d points are stored in a priority search tree $T_{PST,w}$ [7].

Lemma 4 The storage space needed by the above data structure is $O(n^2)$.

4.3.5 Query Algorithm:



Figure 5: The segment [a, b] of the query rectangle $q = [a, b] \times [c, d]$ is allocated to the nodes marked black. The scenario at a node w to whom [a, b] is allocated.

- 1. Given a query rectangle $q = [a, b] \times [c, d]$ we first allocate the segment [a, b] to the nodes of T_x . The rule that we follow for allocating [a, b] to the node μ is $Int(\mu) \subseteq [a, b]$ but $Int(parent(\mu)) \not\subseteq [a, b]$. It should be noted that the way the segment [a, b] is allocated is not the same as the way we allocate the horizontal rays. It should also be mentioned that the segment [a, b] will be allocated to $O(\log n)$ nodes of the tree T_x .
- 2. Let S_{can} be the set of $O(\log n)$ canonical nodes to which the segment [a, b] is allocated. At each node $w \in S_{can}$ first search the priority search tree $T_{PST,w}$ with the query $[c, \infty) \times (-\infty, d]$.
 - If we find a point in $T_{PST,w} \cap [c, \infty) \times (-\infty, d]$, we return "YES".
 - Else we search the 3-d dominance data structure $D_{3,w}$ with the query $[a,\infty) \times [c,\infty) \times (-\infty,d]$. If we find a point in $D_{3,w} \cap [a,\infty) \times [c,\infty) \times (-\infty,d]$, we return "YES".

Lemma 5 The query time of the above algorithm is $O(\log^2 n)$.

We therefore summarize the results of the Problem 3.1 with the following theorem

Theorem 6 Given a set S of n points in \mathbb{R}^2 such that each point $p = (p_x, p_y)$ has two associated colors gid(p), sid(p) associated with it,

1. There exists a data structure of size $O(n^2(\frac{\log n}{\log \log n})^3)$ can be constructed such that given a query rectangle q we can decide in $O(\frac{\log^2 n}{\log \log n})$ time, $sid(p_1) = sid(p_2)$ but $gid(p_1) \neq gid(p_2)$.

- 2. There exists a data structure of size $O(n^2 \log^2 n)$ such that given query rectangle q, in $O(\log^2 n)$ time we can decide if there exist two points p_1, p_2 in q such that $sid(p_1) = sid(p_2)$ but $gid(p_1) \neq gid(p_2)$.
- 3. A space-time trade off data structure with storage bound $O((n/r)^2 \log^6 n + n \log^4 n)$ and query time $O(r \log^7 n)$ is also possible. Here r is a user defined parameter.
- 4. We also have a data structure with storage space requirement of $O(n^2)$ and query time $O(\log^2 n)$ to answer the same query.

5 Solution for the Problem 3.2

Consider a horizontal vertical segment intersection inside a query rectangle. There are two possible scenarios

(a) at least one end point of either of the segments is inside the query rectangle. We call this kind of intersections as intersections of type 1.

(b) Both the end points of both the segments are outside q that is the segments completely cross the query rectangle. We call this kind of intersections as intersections of type 2.

For dealing with the first situation, we consider the case where the lower end point of the vertical segment is inside the query rectangle. Similar arrangements have to be done for the upper end point.

5.1 Preprocessing a data structure

5.1.1 Preprocessing Phase $1 {\rightarrow}$ Creation of 2-d points:

Consider the lower end point p'' of a vertical segment v. Let the y coordinate of p'' be v_y . We find the horizontal segment h whose y projection is just above v_y and $sid(v) \neq sid(h)$. Let the y projection of h be h_y . We create a 2-d point (v_y, h_y) . The step has to be repeated for all the vertical segments of the set V.

5.1.2 Preprocessing Phase $2 \rightarrow$ Constructing a Segment tree:

Let M' be the sorted list of the x coordinates of the end points of the horizontal and vertical segments in H and V respectively. We construct a segment tree T_x whose leaf nodes correspond to the elementary intervals induced by the x coordinates of the set M'. Each internal node $\mu \in T_x$ stores an interval $Int(\mu)$ which is union of the elementary intervals being stored in the leaf nodes of the subtree rooted at μ . A horizontal segment $h \in H$ is allocated to a node $\mu \in T_x$ if $Int(\mu)$ is completely contained in the horizontal projection of the h whereas $Int(parent(\mu))$ is not. For each vertical segment v we locate the leaf node in T_x which stores the x coordinate of v. Then starting from that leaf node, we store a copy of v in all the ancestors of the leaf node including the root of T_x .

Consider a node $\mu \in T_x$. Let $L_{H,\mu}$ and $L_{V,\mu}$ be the set of horizontal and vertical segments allocated to the node μ . At the node μ , we do the following:

- 1. Consider a vertical segment $v \in L_{V,\mu}$ and consider the point (v_y, h_y) we have created in phase 1. We store the point in a priority search tree $T_{\mu,1}$. This has to be done for all the vertical segments in $L_{V,\mu}$.
- 2. Next, for each $v \in L_{V,\mu}$ whose y projection is $[v_{y_1}, v_{y_2}]$, we create a 2-d point (v_{y_1}, v_{y_2}) . We store these points in a priority search tree $T_{\mu,2}$.
- 3. Finally, for each $v \in L_{V,\mu}$, we create a 3-d point $(v_{y_1}, v_{y_2}, sid(v))$. We store these points in a 3-d dominance reporting data structure $T_{\mu,3}$ of [5]. The $T_{\mu,3}$ will support queries of the form $[x, \infty) \times (-\infty, y] \times [z, \infty)$ and $[x, \infty) \times (-\infty, y] \times (-\infty, z]$.

5.1.4 Preprocessing Phase $4 \rightarrow$ Range Minima data structure:

Consider any node $\mu \in T_x$. Consider the horizontal segments $h \in L_{H,\mu}$. We store the *y* coordinates of these horizontal segments in an array $Y_{sort,\mu}$ in a sorted order. Next, we create a 1-d range minima data structure RM_{μ} [6] such that given two indices of the array $Y_{sort,\mu}$, we can decide in O(1) time if all the horizontal segments whose *y* coordinates are stored in between the two query indices in the array $Y_{sort,\mu}$ have the same *sid*.

5.2 Query Algorithm

Given a query rectangle $[a, b] \times [c, d]$, allocate the segment [a, b] to a node μ of the segment tree T_x if $Int(\mu) \subseteq [a, b]$ but $Int(parent(\mu)) \notin [a, b]$. The segment [a, b] will be allocated to $O(\log n)$ nodes of the tree T_x .

5.2.1 Intersections of type 1

Let S_{can} be the set of such nodes. Let us first focus on the intersections of type 1 that is when at least one of the end points of the intersecting segments are inside the query rectangle. We will explain our steps assuming that our focus is on the lower end points of the vertical segments.

1. Any vertical segment that intersects the query rectangle q is present in any of the nodes $u \in S_{can}$. We therefore search the priority search tree $T_{u,1} \ \forall u \in S_{can}$ with the query $q' = [c, \infty) \times (-\infty, d]$.

2. While searching $T_{u,1}$ with q', if we find any point p in q', we return "Yes" and Exit.

5.2.2 Intersections of type 2



Figure 6: (a) All the line segments except the dotted horizontal segment are of same *sid*. (b) All the horizontal segments are of same *sid*. The dotted vertical segment is of different *sid*.

When the end points of the intersecting horizontal and vertical segments are outside the query rectangle, we do the following:

- 1. At each node $\mu \in S_{can}$ and the ancestors of the node μ up to root in the tree T_x , we search the array Y_{sort} at the respective nodes to find the the indices i, j such that $c \leq Y_{sort}[i] < Y_{sort}[j] \leq d$.
- 2. By using the range minima data structure RM at that respective node, we then decide if all the horizontal segments whose y projections are in between the indices i and j have the same sid.
- 3. If we find that all the horizontal segments allocated to the node μ or its ancestors are not of same *sid*, we come back to the node μ and search the data structure $T_{2,\mu}$ with the query $(-\infty, c] \times [d, \infty)$. If there is any point inside $(-\infty, c] \times [d, \infty)$, we return "YES" and Exit. See Figure 6 (a).
- 4. Suppose all the horizontal segments allocated to the node μ or its ancestors are of same *sid*, we then find that particular *sid*. Next, We come back to the node μ and search $T_{\mu,3}$ with $(-\infty, c] \times [d, \infty) \times [sid+1, \infty)$ and $(-\infty, c] \times [d, \infty) \times (-\infty, sid-1]$. If we find any point in either of the queries, we return "YES" and Exit. See Figure 6 (b).

Lemma 7 The algorithm returns a "YES" iff there exist a pair of horizontal-vertical segments h,v, such that $h \cap v \cap q \neq \emptyset$ and $sid(v) \neq sid(h)$.

Theorem 8 There exists a data structure D of size $O(n \log n)$ such that given a query axes parallel rectangle q, we can decide in $O(\log^2 n)$ time if there exists a pair (h, v) where h is a horizontal segment and v is a vertical segment such that h intersects v inside q and $sid(h) \neq sid(v)$.

References

- P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In Journal of Algorithms, 19, pp. 282–317, 1995.
- [2] A. Agrawal, P. Gupta. Incremental Analysis of Large VLSI Layouts. In Integrations, 42(2), 205– 210, 2009.
- [3] H. Kaplan, N. Rubin, M. Sharir, E. Verbin. Counting colors in boxes. In Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 785–794, 2008.
- [4] B. Chazelle, H. Edelsbrunner, L. J. Guibas and M. Sharir. Algorithms for Bichromatic Line Segment Problems and Polyhedral Terrains. In Algorithmica, 11, pp. 116–132, Springer Verlag, 1994.
- [5] P. Afshani. On Dominance Reporting in 3D Proceedings of 16thEuropean Symposium on Algorithms (ESA), pp. 41–51, 2008.
- [6] H. Yuan, M. Atallah. Data Structures for Range Minimum Queries in Multidimensional Arrays. In Proceedings of SODA, pp. 150–160, 2010.
- [7] E. M. McCreight. Priority Search Trees. In SIAM Journal of Computing. vol. 14(2), pp. 257–276, 1985.
- [8] P. Afshani, L. Arge, K. D. Larsen Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements In Proceedings of Symposium on Computational Geometry, pp. 240–246, 2010.
- [9] www.bwrc.eecs.berkeley.edu/Classes/icbook /magic/index.html

Finding The Maximum Density Axes Parallel Regions for Weighted Point Sets

Ananda Swarup Das *

Prosenjit Gupta[†]

Kannan Srinathan[‡]

Kishore Kothapalli[§]

Abstract

In this work we study the problem of finding axesparallel regions of maximum density for weighted point sets in \mathbb{R}^2 and \mathbb{R}^3 . The 2-d variant is motivated by applications in thermal analysis of VLSI chips.

1 Introduction

We are given a set of n weighted points in \mathbb{R}^2 (respectively in \mathbb{R}^3). Our goal is to find the highest density axes parallel rectangle in \mathbb{R}^2 (in \mathbb{R}^3 we find the highest density axes parallel 3-dimensional box) where density of a region is defined as the sum of weights per unit area (respectively per unit volume). The problem for finding the highest density axes parallel rectangle for \mathbb{R}^2 was introduced by Majumder et al. in [1] and has applications in thermal analysis of VLSI chip.

Preliminaries and Problems: Let us first define the term density formally. The definition that we are providing here has been mentioned in [1].

Definition 1 Let F be a rectangular floor containing a set S of n points such that no two points lie on the same horizontal or vertical line. Each point $p_i \in S$ is associated with a positive real weight w_i . The density of an axes parallel region R with area A(R) is defined as $\frac{\sum_{p_i \in R} w_i}{A(R)}.$

It should be mentioned that in case of unweighted points (or when points are of equal weight), the density is $\frac{|S \cap R|}{A(R)}$. Next, we introduce the main problem that we study in this work.

Problem 1.1 Given a set S of n points in a plane such that each point has a positive weight associated with it, find the cluster of $k \geq 2$ points in S such that the minimum area axes parallel rectangle covering them attains the highest density.

Our Contribution:

In this work we first show that if the coordinates of the *n* points are integers in $[0, U] \times [0, U]$, and the points have distinct positive weights w(p), then finding the highest density axes parallel rectangle can be done in $O(n \log n \log U)$ time using $O(n \log^2 n)$ storage. We then present another data structure which solves Problem 1.1 in $O(n \log n \log U)$ time using $O(n \frac{\log^2 n}{\log \log n})$ storage. We finally discuss how to find the highest density axes parallel 3-dimensional box provided the coordinates of the n weighted points are integers in $[0, U] \times [0, U] \times [0, U].$

Special note: We assume that all the coordinates of the points are distinct. This particular assumption of distinctness is important for the correctness of the solution as because, if there are two points whose x coordinates (or y coordinates) are the same, the area of the axes parallel rectangle induced by them is zero and hence the corresponding density will be ∞ .

2 Solution

In [1], Majumder et al. proved the following,

Lemma 1 Let each point $p_i \in S$ be associated with a positive weight w_i and there exists a cluster $S' \subseteq S$ of k > 2 points such that no two of them lie in the same horizontal (vertical) line. Then there exists a pair $p_i, p_j \in S'$ such that the density of the smallest area axes parallel rectangle containing (p_i, p_j) is greater than the density of the axes parallel rectangle containing S'.

In short, the Lemma 1 says the following: let the cluster S' contains k > 2 points and let the density D' of S' is the sum of the weights of the k points in S' divided by the smallest area of the axes parallel rectangle bounding all the points of S'. Also assume that S' is the cluster of highest density among all possible clusters for the points of S. Then as a contradiction, it can be shown that there exist two points $p_i, p_i \in S'$ such that the density of the smallest area axes parallel rectangle containing (p_i, p_j) is greater than the density of the axes parallel rectangle containing S'. It therefore means that the maximum density occurs for a cluster $C \subseteq S$ containing only two points. Though Majumder et al. proved the Lemma 1, they solved the unweighted

^{*}International Institute of Information Technology, Hyderabad, India, anandaswarup@gmail.com

[†]Heritage Institute of Technology, Kolkata, India prosenjit_gupta@acm.org

[‡]International Institute of Information Technology, Hyderabad, India, srinathan@iiit.ac.in

[§]International Institute of Information Technology, Hyderabad, India, kkishore@iiit.ac.in

version of the problem in which case, the problem gets reduced to finding the smallest area axes parallel rectangle enclosing two points of S as diagonally opposite corners. No efficient solution is however known for the problem with a weighted point set. In this work, we show a simple "divide and conquer" technique for the problem assuming that the coordinates of the weighted points are integers in $[0, U] \times [0, U]$.



Figure 1: (a) The y coordinate of the end point of the dotted semi infinite horizontal segment is $p_{y_mid} = \frac{p_1(y)+p_t(y)}{2}$. (b) The rectangle enclosing the points p_1, p_t as diagonally opposite corner points cannot be a candidate for highest density rectangle as it contains the point p_{adv} in it.

2.1 Our Algorithm

- 1. Consider a point $p_1 \in S$. Let $p_1 = (p_1(x), p_1(y))$. Consider the northeast quadrant $NE(p) = (P_1(x), \infty] \times (P_1(y), \infty]$.
- 2. Let the weight of p_1 be $w(p_1)$. Create a query box $q = [p_1(x), \infty) \times [p_1(y), \infty) \times [w(p_1), \infty)$.
- 3. Find the point with the smallest x coordinate in the query box q. Let this point be denoted by $p_t = (p_t(x), p_t(y))$.
- 4. Consider the axes parallel rectangle $R_{1,t}$ enclosing the points p_1, p_t as the diagonally opposite corners. Check if the rectangle $R_{1,t}$ contains any point $p_{adv} \in S$. See Figure 1 (b).
 - (a) If $R_{1,t} \cap S = \emptyset$, then $R_{1,t}$ is a candidate for being the highest density rectangle.
 - (b) Else, as per Lemma 1, $R_{1,t}$ cannot be the highest density rectangle.
- 5. Repeat the above steps but this time with the query box $q = [p_1(x), \infty) \times [p_1(y), \lfloor \frac{p_1(y)+p_t(y)}{2} \rfloor] \times [w(p_1), \infty)$. The reason for this step is explained in Lemma 2.

6. Stop the algorithm if
$$\lfloor \frac{p_1(y)+p_t(y)}{2} \rfloor = p_1(y)$$
.

- 7. Follow a similar procedure for the southeast quadrant $SE(p_1)$, southwest quadrant $SW(p_1)$, and northwest quadrant $NW(p_1)$ for the point p_1 .
- 8. Repeat the steps (1) to (7) for all the points in S.

Consider the query box $q = [p_1(x), \infty) \times [p_1(y), \infty) \times [w(p_1), \infty)$ mentioned in the step (2) of the algorithm and let $S_{NE(p_1)}$ be the set of all the points in S lying in the northeast quadrant of p_1 such that these points have their respective weight greater than the weight of p_1 . Then we have the following lemma.

Lemma 2 Let $p_t, p_2 \in S_{NE(p_1)}$ be two points such that (a) $p_t = (p_t(x), p_t(y))$ has the smallest x coordinate among all points in $S_{NE(p_1)}$ and (b) the axes parallel rectangle $R_{1,2}$ enclosing p_1 and p_2 as diagonally opposite corners has the highest density in $S_{NE(p_1)}$. Then the y coordinate of p_2 must be less than $\frac{p_1(y)+p_t(y)}{2}$.

Proof: See Figure 1 (a). Since $p_2(x) > p_t(x) > p_1(x)$, $p_2(x) - p_t(x) < p_2(x) - p_1(x)$. By Lemma 1, $p_2(y) < p_t(y)$ or else the rectangle $R_{1,2}$ will also contain the point p_t and hence cannot have highest density. If $p_2(y) \in [\frac{p_1(y)+p_t(y)}{2}, p_t(y)]$, then $p_t(y) - p_2(y) \leq p_2(y) - p_1(y)$. Therefore the area of the rectangle $R_{t,2}$, the one enclosing p_t, p_2 as the diagonally opposite points will have its area $A(R_{t,2}) < A(R_{1,2})$. Since $w(p_t) > w(p_1)$, $\frac{w(p_t)+w(p_2)}{A(R_{t,2})} > \frac{w(P_1)+w(P_2)}{A(R_{1,2})}$, a contradiction to the fact that $R_{1,2}$ has the highest density. \Box

We therefore conclude the section by stating the following lemma.

Lemma 3 When the coordinates of the points are integers and in the range of $[0, U] \times [0, U]$, the maximum number of candidate rectangles we generate for any point p_1 is $O(\log U)$. The total number of candidate rectangles thus generated is $O(n \log U)$.

3 The Choice of Data Structures

As evident from our algorithm in Section 2.1, we need two particular data structures namely (a) a 2-d range aggregate data structure D such that given a query rectangle q we can efficiently decide if $q \cap D = \emptyset$ or not, and (b) a 3-d range successor data structure for efficient execution of our algorithm. We skip the discussion on 2-d range queries as they are very well studied [8] and focus on 3-d range successor problem. Formally the problem can be defined as follows.

Problem 3.1 Given a set S of n points in \mathbb{R}^3 preprocess them into a data structure such that given an axes parallel d-box q for d = 3, one can efficiently report the point with the smallest x coordinate in $q \cap S$.

The above problem has been studied in [3, 7]. The data structure presented in [7] takesexpected $O(n \log n \log \log n)$ preprocessing time, occupies $O(n \log^2 n)$ space and can be queried in $O(\log n \log \log n)$ time. The data structure presented in [3] can be built in $O(n^{1+\epsilon})$ time, occupies $O(n^{1+\epsilon})$ space and can be queried in O(1) time. For any data structure for the range successor problem let P(n) be the preprocessing time and let Q(n) be the query time. Since we may need to answer $O(n \log U)$ range successor queries in the worst case for solving Problem 1.1, the total time required to answer range successor queries is $R(n) = P(n) + O(n \log U)Q(n)$. Hence we propose a data structure RSQ to solve the 3-dimensional range successor problem so that the R(n) value is smaller than that obtained by using the solutions from [3] and [7]. RSQ is a variant of range aggregate tree with fractional cascading [8] and is also a variant of [4].

3.1 The Preprocessing Algorithm

- 1. Let x_1, x_2, \ldots, x_n be a sorted list of points on the real line, being the x-projections of the points in S. Consider the elementary intervals created by the partitioning of the real line induced by these points. Construct a balanced binary tree T_x , associating the above elementary intervals with its leaves.
- 2. To each internal node μ , assign an interval $int(\mu)$ which is union of the elementary intervals of the points associated with the leaf nodes of the subtree rooted at μ .
- 3. At each internal node $\mu \in T_x$ maintain an array A_{μ} which stores the *y* coordinates of the points present in the leaf nodes of the subtree rooted at μ .
- 4. Also maintain a range minima data structure $RM_{A_{\mu}}$ (see [6]) such that given two indices i, j of A_{μ} , we can return the maximum weight among the points whose y coordinates are stored between $A_{\mu}[i]$ to $A_{\mu}[j]$.
- 5. Let w and v be the two children of μ . Since $A_{\mu} = A_w \cup A_v$, each index i of A_{μ} has two pointers one pointing to the smallest value in A_w which is greater than equal to $A_{\mu}[i]$ and the other pointing to the smallest value in A_v greater than equal to $A_{\mu}[i]$. Similarly each index i of A_{μ} has two pointers one pointing to the largest value in A_w which is smaller than equal to $A_{\mu}[i]$ and the other pointing to the largest value in A_v smaller than equal to $A_{\mu}[i]$.
- 6. Now, at the node μ , construct a height balanced binary search tree $T_{\mu,y}$ on the points of A_{μ} .

- 7. At each node $\phi \in T_{\mu,y}$, store a sorted array W_{ϕ} which stores the weights of the points whose y coordinates are stored in the leaf nodes of subtree rooted at ϕ .
- 8. Maintain a range minima data structure RM'_{ϕ} such that given two indices i, j of W_{ϕ} , we can return the minimum x coordinate among the points stored between $W_{\phi}[i]$ to $W_{\phi}[j]$.

Lemma 4 The data structure RSQ for range successor queries can be built in $O(n \log^2 n)$ time and occupies $O(n \log^2 n)$ space.

3.2 The Query Algorithm

Let our query be $q = [x_1, \infty) \times [y_1, y_2] \times [wt, \infty)$. We wish to find out the point $p \in S$ with the smallest x coordinate and fitting inside q.



Figure 2: The nodes marked black are the ones to which the interval $[x_1, \infty)$ is allocated

- 1. Find the leaf node $leaf \in T_x$ such that it contains the value x_1 . Trace the path π from the node leafto the root node of the tree T_x .
- 2. For any node v which is a right sibling of any node u on the path π , its interval $int(v) \subset [x_1, \infty)$. Allocate the semi-infinite interval $[x_1, \infty)$ to the nodes v. See Figure 2. The nodes marked black are the ones to which the interval $[x_1, \infty)$ is allocated. In general, the interval $[x_1, \infty)$ will be allocated to $O(\log n)$ canonical nodes of T_x . Since $[x_1, \infty)$ is a semi-infinite interval, it will be allocated to at most one node at each level of the tree. For $l \in O(\log n)$, let us number these nodes as v_1, \ldots, v_l , starting from the leaf level of the tree T_x . See Figure 2.
- 3. Search the array A_{root} to find the indices i and j such that $y_1 \leq A_{root}[i] < A_{root}[j] \leq y_2$. Then find the smallest value greater than y_1 and the largest value smaller than y_2 in the all the arrays starting from A_1, \ldots, A_l . This can be done by chasing the pointers starting from A_{root} .

- 4. As v_1 is a leaf node, $|A_{v_1}| = 1$. Check if the y coordinate stored in the node marked 1 is in between y_1 and y_2 . If so, check if the weight of the point is greater than wt. If so, we have our desired point in the node 1.
- 5. Else we move to the node marked 2. Let i', j' be the indices such that $y_1 \leq A_2[i'] < y_2 \leq A_2[j']$.
- 6. Using Range Minima data structure RM_{A_2} , find the maximum weight w' among the points stored in between $A_2[i']$ and $A_2[j']$.
- 7. Let w' > wt. It means we have our desired point at the node 2. We move to the node 2.
 - Consider the data structure $T_{\mu,y}$ for $\mu = 2$, created in steps (6) to (8) of the preprocessing step. By repeating steps similar to (4) to (7) of the query algorithm on the tree $T_{\mu,y}$, on required auxiliary arrays W_{ϕ} and on required range minima data structures RM'_{ϕ} , we can find out the point with the smallest x coordinate present in $S \cap q$.
 - Return the point discovered in the previous step
- 8. On the other hand, if w' < wt, we move to the next node that we have marked as node 3.
- 9. For any query q, the above steps continue until
 - we discover the point with the smallest x coordinate in q or
 - we have visited all the l nodes and have failed to find the point $p \in S$ with the smallest xcoordinate and fitting in q.

Lemma 5 The data structure RSQ supports 3dimensional range successor queries in $O(\log n)$ time.

Proof: Finding the leaf node $leaf \in T_x$ needs $O(\log n)$ time. Next, searching the indices i, j such that $y_1 \leq$ $A_{root}[i] < A_{root}[j] \le y_2$, needs another $O(\log n)$ time. Once the indices i, j of the root node are found, finding the respective indices in all the arrays of the nodes marked black in Figure 2 needs $O(\log n)$ time. Next, we check if the point in the node marked 1 fits our requirement. If so, our job is done in $O(\log n)$ time. Else, we move to the node marked 2 and find the maximum weight among the points which are stored between $A_2[i']$ to $A_2[j']$ where $y_1 \leq A_2[i'] < A_2[j'] \leq y_2$. This needs O(1) time using the range minima data structure. If we find the maximum weight to be greater than wt, we restrict our searching only at node 2. Next, we repeat similar searches at the tree $T_{2,y}$. This needs another $O(\log n)$ time. Hence the result. \Box

From Lemma 4 and Lemma 5, we conclude the following theorem.

Theorem 6 A set S of n points in \mathbb{R}^3 can be preprocessed in time $O(n \log^2 n)$ into a data structure of size $O(n \log^2 n)$ so that given a query axes-parallel rectangle q, the range successor query can be answered in $O(\log n)$ time.

By using Lemma 3 and Theorem 6, we can conclude the following:

Lemma 7 If the points are in the range $[0, U] \times [0, U]$, $O(\log U)$ candidate rectangles for the point P_1 are generated in $O(\log U \log n)$ time. Hence the total time needed to find the highest density axes parallel rectangle is $O(n \log U \log n + n \log^2 n)$.

The above lemma will also hold when the coordinates of the points are integers in $\mathbb{R} \times [0, U]$.

3.3 A Reduced Spaced Data Structure

Next we present RSL, a reduced space data structure for the 3-dimensional range successor problem. **Steps of Preprocessing :**

- 1. Let us change the degree of the internal nodes of the tree T_x to $O(\sqrt{\log n})$ instead of two. The height of the tree is therefore $O(\frac{\log n}{\log \log n})$.
- 2. In each internal node $\mu \in T_x$, we create the auxiliary array A_{μ} . The array A_{μ} stores the sorted ycoordinates of the points whose x coordinates are associated in the leaf nodes.
- 3. Any element of A_{μ} will now have with 2 pointers pointing to two elements in each of the auxiliary arrays belonging to the $\sqrt{\log n}$ children of μ . One pointer will point to the smallest value in the array A_w greater than the element and the other pointer will point to the largest element in the array A_w smaller than the element. Here w is a child of μ . Hence any element in A_{μ} will have $2\sqrt{\log n}$ pointers. The construction of A_{μ} is discussed later.
- 4. Repeat the steps (3), (6), (7), (8) of the preprocessing algorithm in sub section 3.1.

3.3.1 Construction of the array A_{μ}

Building the array A_{μ} is easy. Let $A_{w_1}, \ldots, A_{w_{\sqrt{\log n}}}$ are the sorted arrays present at the $\sqrt{\log n}$ children of the node μ . From each array $A_{w_i} \forall i = 1, \ldots, \sqrt{\log n}$, take its smallest element and construct a min-heap. The height of the heap will be $O(\log \log n)$. Now the root node of the heap will contain the smallest element of the heap. We will store the element of the root to the first available index of A_{μ} and this element will have a pointers to the elements currently present in the heap at



Figure 3: The nodes marked black are the ones to which the interval $[x_1, \infty)$ is allocated

their respective arrays. It will also have a pointer to the next value of the array A_{w_i} from which it originated. From the array A_{w_i} take the next element and insert it into the heap. This method has been used in [5] for reporting the top k weights in a query rectangle.

Lemma 8 The data structure RSL for range successor queries can be built in $O(n \frac{\log^2 n}{\log \log n})$ time and occupies $O(n \frac{\log^2 n}{\log \log n})$ space.

3.4 The Query Procedure

Let our query $q = [x_1, \infty) \times [y_1, y_2] \times [wt, \infty)$ and we would like to find the point with the smallest x coordinate in it. Let us denote the tree T_x as the primary tree. This tree is a variant of the range tree used by [2] and [3]. As mentioned in [2], an interval [a, b] can be represented as a union of node ranges of some nodes v_1, \ldots, v_k that can be grouped into $\frac{\log n}{\log \log n}$ groups G_1, \ldots, G_h . Each group G_i contains a set of children $v_{l_i}, v_{l_{i+1}}, \ldots, v_{l_r}$ for some node v_l . There are at most two groups in each level. Hence there are $O(\frac{\log n}{\log \log n})$ groups. Consider the interval $[x_1, \infty)$. We can also write this interval as $[x_1, x_{max}]$ where x_{max} is the maximum x coordinate among all the points of the set S. See Figure 3. Let $[x_1, x_{max}]$ is equal to the union of the intervals of the black nodes. These black nodes are divided into three groups and are assigned to the nodes marked G_1, G_2 and G_3 .

Let p_1 be the point with the smallest x coordinate in q. Let us also suppose that p_1 is not present in the nodes marked by the groups G_1 and G_2 . Now suppose we are at the node marked G_3 . We need to decide, among the three children of G_3 marked as n_1, n_2, n_3 , which node should we visit? It can be noticed that if there is even a single point from the node marked n_1 fitting inside the query q, the point p_1 has to be present in n_1 . Therefore, we need a way to decide if it is profitable to visit the node n_1 (or in fact any node). Remember that we have an array A_{G_3} at the node marked G_3 such that A_{G_3} sorts all the points that are stored at the leaf nodes of the subtree rooted at G_3 . The array A_{G_3} is sorted according to the y coordinates of the points that are present in the leaf nodes of the tree rooted at node marked G_3 . Moreover each index *i* of the array A_{G_3} has a pointer to the smallest value greater than $A_{G_3}[i]$ and the largest value smaller than $A_{G_3}[i]$ in the array A_{n_1} . So if we find the smallest value greater than y and the largest value smaller than y_1 for the array A_{G_3} , then we can easily do the same for A_{n_1} . Once we discover the two indices of the array A_{n_1} , using range minima query we can find the largest weight w' for the points whose y coordinates are stored in the array A_{n_1} in between those two indices in O(1) time. If w' > wt, we focus our searching only at the node n_1 . We move to the node n_1 and follow steps similar to that of the query algorithm in subsection 3.2 to find out the point with the smallest x coordinate present in $S \cap q$ and return it as an answer. On the other hand, if wt < w', we move to the next node $(n_2 \text{ as per our Figure 3}).$

Lemma 9 The data structure RSL supports 3dimensional range successor queries in $O(\log n)$ time.

Theorem 10 A set S of n points in \mathbb{R}^3 can be preprocessed in time $O(n \frac{\log^2 n}{\log \log n})$ into a data structure of size $O(n \frac{\log^2 n}{\log \log n})$ so that given a query axes-parallel rectangle q, the range successor query can be answered in $O(\log n)$ time.

Using the above theorem, we conclude the following

Theorem 11 Given a set of n weighted points whose coordinates are integers in $[0, U] \times [0, U]$, the highest density axes parallel rectangle can be found in time $O(n \frac{\log^2 n}{\log \log n} + n \log U \log n)$ using space of $O(n \frac{\log^2 n}{\log \log n})$.

4 On Finding the Highest Density Axes Parallel 3dimensional box

Before we start the section, we define our density as follows:

Definition 2 Let S be a set of n points in \mathbb{R}^3 . The density of the axes parallel d-box R (for d = 3) with volume V(R) and covering the points in R, is defined as $\frac{\sum_{p_i \in \mathbb{R}} w_i}{V(R)}$.

In this section, we wish to solve the following problem

Problem 4.1 We are given a set of n weighted points such that their coordinates are integers in the range of $[0, U] \times [0, U] \times [0, U]$. We wish to find the cluster of $k \ge 2$ points such that the minimum area axes parallel dbox for d = 3 covering them attains the highest density.

Before we try to solve the Problem 4.1, we refer to the following fact proved by Majumder et al. in [1].

Fact 1 Let $Q = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}$ for $a_i, b_i > 0$, then $Min_{i=1}^{n} \frac{a_i}{b_i} \le Q \le Max_{i=1}^{n} \frac{a_i}{b_i}$

Now, assuming that all the coordinates of the points are distinct, we propose the following lemma



Figure 4: The case of axes parallel 3-dimensional box for d = 3

Lemma 12 The highest density axes parallel 3dimensional box will consist of two points.

Proof: The proof is similar to the proof of Lemma 1 provided in [1]. See Figure 4. Let our highest density axes parallel 3- dimensional box contains k points p_1, p_2, \ldots, p_k . We partition the box into k-1 smaller pieces as shown in the Figure 4. The density of our box is $\frac{wt(p_1)+wt(p_2)+\ldots+wt(p_k)}{V(R)} \leq \frac{wt(p_1)+wt(p_2)}{V(R_1)} + \frac{wt(p_2)+wt(p_3)}{V(R_2)} + \ldots + \frac{wt(p_{k-1})+wt(p_k)}{V(R_{k-1})}$, where $V(R_i)$ denotes the volume of the rectangular axis parallel d-box R_i . Using Fact 1, we can prove the lemma. \Box

The highest density axes parallel 3-dimensional box will contain two points of the set S as diagonally opposite corners. Using the Lemma 12 and the ideas of Section 2.1 and data structure similar to 3.1, we have the following theorem.

Theorem 13 Given a set of n weighted points whose coordinates are integers in range of $[0, U] \times [0, U] \times [0, U]$, the highest density axes parallel 3-dimensional box can be found in $O(n \log U \log^3 n)$ times using $O(n \log^3 n)$ space.

References

- S. Majumder, B. B. Bhattacharya, On the density and discrepancy of a 2D point set with applications to thermal analysis of VLSI chips. Information Processing Letters 107 (2008), pp. 177–182.
- [2] Y. Nekrich, Orthogonal Range Searching in Linear and Almost-Linear Space. Computational Geometry: Theory and Applications 42(4) (2009), pp. 342–351.
- [3] C. C. Yu, W. K. Hon, B. F. Wang, Improved Data Structures for Orthogonal Range Successor

Queries. Computational Geometry: Theory and Applications **44** (2011), pp. 148–159.

- [4] S. Saxena, Dominance made simple. Information Processing Letters 109 (2009), pp. 419–421.
- [5] S. Rahul, P. Gupta, R. Janardan K. S. Rajan, *Efficient top-k queries for orthogonal ranges*, In Proc. International Workshop on Algorithms and Computation, Springer Verlag Lecture Notes in Computer Science No. 6552, pp. 110–121.
- [6] H. Yuan, M. Atallah, Data Structures for Range Minimum Queries in Multidimensional Arrays. In Proceedings of SODA 2010, pp. 150–160.
- H. P. Lenhof, M. H. M. Smid, Using Persistence for adding range restrictions to searching problems. RAIRO Theoretical Informatics and Applications. 28(1) (1994), pp. 25–49.
- [8] M. de. Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, Verlag, 2000.

Bichromatic Line Segment Intersection Counting in $O(n\sqrt{\log n})$ Time

Timothy M. Chan*

Bryan T. Wilkinson[†]

Abstract

We give an algorithm for bichromatic line segment intersection counting that runs in $O(n\sqrt{\log n})$ time under the word RAM model via a reduction to dynamic predecessor search, offline point location, and offline dynamic ranking. This algorithm is the first to solve bichromatic line segment intersection counting in $o(n \log n)$ time.

1 Introduction

We consider bichromatic line segment intersection *counting*: given a set of disjoint blue line segments and a set of disjoint red line segments in the plane, output the number of intersections of blue segments with red segments. Bichromatic line segment intersection problems arise in applications such as map overlay in GIS. We give an algorithm that runs in $O(n\sqrt{\log n})$ time, where n is the total number of segments, under the standard word RAM model with *w*-bit words. The input segments are specified by their endpoints, which are given as O(w)bit integer or rational coordinates. Our result answers an open question of Chan and Pătrașcu [6] by showing that bichromatic line segment intersection counting can be solved in $o(n \log n)$ time. Thus, the problem finally joins the ranks of many other problems in computational geometry that have $\Omega(n \log n)$ lower bounds under the comparison model but $o(n \log n)$ upper bounds under the word RAM model. Recent examples include 2-d Voronoi diagrams [2], 3-d convex hulls [6], and 3-d layers-of-maxima [12]. The word RAM model is important because its power corresponds very closely to that of actual programming languages (our algorithm uses only standard operations, such as arithmetic and bitwise logical operations) and it includes only the reasonable assumption that each input value is an integer (or rational) that fits in a word.

Chazelle et al. [7] give algorithms based on segment trees for bichromatic segment intersection reporting in $O(n \log n + k)$ time, where k is the number of intersections reported, and for counting in $O(n \log n)$ time. Chan's trapezoid sweep algorithm [3] for the reporting problem also achieves $O(n \log n + k)$ time, but has smaller constant factors and can be extended to curve segments. Mantler and Snoeyink [11] modify the trapezoid sweep to use operations of algebraic degree at most 2 and also to support counting in $O(n \log n)$ time.

Our algorithm follows the high-level idea of Mantler and Snoeyink [11] but reduces the low-level computations to dynamic predecessor search, offline point location, and offline dynamic ranking. Note that we cannot base similar reductions on just any high-level algorithm; the algorithm of Mantler and Snoeyink [11] gives particularly exploitable structure to the bichromatic line segment intersection counting problem. The algorithm of Chazelle et al. [7] inherently requires $O(n \log n)$ time due to the use of segment trees, and Chan's trapezoid sweep does not address the counting problem.

Recently, efficient algorithms have arisen for both offline point location and offline dynamic ranking under the word RAM model. Chan and Pătraşcu [6] give an algorithm for offline point location that locates O(n)points in a subdivision of the plane defined by O(n) segments in $n \cdot 2^{O(\sqrt{\log \log n})}$ time. Chan and Pătraşcu [5] also give an algorithm for offline dynamic ranking that processes O(n) queries and updates in $O(n\sqrt{\log n})$ time. We use both of these algorithms as subroutines.

In Section 2 we give an overview of Mantler and Snoeyink's algorithm [11] and describe a 1-d data structure problem to which it reduces. In Section 3 we discuss a rank space reduction which is integral to achieving speed ups under the word RAM model. In Section 4 we describe our data structure, which we analyse in Sections 5 and 6.

2 High-Level Algorithm

We assume that the endpoints of all of the given segments are in general position. Otherwise, perturbation techniques can be used to handle any degeneracies [10]. We begin with an overview of the algorithm of Mantler and Snoeyink [11], simplified under our non-degeneracy assumption. The algorithm is a plane sweep that keeps track of all of the segments that intersect the vertical *sweep line*. The efficiency of the algorithm is achieved by storing the segments in maximal monochromatic *bundles* of segments. The algorithm keeps an alternating list of red and blue bundles. The order of the bundles in this list may not be consistent with the order of the segments along the sweep line. However, the order of the

^{*}David R. Cheriton School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca

[†]David R. Cheriton School of Computer Science, University of Waterloo, b3wilkin@uwaterloo.ca. Supported by NSERC.

bundles of a single colour in the list is always consistent with the order of segments of the same colour along the sweep line. Figure 1 shows the grouping of segments into bundles at various positions of an example plane sweep.



Figure 1: Ordering of red (dashed) and blue (solid) bundles before and after processing endpoints e_1 and e_2 .

When the plane sweep reaches an endpoint e, our goal is to swap red and blue bundles until all blue bundles below e are below all red bundles above e, and all blue bundles above e are above all red bundles below e. When we swap a red bundle b_r and a blue bundle b_b it is because all segments of b_r intersect with all segments of b_b . So, whenever we perform a swap, we add $|b_r| \cdot |b_b|$ to our bichromatic segment intersection counter.

First, we find the red bundles immediately above and below e. If e is inside a red bundle, we split this bundle into two bundles such that one is above e and the other is below e. If b_r is the red bundle above e, we check if the blue bundle b_b that follows b_r in our list of bundles is below e. If so, we swap b_r and b_b . Doing so may result in two adjacent red bundles and/or two adjacent blue bundles. We merge these adjacent bundles of the same colour. We repeat this process until b_b is not below e. In the last repetition, e may be inside b_b , in which case we split b_b around e before swapping. We follow a similar process in the other direction.

After all of the bundle swapping has occurred, we still need to handle the inclusion or exclusion of the segment with endpoint e. If e is a left endpoint, we insert the segment into a new bundle which is placed between the lowest bundle above e and the highest bundle below e. If e is a right endpoint, we delete the segment. If the segment is the only segment in its bundle, deleting the segment removes its bundle. In either the insertion or deletion case, we merge adjacent bundles of the same colour as necessary.

For a proof of correctness, we refer the reader to Mantler and Snoeyink's paper [11]. The main idea of the proof is that the order of the bundles is always consistent with the order of a certain deformation of the segments along the sweep line. This deformation pushes intersections as far to the right as possible without moving endpoints, adding intersections, or removing intersections.

The low-level computations of the algorithm can be encapsulated into a data structure that supports the following operations:

Insert (s, b_{ℓ}, b_{h})

Inserts a new bundle containing only segment s between bundles b_{ℓ} and b_h .

Delete(s)

Deletes segment s, removing its bundle b if s is the only segment in b.

IsAbove(e, b) / IsBelow(e, b)

Determines whether or not endpoint e lies above or below all segments in bundle b.

$\mathbf{Split}(e, b)$

Splits bundle b into two bundles b_{ℓ} and b_h such that b_{ℓ} contains all segments below endpoint e and b_h contains all segments above endpoint e. Figure 2 shows an example of a bundle being split.



Figure 2: Splitting bundle b at endpoint e.

$Merge(b_{\ell}, b_h)$

Merges two adjacent bundles b_{ℓ} and b_h of the same colour into a single bundle.

Swap (b_{ℓ}, b_h)

Swaps the order of two adjacent bundles b_{ℓ} and b_h of different colours.

HighestBelow(e) / LowestAbove(e)

Finds the highest (lowest) red bundle in which all segments are below (above) endpoint e.

Next(b) / Previous(b)

Finds the bundle that follows or precedes bundle b in sorted order.

Size(b)

Calculates the number of segments in bundle b.

The analysis presented by Mantler and Snoeyink [11] reveals that each of these operations is invoked at most O(n) times. The main idea of their analysis is that during the processing of each endpoint, there is a constant upper bound on the number of bundle splits, and thus there are a linear number of bundles over the course of the algorithm. All of the other operations can be charged to bundles.

3 Rank Space Reduction

Before we describe our data structure, we discuss a preprocessing step in which we perform rank space reduction. Rank space reduction is important under the word RAM model to, for example, reduce the cost of predecessor search with van Emde Boas trees [14] from $O(\log \log U)$, where U is the size of the universe, to $O(\log \log n)$. In particular, we want to assign *ids* to segments such that segment s_1 's id is greater than segment s_2 's id if and only if s_1 is above s_2 . Since red and blue segments can intersect, there may be no mapping that is consistent for all segments throughout the entire plane sweep. However, Palazzi and Snoeyink [13] give a topological ordering that is consistent for a set of disjoint segments. We can thus assign ids that are consistent with aboveness to all red segments and all blue segments separately, based on their own separate topological orderings. The topological sort involves a plane sweep that runs in $O(n \log n)$ time, since it uses a balanced search tree in order to find the segment above the current endpoint.

Instead of using this balanced search tree, we can find the segment above every endpoint in advance by computing the trapezoidal decomposition of the segments. Chan and Pătraşcu [4] reduce general offline 2-d point location to offline 2-d point location in a vertical slab that is divided into regions by disjoint segments that cut across the slab. We call this latter problem the *slab problem*. The same techniques (persistence and exponential search trees) can be used to reduce trapezoidal decomposition of disjoint segments to the slab problem. Chan and Pătraşcu [6] give an algorithm for the slab problem that yields an algorithm for trapezoidal decomposition of disjoint segments that runs in $n \cdot 2^{O(\sqrt{\log \log n})}$ deterministic time. The topological sort thus has the same runtime.

4 Data Structure

All segments have ids, as assigned by the rank space reduction described in Section 3. We keep a doubly linked list of bundles. A bundle stores pointers to its top and bottom segments. Only segments which are

top or bottom segments of a bundle have pointers back to their bundle. In addition to this doubly linked list, we keep two predecessor search data structures that use segment ids as keys. These trees, T and B, contain only those red segments that are at the tops and bottoms of their bundles, respectively. If we were to use van Emde Boas trees [14], updates would run in $O(\log \log U)$ expected time. Since we want to avoid randomization, we use instead a data structure of Andersson and Thorup [1], which supports queries and updates in $O(\log \log n \frac{\log \log U}{\log \log \log U})$ deterministic time. Due to the rank space reduction, these operations actually run in $O(\frac{\log^2 \log n}{\log \log \log n})$ time. Finally, we keep dynamic ranking data structures R_r and R_b , also using segment ids as keys, for all red segments and blue segments, respectively, that intersect the sweep line. We defer our selection of a particular dynamic ranking data structure to Section 5.

Lemma 1 After a preprocessing step that runs in $n \cdot 2^{O(\sqrt{\log \log n})}$ time, we can find the segment of a given colour above or below endpoint e along the sweep line in O(1) time.

Proof. Assume without loss of generality that e is red. The red segments above and below e can be found in O(1) time by navigating the red trapezoidal decomposition that was computed in $n \cdot 2^{O(\sqrt{\log \log n})}$ time in Section 3. Finding the blue segments above and below e is equivalent to locating e within the blue trapezoidal decomposition. So, in the preprocessing step, we perform offline planar point location of all red endpoints in the blue trapezoidal decomposition. The blue trapezoidal decomposition has linear complexity and has vertices with O(w)-bit rational coordinates. We can perform offline planar point location of the red endpoints in such a subdivision of the plane in $n \cdot 2^{O(\sqrt{\log \log n})}$ time using another algorithm of Chan and Pătraşcu [6].

We now describe how we implement all of the operations of the data structure, using the ability to find the segments above and below an endpoint in constant time via Lemma 1 as a primitive.

Insert (s, b_{ℓ}, b_{h})

We create a new bundle b in which s is both the top and bottom segment and rewire the next/previous bundle pointers between b, b_{ℓ} , and b_h . We insert sinto R_c , where c is the colour of s. If s is red, we also insert it into both T and B.

Delete(s)

If s is both the top and bottom segment of its bundle b, s has a pointer to b. We rewire the next/previous bundle pointers around b to exclude b. If s is only the top (bottom) segment of b, we can find the new boundary of b by finding the segment of the same colour below (above) the endpoint of s on the sweep line. If s is red, we delete s from T and/or B, as necessary. In any case, we delete s from R_c , where c is the colour of s.

IsAbove(e, b) / IsBelow(e, b)

We check in constant time whether or not e is above (below) the top (bottom) segment of b. The endpoint e must then be above (below) all other segments of b.

$\mathbf{Split}(e, b)$

We find the segments of b's colour above and below e. The segment below e becomes the top segment of b, which we relabel to b_{ℓ} . The original top segment of b becomes the top segment of a new bundle b_h . The bottom segment of b_h is the segment above e. We rewire the next/previous bundle pointers to include b_h after b_{ℓ} . If b was red, we add the segment below e to T and the segment above eto B.

$Merge(b_{\ell}, b_h)$

If b_{ℓ} and b_h are red, we remove the top segment of b_{ℓ} from T and the bottom segment of b_h from B. We replace the top segment of b_{ℓ} with the top segment of b_h and rewire the next/previous bundle pointers to exclude b_h .

$\mathbf{Swap}(b_\ell, b_h)$

We rewire next/previous bundle pointers to swap the order of b_{ℓ} and b_h .

HighestBelow(e) / LowestAbove(e)

We find the red segment immediately below e and find its predecessor in T. The bundle of the resulting top segment is the highest bundle below e. A similar process finds the lowest bundle above e.

Next(b) / Previous(b)

We follow b's next or previous bundle pointer.

Size(b)

We query R_c , where c is the colour of b, for the ranks of the top and bottom segments of b. We obtain the size of b by subtracting the latter from the former and adding 1.

5 Handling Dynamic Ranking

Dynamic ranking can be solved using Dietz's data structure [8], which supports both queries and updates in $O(\frac{\log n}{\log \log n})$ time. The high-level algorithm described in Section 2 must determine the sizes of at most O(n) bundles. Also, each endpoint causes a single insertion or deletion from a rank query data structure. Thus, if we were to use Dietz's data structure, dynamic ranking would contribute $O(n \frac{\log n}{\log \log n})$ time to the runtime of our algorithm. However, by considering the purpose of our rank queries, it turns out that we can do better.

We use dynamic ranking data structures to determine the sizes of bundles. The high-level algorithm requires these sizes for a single purpose: to calculate the total number of bichromatic intersections between a red bundle and a blue bundle that intersect. It is important to note that the results of these calculations have no effect on future decisions of the algorithm. In fact, the results only affect the algorithm's output value. Another way to calculate the same output value is to perform all of the rank queries and updates offline at the end of the algorithm. Offline dynamic ranking can be solved faster than its online counterpart. Specifically, Chan and Pătraşcu [5] give an algorithm for offline dynamic ranking that handles O(n) queries and updates in $O(n\sqrt{\log n})$ time.

6 Analysis

The plane sweep requires that all endpoints are sorted by their x-coordinate, which can be performed in $O(n \log \log n)$ deterministic time [9]. The rank space reduction of Section 3 is performed in $n \cdot 2^{O(\sqrt{\log \log n})}$ time. The preprocessing step of Lemma 1 also runs in $n \cdot 2^{O(\sqrt{\log \log n})}$ time. The high-level algorithm invokes each operation of our data structure at most O(n) times. All operations consist of a constant number of pointer assignments, queries and updates to dynamic predecessor search data structures, and queries and updates to dynamic ranking data structures. The queries and updates to the dynamic predecessor search data structures contribute at most $O(n \frac{\log^2 \log n}{\log \log \log n})$ deterministic time to the runtime of the algorithm, using Andersson and Thorup's data structure [1]. As discussed in Section 5, the queries and updates to the dynamic ranking data structures can be handled offline in $O(n\sqrt{\log n})$ time. This final contribution to the algorithm's runtime dominates all others; thus, the algorithm as a whole runs in $O(n\sqrt{\log n})$ time.

References

- [1] A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. J. ACM, 54, June 2007.
- [2] K. Buchin and W. Mulzer. Delaunay triangulations in O(sort(n)) time and more. J. ACM, 58:6:1–6:27, April 2011.
- [3] T. M. Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In In Proc. 6th Canad. Conf. Comput. Geom, pages 263–268, 1994.
- [4] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.*, 39:703–729, July 2009.

- [5] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10, pages 161–173, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.
- [6] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, II: Offline search. CoRR, abs/1010.1948, 2010. Also in Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, STOC '07, pages 31–39, New York, NY, USA, 2007. ACM.
- [7] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994. 10.1007/BF01182771.
- [8] P. F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proceedings of the Workshop on Algorithms and Data Structures*, WADS '89, pages 39–46, London, UK, 1989. Springer-Verlag.
- [9] Y. Han. Deterministic sorting in O(n log log n) time and linear space. In Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, STOC '02, pages 602–608, New York, NY, USA, 2002. ACM.
- [10] H. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In *Theoretical Foundations of Computer Graphics and CAD*, volume 40 of *Proceedings of the NATO Advanced Science Institute, Series F*, pages 307–326. Springer-Verlag, 1988.
- [11] A. Mantler and J. Snoeyink. Intersecting red and blue line segments in optimal time and precision. In J. Akiyama, M. Kano, and M. Urabe, editors, *Discrete* and Computational Geometry, volume 2098 of Lecture Notes in Computer Science, pages 244–251. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-47738-1_23.
- [12] Y. Nekrich. A fast algorithm for three-dimensional layers of maxima problem. CoRR, abs/1007.1593, 2010.
- [13] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. CVGIP: Graph. Models Image Process., 56:304–310, July 1994.
- [14] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In Proceedings of the 16th Annual Symposium on Foundations of Computer Science, pages 75–84, Washington, DC, USA, 1975. IEEE Computer Society.

Sequential Dependency Computation via Geometric Data Structures

Gruia Calinescu *

Howard Karloff[†]

Abstract

We are given integers $0 \leq G_1 \leq G_2 \neq 0$ and a sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$ of N integers. The goal is to compute the minimum number of insertions and deletions necessary to transform S_N into a valid sequence, where a sequence is *valid* if it is nonempty, all elements are integers, and all the differences between consecutive elements are between G_1 and G_2 . For this problem from the database theory literature, previous dynamic programming algorithms have running times $O(N^2)$ and $O(A \cdot N \log N)$, for a parameter A unrelated to N. We use a geometric data structure to obtain a $O(N \log N \log \log N)$ running time.

1 Introduction

Golab, Karloff, Korn, Saha, and Srivastava introduce the following problem in VLDB 2009 [3]: We are given integers $0 \leq G_1 \leq G_2 \neq 0$ and a (not necessarily sorted) sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$ of N integers. The goal is to compute the minimum number of insertions and deletions necessary to transform S_N into a valid sequence, where a sequence is *valid* if it is nonempty, all elements are integers, and all the differences between consecutive elements are between G_1 and G_2 . That is, $\langle b_1, b_2, ..., b_M \rangle$ is valid if $M \geq 1$ and for all $i \in \{1, \ldots, M-1\}, G_1 \leq b_{i+1} - b_i \leq G_2$. We term the problem GAP DEPENDENCY.

An example instance of GAP DEPENDENCY and its solution has $G_1 = 4$, $G_2 = 6$, and $\langle 1,7,5,9,12,25,31,30,34,40 \rangle$ as the (invalid) input sequence. A feasible solution deletes the first five elements and the seventh element, resulting in the valid sequence $\langle 25,30,34,40 \rangle$, at cost 6. A better feasible solution, of cost 5, starts by deleting 12 and inserting 15 and 20 in its place, obtaining the sequence $\langle 1,7,5,9,15,20,25,31,30,34,40 \rangle$, which is still not valid since 5 - 7 < 4 and 30 - 31 < 4. After deleting 7 and 31, we obtain the valid sequence $\langle 1,5,9,15,20,25,30,34,40 \rangle$. Yet another solution of cost 5 deletes 5,9,31 (resulting in sequence $\langle 1,7,12,25,30,34,40 \rangle$, which is invalid since 25 - 12 > 6), followed by inserting 16 and 20 between 12 and 25.

Golab et al. [3] present an algorithm with running time $O(\frac{G_2}{G_2-G_1}N\log N)$ for $G_2 > G_1 > 0$ (and $O(N\log N)$ if $G_1 = 0$ or $G_1 = G_2$). This is pseudopolynomial running time. Implicit in [3] is also a $O(N^2)$ -time algorithm. In this paper we give a $O(N\log N\log\log N)$ -time algorithm for $G_2 > G_1 > 0$, by exploiting a surprising connection to geometric data structures.

2 Preliminaries

We include definitions and results from [3]. Given a sequence $S_N = \langle a_1, a_2, ..., a_N \rangle$, define S_i to be the prefix $S_i = \langle a_1, a_2, ..., a_i \rangle$, and OPT(i) to be the value of the GAP DEPENDENCY optimum with input S_i .

Given a sequence $\langle a_1, a_2, ..., a_N \rangle$ of integers, for i = 1, 2, ..., N, let $v = a_i$ and define T(i) to be the minimum number of insertions and deletions one must make to $\langle a_1, a_2, ..., a_i \rangle$ in order to convert it into a valid sequence ending in the number v.

Computing OPT(N) from the T(i)'s can be done as follows. $OPT(N) = \min_{0 \le r \le N-1} \{r + T(N - r)\}$, as proven in Claim 1.

Claim 1 [Claim 3 of [3]] The minimum number OPT(i) of insertions and deletions required to convert sequence S_i into a valid one is given by $\min_{0 \le r \le i-1} \{r + T(i-r)\}$. Furthermore, OPT(i) can be calculated inductively by OPT(1) = 0 and $OPT(i) = \min\{1 + OPT(i-1), T(i)\}$ for all $i \ge 2$.

In order to show how to compute the T(i)'s, we need the following definition from [3]:

Definition 1 Define dcost(d), for d = 0, 1, 2, ..., to be the minimum number of integers one must append to the length-1 sequence $\langle 0 \rangle$ to get a valid sequence ending in d, and ∞ if no such sequence exists.

For example, if $G_1 = 4$ and $G_2 = 6$, then $dcost(7) = \infty$. Furthermore, dcost(8) = 2, uniquely obtained by appending 4 and 8. We compute dcost very differently. Precisely, we use existing geometric data structures. Instead of this lemma:

Lemma 1 (Lemma 6 of [3]) If $G_1 = 0$, then $dcost(d) = \lceil d/G_2 \rceil$. Otherwise, $dcost(d) = \lceil d/G_2 \rceil$ if $\lceil (d+1)/G_1 \rceil > \lceil d/G_2 \rceil$ and ∞ otherwise,

^{*}Department of Computer Science, Illinois Institute of Technology; calinescu@iit.edu. Research supported in part by NSF grant CCF-0515088.

 $^{^{\}dagger}AT\&T$ Labs-Research, 180 Park Ave., Room C231, Florham Park, NJ 07932, USA; howard@research.att.com.

we use the method of the following section. We do so since the previous dynamic programs [3] may use the lemma for $\Omega(\min\{N^2, \frac{G_2}{G_2-G_1}N\log N\})$ values of d, even though *dcost* can be computed in constant time.

The $O(N^2)$ algorithm of [3] follows in a rather straightforward way from Claim 1, the lemma above, and Theorem 2 which appears later. We refer to [3] for the more sophisticated $O(\frac{G_2}{G_2-G_1}N\log N)$ algorithm.

3 The new algorithm for computing the T(i)-values

In this paper we will assume that $0 < G_1 < G_2$.

What differentiates this paper from [3] is the use of a fast geometric data structure to calculate the T(i)'s, in amortized time $O(\log N \log \log N)$ each. We show how the recurrence used in [3] can be modified to make use of a data structure allowing fast 2-dimensional range minimum queries, and thereby to decrease the running time from $O(\min\{N^2, \frac{G_2}{G_2-G_1} \cdot N \log N\})$ to $O(N \log N \log \log N)$. (This is only an improvement, of course, if $\frac{G_2}{G_2-G_1} > \log \log N$.)

We assume all the values a_i are nonnegative. (Otherwise, let $m = \min_i a_i$ and set $a_i := a_i - m$.) For each j, create point $P_j = (x_j, y_j)$ with $x_j = a_j \mod G_2$ and $y_j = \lfloor a_j/G_2 \rfloor$. Two values of j can have points P_j with the same coordinates; we treat the points P_j as distinct. Let $\Delta := G_2 - G_1 > 0$.

For given i, define two regions in the two dimensional Euclidean plane as follows (see Figure 1 for an example). Let $q_i(x)$ be the linear map

$$q_i(x) = y_i - (x - (x_i - G_1))/\Delta$$

and let Q_i be the halfspace

$$Q_i = \{(x, y) : y \le q_i(x)\}.$$

Let $r_i(x)$ be the linear map

$$r_i(x) = y_i - (x - x_i)/\Delta$$

and let R_i be the intersection of the halfspaces

$$\{(x,y)|y \le r_i(x)\}$$

and

$$\{(x,y)|x \ge x_i\}$$

and last, let $R_i^* = R_i \setminus \{(x_i, y_i)\}.$

(It will be crucial later that all the lines $r_i(x)$, over all *i*, and all lines $q_i(x)$, over all *i*, have the same slope. These facts will allow us to find *one* affine transformation converting, for all *i*, Q_i into a halfspace with *axis-parallel* bounding line, and R_i into an intersection of two halfspaces, whose bounding lines are orthogonal *axis-parallel* lines.)

Our algorithm relies on the following theorem from [3].



Figure 1: Here $G_2 = 10$, $G_1 = 7$, $a_i = 35$. The point P_i is given by the small dark circle. R_i and Q_i are unbounded and we only show their relevant parts—where other points P_j could be located. R_i is the region on the right, colored using a diagonal pattern. Q_i is the region on the left, colored using a doubly diagonal pattern. Where the regions intersect, we use a solid pattern.

Theorem 2 [3] Fix $i \ge 2$. Assume $G_1 > 0$. Define $m := \min_{j < i, a_j < a_i} \{T(j) + (i-1-j) + [dcost(a_i-a_j)-1]\}$. Then $T(i) = \min\{i-1,m\}$.

For intuition only, we explain the recurrence. To end an optimal subsequence with a_i , we either delete the first i-1 elements, or, with j being such that j < iand $a_j < a_i$, take the optimal subsequence ending with a_j , delete the i-1-j elements between a_j and a_i , and insert $dcost(a_i - a_j) - 1$ elements between a_j and a_i . (The "-1" is here since, as defined, dcost(d) also inserts "d", while we do not have to insert " a_i ".)

We will prove the following theorem by relating it to Theorem 2.

Theorem 3 Fix $i \ge 2$. Define

$$r := \min_{j : j < i, P_j \in R_i^*} \{ T(j) + (i - j - 1) + (y_i - y_j) - 1 \}$$

and

$$q := \min_{j : j < i, P_j \in Q_i} \{T(j) + (i - j - 1) + (y_i - y_j)\}.$$

Then $m = \min\{q, r\}.$

To prove Theorem 3, we need Claim 2. Recall that the x-coordinate of each P_k is at most $G_2 - 1$.

- **Claim 2** 1. For any k < i, $[a_k < a_i \text{ and } dcost(a_i a_k) < \infty]$ if and only if $P_k \in Q_i \cup R_i^*$.
- 2. If $P_k \in R_i^*$, then $a_k < a_i$ and $dcost(a_i a_k) = y_i y_k$.
- 3. If $P_k \in Q_i \setminus R_i$, then $a_k < a_i$ and $dcost(a_i a_k) = y_i y_k + 1$.

We will prove Claim 2 in a moment. **Proof of Theorem 3**. We need to prove that

$$\min\{q, r\} = \min_{j < i, a_j < a_i} \{T(j) + (i - 1 - j) + [dcost(a_i - a_j) - 1]\}.$$

By part 1 of Claim 2, the two minima are infinite on exactly the same set. Using this and the fact that $P_i \notin Q_i$ (because $q_i(x_i) = y_i - G_1/\Delta$ and $G_1 > 0$ so that $y_i > q_i(x_i)$),

$$m = \min_{j < i, P_j \in Q_i \cup R_i^*} [T(j) + (i - 1 - j) + dcost(a_i - a_j) - 1]$$

= min { min_{j < i, P_j \in R_i^*} [T(j) + (i - 1 - j) + dcost(a_i - a_j) - 1],
min_{j < i, P_j \in Q_i \setminus R_i} [T(j) + (i - 1 - j) + dcost(a_i - a_j) - 1],
min_{j < i, P_i \in Q_i \cap R_i} [T(j) + (i - 1 - j) + dcost(a_i - a_j) - 1] \}.

Now we use parts 2 and 3 of Claim 2 and the fact that $P_i \notin Q_i$ to infer that m equals

$$\min\{\min_{j < i, P_j \in R_i^*} [T(j) + (i - 1 - j) + y_i - y_j - 1],$$
$$\min_{j < i, P_j \in Q_i \setminus R_i} [T(j) + (i - 1 - j) + y_i - y_j],$$
$$\min_{j < i, P_j \in Q_i \cap R_i} [T(j) + (i - 1 - j) + y_i - y_j - 1]\}.$$

Letting

$$A := \min_{j < i, P_j \in R_i^*} [T(j) + (i - 1 - j) + y_i - y_j - 1],$$
$$B := \min_{j < i, P_j \in Q_i \setminus R_i} [T(j) + (i - 1 - j) + y_i - y_j],$$

and

$$C := \min_{j < i, P_j \in Q_i \cap R_i} [T(j) + (i - 1 - j) + y_i - y_j - 1],$$

we want to show that $\min\{A, B, C\} = \min\{q, r\}$. Since r = A and $q = \min\{B, C+1\}$, $\min\{q, r\} = \min\{A, \min\{B, C+1\}\} = \min\{A, B, C+1\}$. We want to show that $\min\{A, B, C\} = \min\{A, B, C+1\}$, which follows from the fact that $A \leq C$. \Box

Sketch of proof of Claim 2. Note that $a_i = x_i + y_i G_2$ and $a_k = x_k + y_k G_2$.

Let $I_k = [kG_1, kG_2]$ for $k \ge 0$. It is easy to see that $I_k \cap \mathbb{Z}$ is precisely the set of all integers which can be written as the sum of exactly k integers all between G_1 and G_2 . Then dcost(d) is the minimum k such that $d \in I_k$, if one exists, and ∞ otherwise. In other words, here is a way to compute dcost(d) for all d, in principle:

Algorithm **Simpledcost**:

• Set $dcost(d) = \infty$ for all $d \ge 0$.

- For k = 0, 1, 2, ..., do:
 - Set dcost(d) = k for all $d \in I_k$, unless dcost(d)was already defined.

We will show that the three statements in the claim are obtained in effect by "running" algorithm **Simpledcost** above.

Label the lattice points $0, 1, 2, ..., a_i$, starting by labeling the point $P_i = (x_i, y_i)$ "0", and then moving leftward, labeling points with successive integers, until a point (0, y) on the y-axis is reached, and (after labeling that point) continuing with point $(G_2 - 1, y - 1)$. The point labeled " a_i " will be the origin (0, 0), since the top row has $x_i + 1$ labeled points, and each of the other y_i rows has G_2 points, or $1 + a_i$ points in total, as desired.

For all $y \in \{0, 1, 2, ..., y_i\}$, the point (x_i, y) is labeled $(y_i - y)G_2$, which is the right endpoint of interval I_{y_i-y} . Now execute the following:

For $l = 0, 1, 2, ..., y_i + 1$, do:

• Starting at point $(x_i, y_i - l)$ and continuing for $|I_l| = l \cdot (G_2 - G_1)$ additional steps, move rightward by one lattice point each time;

however, if a point $(G_2 - 1, y)$ is hit, then after visiting that point, visit the point (0, y+1) next and afterward continue proceeding rightward as before. (Every visited point (x, y) has $y \ge -1$.)

• Assign *dcost* equal to *l* for each point visited, unless its *dcost* was already assigned or its second coordinate was negative.

The points with nonnegative second coordinate visited during iteration l are exactly those whose labels are in I_l , so we are in effect executing algorithm **Simpledcost**. In other words, the existence of a point with nonnegative second coordinate with label l and assigned dcost d means that dcost(l) = d, and the existence of such a point with label l and no dcost means that $dcost(l) = \infty$.

(As an example, look at Figure 1. $I_0 = [0]$ and only (5,3) is assigned dcost 0. $I_1 = [7,10]$ and the lattice points with dcost 1 are (5,2), (6,2), (7,2), (8,2). $I_2 = [14,20]$ and the lattice points with dcost = 2 are (5,1), (6,1), (7,1), (8,1), (9,1), (0,2), (1,2). $I_3 = [21,30]$ and the lattice points with dcost = 3 are (5,0), (6,0), (7,0), (8,0), (9,0), (0,1), (1,1), (2,1), (3,1), (4,1). $I_4 = [28,40]$ and the lattice points with dcost 4 are (0,0), (1,0), (2,0), (3,0), (4,0).)

The following crucial statements are easy to verify. All the points assigned a finite *dcost* are in $Q_i \cup R_i$, and all such points P_k in the nonnegative quadrant get a finite *dcost*. If $P_k \in R_i^*$, then $a_k < a_i$, since $r_i(x)$ has negative slope. If $P_k \in Q_i \setminus R_i$, then, since $q_i(0) = y_i + (x_i - G_1)/\Delta \le y_i + [(G_2 - 1) - G_1]/\Delta = y_i + (\Delta - 1)/\Delta < y_i + 1$, all $P_j \in Q_i$ have $y_j \le y_i$ and hence $a_j < a_i$. Because we assign dcost equal to l for points in row $y_i - l$ in R_i , as well as some to the left in Q_i in row $y_i - l + 1$, we infer that $dcost(a_i - a_k) = y_i - y_k$ if $P_k \in R_i$, and that $dcost(a_i - a_k) = y_i - y_k + 1$ if $P_k \in Q_i \setminus R_i$. \Box

Here is our geometric algorithm to compute the T(i)'s. Recall that before defining Q_i and R_i^* , for each j, we defined points $P_j = (x_j, y_j)$ with $x_j = a_j \mod G_2$ and $y_j = \lfloor a_j/G_2 \rfloor$.

- T(1) := 0 and $z_1 := T(1) 1 y_1$.
- For i := 2, 3, ..., n, do
 - $-r := i + y_i 2 + \min_{j < i : P_j \in R_i^*} z_j.$ $-q := i + y_i - 1 + \min_{j < i : P_j \in Q_i} z_j.$ $-T(i) := \min\{i - 1, r, q\}.$
 - $-z_i := T(i) i y_i.$

The running time of this algorithm is O(n) plus the time to do the 2n mins involved in the definitions of m_2 and m_3 . The idea is to use a geometric data structure to do each min in time $O(\log N \log \log N)$, for $O(N \log N \log \log N)$ time overall. In order to use a standard geometric data structure, we will have to convert each of the regions Q_i (a halfspace) and R_i (an intersection of two halfspaces) into a halfspace with axisparallel boundaries, and into an orthant (an intersection of two halfspaces with axis-parallel boundaries), respectively.

The algorithm requires one to find $\min_{j < i : P_j \in R_i^*} z_j$ and $\min_{j < i : P_j \in Q_i} z_j$. It is an annoyance that the algorithm needs a minimum over $P_j \in R_i^*$ rather than over $P_j \in R_i$. Were the desired minimum over $P_j \in R_i$, one would just apply to all points the affine transformation T mapping $(x, y) \rightarrow (x, y + x/\Delta)$. This affine transformation maps points $(x, q_i(x)) = (x, (y_i +$ $(x_i - G_1)/\Delta) - x/\Delta$ on the bounding line of Q_i to points $(x, (y_i + (x_i - G_1)/\Delta))$, which are on a horizontal line. The same affine transformation maps points $(x, r_i(x)) = (x, (y_i + x_i/\Delta) - x/\Delta)$ on the "diagonal" bounding line of R_i to $(x, y_i + x_i/\Delta)$, another horizontal line, and maps points (x_i, y) on R_i 's vertical bounding line to $(x_i, y + x_i/\Delta)$, the same vertical line. This means that the question, "Is $(x, y) \in Q_i$?" could be answered, in the transformed space, by asking if T(x, y) is on or below a horizontal line, and "Is $(x, y) \in R_i$?" could be answered in the transformed space by asking if T(x, y)is on or to the right of a vertical line and on or below a horizontal one.

Unfortunately, though, the min is over $P_j \in R_i^*$ instead of over R_i . We now exploit the fact that all the (untransformed) query points are of the form $(x, y) \in$ \mathbb{N}^2 , $x \leq G_2 - 1$. It suffices to make an affine transformation which correctly answers queries about these points.



Figure 2: Here P_i is the solid point, $\Delta = 2$, the relevant part of R_i is given by the shaded area, and R'_i 's bounding lines are thicker.

The idea is to replace each line $q_i(x)$ by a line $q'_i(x)$ which very closely tracks $q_i(x)$ (and to define $Q'_i = \{(x, y) | y \leq q'_i(x)\}$) and (see Figure 2 for intuition) to replace the line $r_i(x)$ by a line $r'_i(x)$ which very closely tracks $r_i(x)$, and to replace the line $x = x_i$ by $x = x_i - \epsilon$ (and to define $R'_i = \{(x, y) | (x \geq x_i - \epsilon) \land (y \leq r'_i(x))\}$) (for a small $\epsilon > 0$) such that (1) all lines $q'_i(x)$ over all iand $r'_i(x)$ over all i have the same slope, and (2) a point $P \in \mathbb{N}^2$ with first coordinate at most $G_2 - 1$ is in Q_i if and only if $P \in Q'_i$, and (3) a lattice point P with first coordinate at most $G_2 - 1$ is in R^*_i if and only if $P \in R'_i$.

This is done as follows. Let $h = \lceil G_2/\Delta \rceil$. The line $y = r_i(x)$, which we will call L_0 , passes through $P_i =$ (x_i, y_i) and $Z := (x_i + h\Delta, y_i - h)$, since it has slope $-1/\Delta$. Consider the line segment corresponding to xcoordinates in interval $I = [x_i, x_i + h\Delta]$. (Clearly $x_i + h\Delta$) $h\Delta \geq G_2$.) For any $x \in I$, the lowest lattice point (x, y)strictly above the line segment is at least $1/\Delta$ above it. This means that if we hold P_i fixed and raise the right endpoint by $1/(2\Delta)$ —in other words, consider the line L_1 passing through P_i and $Z' = (x_i + h\Delta, y_i$ $h + 1/(2\Delta)$)—then "raising" the line segment causes it to "pass through" no lattice points. (The slope $\gamma :=$ $(-h + 1/(2\Delta))/(h\Delta) = -1/\Delta + 1/(2h\Delta^2)$ of L_1 does not depend on *i*.) Clearly, between $x = x_i$ and $x = x_i + x_i$ $h\Delta$, L_1 passes through no lattice points except P_i , and furthermore, the minimum distance upward from any point on L_1 , whose x-coordinate is integral, to a lattice point is at least $1/\Delta - 1/(2\Delta) = 1/(2\Delta)$. In addition, the minimum distance downward from any point on L_1 in that interval to a lattice point other than P_i is at least $(1/(2\Delta))/(h\Delta) = 1/(2h\Delta^2)$, since the interval has length $h\Delta$.

Now simply "lower" L_1 uniformly by $\tau := 1/(4h\Delta^2)$ to get a new line L_2 which is below P_i but above every other lattice point with x-coordinate between x_i and $x_i + h\Delta$ which had been below L_1 . In other words, L_2 is the line connecting $(x_i, y_i - 1/(4h\Delta^2))$ and $(x_i + h\Delta, y_i - h + 1/(2\Delta) - 1/(4h\Delta^2))$. L_2 is the desired boundary for R'_i provided that L_2 crosses the line $y = y_i$ at a point $x = x_i - \epsilon$ for $\epsilon \in (0, 1)$. Where does L_2
hit the line $y = y_i$? We have $\tau/\epsilon = 1/\Delta - 1/(2h\Delta^2)$ so $\epsilon = \tau/(1/\Delta - 1/(2h\Delta^2)) < \tau/(1/(2\Delta)) = 2\Delta\tau = 1/(2h\Delta) < 1.$

To construct $q'_i(x)$ from $q_i(x)$, just use the line of slope γ passing through $(0, q_i(0))$. The set of lattice points on or under that line, between x-coordinates 0 and $h\Delta$, is the same as the set of those on or under $q_i(x)$. However, if $q_i(0)$ is integral, so that $(0, q_i(0))$ is on both the original line and the "rotated" one, one may want to raise the line slightly to prevent roundoff errors.

Now we just apply the affine transformation T' which maps $(x, y) \to (x, y')$, where $y' = y + x/\gamma$, to turn Q'_i into a halfspace with a horizontal bounding line and R'_i into the intersection of a halfspace with a horizontal bounding line and a halfspace with a vertical bounding line.

We apply this affine transformation to all points P_j . We need to do orthogonal range search queries in which we need to find the minimum z_j in a translated quadrant or halfspace. However, since z_i is defined only after all $z_1, z_2, ..., z_{i-1}$ are defined, the key values are not known in advance. (The points themselves, however, are known in advance.)

3.1 Running time analysis

Here is what a data structure must support in order to run the algorithm. We are given, in advance, n points P_i in \mathbb{Z}^2 with $P_i = (x_i, y_i)$. For each i, we will construct key(i) adaptively in the order 1, 2, 3, ..., n, as follows. Initialize key(1) in some way. The data structure must be able to execute the following code:

- for i = 2 to n do:
 - Find a j minimizing key(j) among those j < isatisfying $x_j \le x_i$ and $y_j \le y_i$.
 - Now define key(i) (somehow).
- $\bullet\,$ end for.

The augmented segment tree of Mehlhorn and Näher [5] guarantees the existence of a $O(N \log N \log \log N)$ time algorithm [4]. In fact, a data structure giving a running time of $O(N \log N \log \log N)$ is likely to be implicit in Gabow, Bentley, and Tarjan [2]; however their result as stated (Theorem 3.3 and the discussion above it) is for the case when all key(i) values are known in advance.

We leave open the existence of a $O(N \log N)$ -time algorithm, and suggest Willard [6] or Chan, Larsen, and Pătrascu [1] as a possible starting point.

4 Acknowledgments

The authors thank Hal Gabow and Kurt Mehlhorn for their help finding data structures supporting all query and update operations in $O(n \log n \log \log n)$ time.

References

- T. M. Chan, K. G. Larsen, and M. Pătrascu. Orthogonal range searching on the RAM, revisited. In F. Hurtado and M. J. van Kreveld, editors, *Symposium on Computational Geometry*, pages 1–10. ACM, 2011.
- [2] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In ACM Symposium on Theory of Computing, pages 135– 143, 1984.
- [3] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.
- [4] K. Mehlhorn, 2011. Personal communication.
- [5] K. Mehlhorn and S. Näher. Dynamic Fractional Cascading. Algorithmica, pages 215–241, 1990.
- [6] D. E. Willard. Examining Computational Geometry, Van Emde Boas Trees, and Hashing from the Perspective of the Fusion Tree. SIAM J. Comput., 29(3):1030–1049, 2000.

Point Location in Well-Shaped Meshes Using Jump-and-Walk*

Jean-Lou De Carufel[†]

Craig Dillabaugh[‡]

Anil Maheshwari[§]

Abstract

We present results on executing point location queries in well-shaped meshes in \mathbb{R}^2 and \mathbb{R}^3 using the *Jump*and-Walk paradigm. If the jump step is performed on a nearest-neighbour search structure built on the vertices of the mesh, we demonstrate that the walk step can be performed in guaranteed constant time. Constant time for the walk step holds even if the jump step starts with an approximate nearest neighbour.

1 Introduction

Point location is a topic that has been extensively studied since the origin of computational geometry. In \mathbb{R}^2 , the point location problem, typically referred to as planar point location, can be defined as follows. Preprocess a planar subdivision, specified as the union of n triangles, so that given a query point q, the triangle containing q can be reported efficiently. There are several well known results showing that a data structure with O(n) space can report such queries in $O(\log n)$ time. In \mathbb{R}^3 , we have a subdivision of the three dimensional space into tetrahedra, and again given a query point qwe wish to return the tetrahedron containing q. From a theoretical standpoint, the problem of general spatial point location in \mathbb{R}^3 is still open [1].

In this paper, we consider a more specialized problem. We wish to answer point location queries in two and three dimensions for well-shaped triangular and tetrahedral meshes. A well-shaped mesh, denoted by \mathcal{M} , is one in which all its simplices have bounded aspect ratio (see Definition 1). This assumption is valid for mesh generation algorithms that enforce the well-shaped property on their output meshes [6]. Our motivation came from an external memory setting, where we have examined data structures for representations that permit efficient path traversals in meshes that typically do not fit in the main memory [3].

Let P be the set of vertices defining \mathcal{M} . In this paper we show that, given a query point q, and its (exact or approximate) nearest neighbor $p \in P$, the number of triangles (or tetrahedra) intersected by the line segment pq is bounded by a constant. On the basis of this result, we develop a point location method following the *Jump-and-Walk* paradigm, which typically works as follows (Devroye *et al.* [2], Mücke *et al.* [8]):

Jump Step: Select a set of possible start (jump) points and store them in a data structure that can efficiently answer proximity queries. Given a query point q, locate a nearest neighbor of q (say p) and then Jump to p.

Walk Step: Walk through the sequence of simplices, starting at p, in a straight line, towards q, until the simplex containing q is located.

While jump-and-walk gives expected search times in most instances, and is often slightly less efficient theoretically than other techniques, it has the advantage of being simple and is often very efficient in practice.

1.1 Our Results

We present results for the "Walk"-step in the Jump-and-Walk strategy in \mathbb{R}^2 and \mathbb{R}^3 for well shaped triangular and tetrahedral meshes. In particular, we show that

- 1. Given a well-shaped mesh \mathcal{M} in \mathbb{R}^2 or \mathbb{R}^3 , jumpand-walk search can be performed in the time required to perform an *exact* nearest neighbour search on the vertices of \mathcal{M} plus *constant time* for the walk-step to find the triangle/tetrahedron containing the query point.
- 2. Given a well-shaped mesh \mathcal{M} in \mathbb{R}^2 or \mathbb{R}^3 , jumpand-walk search can be performed in the time required to perform an *approximate* nearest neighbour (see Definition 2) search on the vertices of \mathcal{M} , plus *constant time* for the walk-step to find the triangle/tetrahedron containing the query point.

While we present results in both \mathbb{R}^2 and \mathbb{R}^3 , we feel that our most interesting contribution is the constant time walk-step in \mathbb{R}^3 using approximate nearest neighbour for the jump-step. In \mathbb{R}^3 , there are no efficient structures for answering exact nearest neighbor queries; in spite of that we are able to show that the walk-step can be performed in constant time using only the knowledge of an approximate nearest neighbor. The major advantage of our approach, in addition to being theoretically optimal, as it matches the query time for this setting as presented in [10], is that it is also practical. The practicality of approximate nearest neighbor

^{*}Research supported by funding from NSERC.

[†]Computational Geometry Lab, School of Computer Science, Carleton University

[‡]School of Computer Science, Carleton University, cdillaba@cg.scs.carleton.ca

 $^{^{\$}}$ School of Computer Science, Carleton University, anil@scs.carleton.ca

searching has already been demonstrated, see ANN Library [7], and the implementation of the walk-step is fairly trivial and straightforward.

2 Background

In this paper we consider point location in triangular and tetrahedral meshes in two and three dimensions respectively. We use \mathcal{M} to denote a mesh in either \mathbb{R}^2 or \mathbb{R}^3 , and if we want to be specific we use \mathcal{M}_2 to denote a triangular mesh in \mathbb{R}^2 , and \mathcal{M}_3 to denote a tetrahedral mesh in \mathbb{R}^3 . We assume that the triangles and tetrahedra, which are collectively referred as simplices, are *well-shaped*, a term which will be defined shortly. If all simplices of a mesh \mathcal{M} are well-shaped, then \mathcal{M} is said to be a *well-shaped mesh*. Triangles in \mathcal{M}_2 are considered adjacent if and only if they share an edge. Similarly, tetrahedra in \mathcal{M}_3 are considered adjacent if and only if they share a face.

Well-Shaped Meshes: We begin by stating the *well-shaped* property (refer to [6]).

Definition 1 We say that a mesh \mathcal{M}_2 (\mathcal{M}_3) is wellshaped if for any triangle (tetrahedron) $t \in \mathcal{M}_2$ ($t \in \mathcal{M}_3$), the ratio formed by the radius r(t) of the incircle (insphere) of t and the radius R(t) of the circumcircle (circumpshere) of t is bounded by a constant ρ , i.e. $\frac{R(t)}{r(t)} < \rho$.

In this paper, all meshes and simplicies (triangles and tetrahedra) are assumed to be well-shaped. We make the following observations related to Definition 1.

Observation 1 Let t be a triangle (tetrahedron).

1. Let v be any vertex of t. Denote by e_v (f_v) the opposite edge (face) of v in t. Let

 $\texttt{mdist}(v,t) = \min_{x \in e_v} |xv| \qquad (\texttt{mdist}(v,t) = \min_{x \in f_v} |xv|),$

where the minimum is taken over all points x on e_v (f_v) . Therefore, mdist(v,t) is an upper-bound on the diameter of the incircle (insphere of) t. Formally, $2r(t) \leq \texttt{mdist}(v,t)$.

2. Let e be the longest edge of t. The diameter of the circumcircle (circumsphere) of t is at least as long as e. Formally, $2R(t) \ge |e|$.

Observation 2 There is a lower bound of α for each of the angles in any triangle of \mathcal{M}_2 . There is a lower bound of Ω for each of the solid angles in any tetrahedron of \mathcal{M}_3 . In particular, we have $\alpha \leq \frac{\pi}{3}$ and $\cos(\alpha) = \frac{1+\sqrt{\rho(\rho-2)}}{\rho}$ for the two dimensional case. For the three dimensional case, $\Omega \leq 3 \arccos\left(\frac{1}{3}\right) - \pi$ and $\sin\left(\frac{\Omega}{2}\right) = \frac{3\sqrt{3}}{8\rho^2}$ (see [5]). **Jump-and-Walk for Point Location:** In [8], point location queries using the jump-and-walk are addressed for Delaunay triangulations of a random set of points in \mathbb{R}^2 and \mathbb{R}^3 . Devroye *et al.* [2] showed that the expected search times for the jump-and-walk in Delaunay triangulations range from $\Omega(\sqrt{n})$ to $\Omega(\log n)$, depending on the distribution and the specific data structure employed for the jump step.

Nearest Neighbour Queries: The nearest neighbour query works as follows. Given a point set P and a query point q, return the point $p \in P$ nearest to q, i.e. for all $v \in P$, $|pq| \leq |vq|$. A closely related query is the approximate nearest neighbour (ANN) query defined as follows.

Definition 2 Let P be a set of points in \mathbb{R}^d , q be a query point and $p \in P$ be an exact nearest neighbour of q. Given an $\epsilon \geq 0$, we say that a point $\hat{p} \in P$ is an $(1 + \epsilon)$ -approximate nearest neighbour of q if $|\hat{p}q| \leq (1 + \epsilon)|pq|$.

3 Planar Point Location in \mathcal{M}_2

Let P be the set of vertices of a well-shaped mesh \mathcal{M}_2 and q be a query point lying in a triangle of \mathcal{M}_2 . Let pbe a nearest neighbour of q. Consider the set of triangles encountered in a straight-line walk from p to q in \mathcal{M}_2 .

Lemma 1 The walk-step along pq visits at most $\lfloor \frac{\pi}{\alpha} \rfloor$ triangles.

Proof. Without loss of generality, suppose |pq| = 1. Let $\mathcal{C}(q, |pq|)$ be the circle with centre q and radius |pq|. Since p is a nearest neighbour of q, there is no vertex of \mathcal{M}_2 in the interior of $\mathcal{C}(q, |pq|)$. Denote by ℓ the line through pq and let $p' \neq p$ be the intersection of ℓ with $\mathcal{C}(q, |pq|)$ (see Fig. 1). Since $\mathcal{C}(q, |pq|)$ is a unit circle, the arc pp' has length π . Any triangle intersecting pqintersects ℓ in the interior of $\mathcal{C}(q, |pq|)$. All such triangles have one vertex to the left of ℓ and two vertices to the right of ℓ or vice-versa. (If a vertex is on ℓ , consider it to be on the right.) We separate the triangles intersecting ℓ into two sets, L and R containing the triangles with exactly one vertex to the left, and right, of ℓ , respectively. Consider an arbitrary triangle $t \in L$ and let the vertices of t be a, b and c. The edge ab (respectively ac) intersects $\mathcal{C}(q, |pq|)$ at b' and b'' (respectively at c' and c'') (see Fig. 1). Let $\theta = \angle bac$. Since ab and acare two secants which intersect $\mathcal{C}(q, |pq|)$, we know that $\theta = \frac{1}{2}(\overrightarrow{b'c'} - \overrightarrow{c''b''})$, from which we conclude $\overrightarrow{b'c'} \ge 2\theta$. Hence $b'c' \ge 2\theta \ge 2\alpha$ by Observation 2. Therefore, we can conclude that the set L contains at most $\lfloor \frac{\pi}{2\alpha} \rfloor$ triangles. The same bound holds for triangles in R. Thus the number of triangles intersecting pq is at most $\left|\frac{\pi}{\alpha}\right|$.



Figure 1: A triangle with fixed minimum angle covers an arc bounded by a minimum fixed length on C.

Next consider the scenario where $\hat{p} \in P$ is an approximate nearest neighbor of the query point q. Note that $C(q, |\hat{p}q|)$ may contain vertices of \mathcal{M}_2 . Therefore, the proof of Lemma 1 does not apply for the walk-step along $\hat{p}q$. Next, we prove that for an ANN search structure (see Definition 2), we can find an ϵ such that the number of triangles encountered in a straight line walk from \hat{p} to q, is bounded by a constant. We begin with the following lemma.

Lemma 2 Let t be a well-shaped triangle and C be a circle of radius r(C) such that none of the vertices of t are in the interior of C. If (i) at least two edges of t intersect C or if (ii) t contains the centre of C, then t has at least one edge of length at least $2r(C) \sin \alpha$.

Proof. Let $t = \triangle abc$. From Observation 2, we know that $\alpha \leq \angle bac$.

(i) Suppose that all the edges of t are strictly smaller than 2r(C) sin α for a contradiction. Let C' be the biggest circle that can be constructed such that none of the vertices of t are in the interior of C and at least two edges of t intersect C'. Thus, C' is strictly smaller than the circumcircle of the equilateral triangle of side length 2r(C) sin α. Therefore, by elementary geometry,

$$\begin{aligned} r(\mathcal{C}') &< \frac{2\sqrt{3}r(\mathcal{C})}{3}\sin\alpha\\ &\leq \frac{2\sqrt{3}r(\mathcal{C})}{3}\sin\left(\frac{\pi}{3}\right) \qquad \text{by Observation 2,}\\ &= r(\mathcal{C}), \end{aligned}$$

which is a contradiction.

(ii) Suppose that less than two edges of t intersect Cand t contains the centre of C. If t contains C, then all the edges of t are longer than $2r(C) \ge 2r(C) \sin \alpha$. Suppose exactly one edge of t intersects C. Let ab be this edge. We form a new triangle t'by translating ac and bc inward until one of ac or ab intersects C. Now t' satisfies the hypothesis of Case (i).

Observation 3 Let $t_i = \triangle abc$ be a well-shaped triangle.



Figure 2: Neighbourhood of the triangle t_i from which the path pq leaves C.

- 1. Let t_{i+1} be the well-shaped triangle adjacent to t_i at edge ab. The edges of t_{i+1} have length at least $|ab| \sin \alpha$.
- 2. Let $a \in \mathcal{M}_2$ be a vertex and ab be an edge incident to a. The edges of any triangle incident to a have length at least $|ab| \sin^{\lfloor \frac{\pi}{\alpha} \rfloor} \alpha$.

Proof.

- 1. It follows from the well-shaped property.
- 2. From Observation 3-1, if a triangle t in \mathcal{M}_2 has an edge of length L, then no triangle that can be reached by walking from t through at most dedge adjacent triangles has an edge shorter than $L \sin^{d+1} \alpha$. Then the result follows from Observation 2.

Consider the walk from \hat{p} to q in \mathcal{M}_2 . It intersects the boundary of $\mathcal{C}(q, |pq|)$ at a point x. Let t_i be the first triangle we encounter in the walk from \hat{p} to q that contains x.

Observation 4 t_i has an edge of length at least $2|pq|\sin^2 \alpha$.

Proof. If t_i contains q, then by Lemma 2, t_i has an edge of length at least $2|pq|\sin \alpha \geq 2|pq|\sin^2 \alpha$. If t_i does not contain q, then there is an edge of t_i intersecting $\hat{p}q$ in the interior of $\mathcal{C}(q, |pq|)$. Let a and b be the two vertices of this edge. Consider the triangle t_{i+1} adjacent to t_i across ab. If $q \in t_{i+1}$, then $|ab| \geq 2|pq|\sin^2 \alpha$ by Lemma 2 and Observation 3-1, otherwise t_{i+1} has two edges intersecting $\mathcal{C}(q, |pq|)$. Again, by Lemma 2 and Observation 3-1, $|ab| \geq 2|pq|\sin^2 \alpha$.

Denote the vertices of t_i by a, b and c. Let \mathcal{G} be the union of all the triangles incident to a, b, and c (see Fig. 2).

Lemma 3 Let $x \in t_i$ be the intersection of $\hat{p}q$ with the boundary of C(q, |pq|). Let $y \in \mathcal{G}$ be the intersection of the line through $\hat{p}q$ with the boundary of \mathcal{G} such that x is between q and y. Then $|xy| \geq 2|pq| \sin^{\lfloor \frac{\pi}{\alpha} \rfloor + 4} \alpha$.



Figure 3: Illustration of the proof of Lemma 3.

Proof. Denote by $t_{ac} = \triangle ab'c$ (respectively by $t_{bc} = \triangle a'bc$) the triangle adjacent to t_i at ac (respectively at bc) (see Fig. 2(a)). Note that t_{ac} and t_{bc} are in \mathcal{G} .

By Observations 3-2 and 4, the length of all edges incident to *a* (respectively to *b* and to *c*) is at least $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+2}\alpha$ (respectively $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+2}\alpha$ and $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+3}\alpha$ by Observation 3-1). Therefore \mathcal{G} contains a ball B(a) (respectively B(b) and B(c)) with centre *a* (respectively *b* and *c*) and radius $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+2}\alpha$ (respectively $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+2}\alpha$ and $2|pq|\sin^{\lfloor\frac{\pi}{\alpha}\rfloor+3}\alpha$), which does not contain any vertices of \mathcal{M}_2 in its interior.

To minimize |xy|, we take x on the boundary of t_i . We will find a lower bound for |xy| by supposing, without loss of generality, that $y \in b'c$. Since y is supposed to be on the boundary of \mathcal{G} , it cannot be inside B(c). With $x \in ac$ and $b \in b'c \setminus B(c)$, the smallest possible value for |xy| is $2|pq| \sin^{\lfloor \frac{\pi}{\alpha} \rfloor + 4} \alpha$ (see Fig. 3).

We can now state our main result.

Theorem 4 Let \mathcal{M}_2 be a well-shaped triangular mesh in \mathbb{R}^2 . Given \hat{p} , an $(1 + \epsilon)$ -approximate nearest neighbour of a query point q from among the vertices of \mathcal{M}_2 , the straight line walk from \hat{p} to q visits at most $2\lfloor \frac{\pi}{\alpha} \rfloor$ triangles.

Proof. Following the notation of Lemma 3, if $\hat{p} \in \mathcal{G}$, then the straight line walk from \hat{p} to q visits at most $2\lfloor \frac{\pi}{\alpha} \rfloor$ triangles. There are $\lfloor \frac{\pi}{\alpha} \rfloor$ triangles for the part of the walk inside C(q, |pq|) (see Lemma 1) and $\lfloor \frac{\pi}{\alpha} \rfloor$ triangles for the part of the walk inside \mathcal{G} . Indeed, in the worst case, the walk inside \mathcal{G} will either cross ab', b'c, ca' or a'b. So this walk will either cross the triangles incident to a, the triangles incident to b or the triangles incident to c.

We can ensure that $\hat{p} \in \mathcal{G}$ by building an ANN search structure with $\epsilon \leq 2 \sin^{\lfloor \frac{\pi}{\alpha} \rfloor + 4} \alpha$ on the vertices of \mathcal{M}_2 . Indeed, in this case

$$\begin{aligned} |\hat{p}q| &\leq \left(1 + 2\sin^{\left\lfloor\frac{\pi}{\alpha}\right\rfloor + 4}\alpha\right)|pq| \\ &= |pq| + 2|pq|\sin^{\left\lfloor\frac{\pi}{\alpha}\right\rfloor + 4}\alpha \\ &\leq |pq| + |xy| \qquad \text{by Lemma 3,} \\ &= |qx| + |xy| \\ &= |qy| \end{aligned}$$

because q, x and y are aligned in this order. So \hat{p} must be in \mathcal{G} .

4 Spatial Point Location in M_3

Searching in a well-shaped three dimensional mesh \mathcal{M}_3 can be performed using essentially the same technique as outlined for \mathcal{M}_2 in Section 3. Let P denote the set of vertices of \mathcal{M}_3 . For a query point q, let $p \in P$ be its nearest neighbour. We will perform the walk-step starting at p and walk towards q in a straight line, and we will show that we visit only a constant number of tetrahedra. Let $\mathcal{S}(q, |pq|)$ denote a ball of radius |pq|centred at q.

Theorem 5 Let \mathcal{M}_3 be a well-shaped triangular mesh in \mathbb{R}^3 . Given p, a nearest neighbour of a query point qfrom among the vertices of \mathcal{M}_3 , the walk from p to qvisits at most $\frac{1}{64}\rho^3(\rho^2+4)^3$ tetrahedra.

Proof. We do not prove Theorem 5 due to lack of space. Refer to the extended version of the paper. \Box

Next, we assume that \hat{p} is an approximate nearest neighbor of q. First, we establish a geometric lemma. Consider an arbitrary ball S. We say S is an empty ball if it contains no vertex of \mathcal{M}_3 . Note that edges and faces of \mathcal{M}_3 may intersect S. Let f be a face in \mathcal{M}_3 that intersects S. We have the following Lemma.

Lemma 6 Let $\mathcal{T} = abcd \in \mathcal{M}_3$ be a tetrahedron and \mathcal{S} be a sphere of radius $r(\mathcal{S})$ such that none of the vertices of \mathcal{T} are in the interior of \mathcal{S} . If (i) $f = \triangle abc$ is tangent to \mathcal{S} or if (ii) f crosses \mathcal{S} in a way that $f \cap \mathcal{S}$ is a disk, then $\frac{2}{\rho}r(\mathcal{S})$ is a lower bound on the length of edges ad, bd and cd.

Proof. (i) Let the tangent point be x. Let \mathcal{H} be the supporting plane of f. Without loss of generality, assume that \mathcal{H} is horizontal, and \mathcal{S} is below \mathcal{H} . There are two tetrahedra of \mathcal{M}_3 that are adjacent to f. We will focus on the tetrahedron that is below \mathcal{H} , and denote it by t_{i+1} . Let d be the fourth vertex of t_{i+1} . If we place x at the pole of \mathcal{S} (we are free to rotate \mathcal{S}) and take the equator of \mathcal{S} and project it onto \mathcal{H} , we obtain a cylinder, say \mathcal{C} . The complement of \mathcal{S} with respect to \mathcal{C} defines the region in which d can be placed (see Figure 4). If d is outside this region then |xd| is greater than the radius of \mathcal{S} , and we have a nice lower bound on |xd|. Let the point d' be the projection of d onto \mathcal{H} .

Now consider some placement of the point d, and assume that d touches the surface of S (which is the worst case in this setting). Consider the line segments xd and dd' and observe that



Figure 4: Illustration of proof of Lemma 6.



Figure 5: Determining the bound for |xd|.

- (a) dd' is at at least twice the radius of the insphere $r(t_{i+1})$ of t_{i+1} by Observation 1-1. Formally, $|dd'| \ge 2r(t_{i+1})$.
- (b) xd lies completely within t_{i+1} . Then $|xd| \le 2R(t_{i+1})$ by Observation 1-2.

Without loss of generality we assume that S is centred at the origin of our coordinate system. Consider the situation on the plane through the parallel lines Ox and dd' (both lines are normal to \mathcal{H}) as depicted in Fig. 5. By the definition of a well-shaped tetrahedron we know that $\frac{R(t_{i+1})}{r(t_{i+1})} \leq \rho$, and by the observations above, we have $\frac{|xd|}{|dd'|} \leq \rho$. Let $x' \neq x$ be the intersection of the line through Ox with S. By elementary geometry, the triangles $\triangle x dx'$ and $\triangle dd'x$ are similar. Therefore, $\frac{2r(\mathcal{S})}{|xd|} = \frac{|xx'|}{|xd|} = \frac{|xd|}{|dd'|} \leq \rho$, so $|xd| \geq \frac{2}{\rho}r(\mathcal{S})$ (see Fig. 5).

(ii) If f crosses S, then the intersection of f with S forms a circle on S (because S is empty). Let x' be the center of this circle. If we translate T so that f is tangent to S at x' then T satisfies the hypothesis of Case (i).

Observation 5 Let $t_i = abcd$ be a well-shaped tetrahedron.

- 1. Let t_{i+1} be the well-shaped tetrahedron adjacent to t_i at edge ab. There exists a constant k_{Ω} that depends only on Ω such that the edges of t_{i+1} have length at least $|ab|k_{\Omega}$.
- 2. Let $a \in \mathcal{M}_3$ be a vertex and ab be an edge incident to a. The edges of any tetrahedron incident to ahave length at least $|ab|k_{\Omega}^{\lfloor \frac{2\pi}{\Omega} \rfloor}$.

Proof. 1. Let $v_0 = a$, $v_1 = b$, $v_2 = c$ and $v_3 = d$. Denote the volume of t_i by V and the solid angle at vertex v_i by θ_i . We have (see [5])

$$\sin\left(\frac{\theta_0}{2}\right) = \frac{12V}{\sqrt{\prod_{1 \le i < j \le 3} \left(\left(|v_0 v_i| + |v_0 v_j|\right)^2 - |v_i v_j|^2\right)}} \quad (1)$$

Let $l = |v_0v_1|$ and suppose without loss of generality that the edges of t_{i+1} have length at least 1 (hence $l > \frac{\sqrt{3}}{3}$). We first explain how to find the biggest possible value l_{\max} for l such that $\theta_0 \ge \Omega$. The worst case is when the edges v_1v_2, v_2v_3 and v_1v_3 all have minimum length 1. Therefore, suppose that $\Delta v_1v_2v_3$ is an equilateral triangle. We are looking for the position of v_0 that maximizes l and such that $\theta_0 \ge \Omega$. Let Δ be the line perpendicular to $\Delta v_1v_2v_3$ that contains the centroid of $\Delta v_1v_2v_3$. To maximize l, we need to take v_0 on Δ .

Therefore, the height of t_i with respect to $\Delta v_1 v_2 v_3$ is equal to $\sqrt{l^2 - \frac{1}{3}}$ and $V = \frac{\sqrt{3l^2 - 1}}{12}$. As we move v_0 up, the solid angle θ_0 decreases. Therefore, by (1), we need to find the biggest $l > \frac{\sqrt{3}}{3}$ such that

$$\sin\left(\frac{\Omega}{2}\right) = \frac{\sqrt{3l^2 - 1}}{(4l^2 - 1)\sqrt{4l^2 - 1}}.$$
 (2)

Let l_{max} be the biggest $l > \frac{\sqrt{3}}{3}$ that satisfies (2). Since (2) reduces to a cubic equation in l^2 , l_{max} exists, it is unique and it can be computed exactly. We have $k_{\Omega} = \frac{1}{l_{\text{max}}}$.

2. The proof is similar to the one of Observation 3-2. It uses Observation 5-1 and the fact that the full solid angle is 4π .

Consider the walk from \hat{p} to q in \mathcal{M}_3 . It intersects the boundary of $\mathcal{S}(q, |pq|)$ at a point x. Let t_i be the first tetrahedron we encounter in the walk from \hat{p} to qthat contains x.

Observation 6 t_i has an edge of length at least $\frac{2}{a}|pq|$.

Proof. Similar to the proof of Observation 4. \Box

We can now apply the same approach as we used in \mathcal{M}_2 to show that the number of tetrahedron visited along $\hat{p}q$ is a constant. Denote the vertices of t_i by a, b, c and d. Let \mathcal{G} be the union of all the tetrahedra incident to a, b, c and d.

Lemma 7 Let $x \in t_i$ be the intersection of $\hat{p}q$ with the boundary of S(q, |pq|). Let $y \in \mathcal{G}$ be the intersection of the line through $\hat{p}q$ with the boundary of \mathcal{G} such that x is between q and y. Then $|xy| \geq \frac{2}{\rho} |pq| k_{\Omega}^{\lfloor \frac{2\pi}{\Omega} \rfloor + 1} \sin\left(\frac{\Omega}{2}\right)$.

Proof. We follow the proof of Lemma 3. In two dimensions, the lower bound on |xy| was computed by calculating the shortest exit out of a well-shaped triangle t. This shortest exit is perpendicular to an edge of t and constrained by the radius of the ball B(c). In three dimensions, we calculate the shortest exit out of a well-shaped tetrahedron t. This shortest exit is perpendicular to a face of t, it goes through an edge of t and it is constrained by the radius of a ball in three dimensions. This leads to $|xy| \geq \frac{2}{\rho} |pq| k_{\Omega}^{\lfloor \frac{2\pi}{\Omega} \rfloor + 1} \sin \left(\frac{\Omega}{2} \right)$.

Theorem 8 Let \mathcal{M}_3 be a well-shaped tetrahedral mesh in \mathbb{R}^3 . Given \hat{p} , an $(1 + \epsilon)$ -approximate nearest neighbour of a query point q from among the vertices of \mathcal{M}_3 , the straight-line walk from \hat{p} to q visits at most $\frac{1}{64}\rho^3(\rho^2+4)^3+\left|\frac{2\pi}{\Omega}\right|$ tetrahedra.

Proof. This proof is similar to the proof of Theorem 4 with $\epsilon \leq \frac{2}{\rho} k_{\Omega}^{\lfloor \frac{2\pi}{\Omega} \rfloor + 1} \sin\left(\frac{\Omega}{2}\right)$.

5 Discussion

Our interest in this problem as such grew out of our research into efficient path traversals of large size wellshaped meshes in external memory settings (see [3]). However, it was assumed that the starting tetrahedron on such a path was given as part of the query. Adding the jump-and-walk point location step, results in efficiently answering a number of queries, without this assumption. Such queries include reporting the intersection of a box with the mesh (analogous to a window query in \mathbb{R}^2) or any other convex shape, and reporting streamlines.

To this point in the paper we have omitted any discussion of the data structures employed in the point location step. An ideal option in many ways is to employ a kd-tree, which is simple, can answer nearest neighbour queries in both \mathbb{R}^2 and \mathbb{R}^3 (and has I/O-efficient variants) [9]. Unfortunately, nearest neighbour queries in kd-trees, while good in the expected case [4], can in the worst-case require linear time. However, we are not aware of any work, which analyzes the worst-case query times for kd-trees with respect to vertices of a wellshaped mesh. An interesting follow on research topic to this paper would be to examine if query times for exact nearest neighbours in kd-trees, for points drawn from a well-shaped mesh, are in fact optimal.

In essence, what we have shown in this paper is that jump-and-walk strategy for point location in wellshaped meshes in \mathbb{R}^2 and \mathbb{R}^3 , is practical, simple, and efficient, and requires only the knowledge of an approximate nearest neighbor. It will be worthwhile to explore other geometric configurations where the jumpand-walk can lead to efficient ways to perform point location queries. **Acknowledgements:** We thank the referees for their helpful comments.

References

- M. de Berg, M. J. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Al*gorithms and Applications. Springer.
- [2] L. Devroye, C. Lemaire, and J.-M. Moreau. Expected time analysis for Delaunay point location. *Computational Geometry: Theory and Applications*, 29(2):61–89, 2004.
- [3] C. Dillabaugh. I/O efficient path traversal in wellshaped tetrahedral meshes. In CCCG, pages 121– 124, 2010.
- [4] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw., 3(3):209–226, 1977.
- [5] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT Numerical Mathematics*, 34:268–287, 1994.
- [6] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Geometric separators for finite-element meshes. *SIAM J. Sci. Comput*, 19(2):364–386, 1998.
- [7] D. Mount and S. Ayra. ANN: A library for approximate nearest neighbor searching. http://www.cs. umd.edu/~mount/ANN/, Jan. 2010.
- [8] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Computational Geometry: Theory and Applications*, 12(1-2):63–83, 1999.
- [9] J. T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In SIGMOD, pages 10–18, 1981.
- [10] S.-H. Teng. Fast nested dissection for finite element meshes. SIAM Journal on Matrix Analysis and Applications, 18(3):552–565, 1997.

Open Problems from CCCG 2010

Erik D. Demaine*

The following is a description of the problems presented on August 9, 2010 at the open-problem session of the 22nd Canadian Conference on Computational Geometry held in Winnipeg, Manitoba, Canada.

Coiling Rope in a Box Joseph O'Rourke Smith College orourke@cs.smith.edu

Is there a procedure to decide whether a rope of length L and radius r can be coiled to fit in an $a \times b \times c$ box? All five parameters can be assumed to be rational numbers for the decision question. The rope is a smooth curve with a tubular neighborhood of radius r > 0, such that the rope does not selfpenetrate. In particular, the curve should not turn so sharply that the disks of radius r orthogonal to the curve that determine the tubular neighborhood interpenetrate. For an open curve, each endpoint is surrounded by a ball of radius r.

For a box of dimensions $1 \times 1 \times \frac{1}{2}$ and rope of radius $r = \frac{1}{4}$, perhaps the maximum length achievable is $L = \frac{1}{2} + \frac{\pi}{4} \approx 1.3$, realized by a 'U'-shape as in Figure 1.

Packing circles in a square is a notoriously difficult problem, but perhaps it is easier to pack a rope in a cube, because the continuity of the curve constrains the options.



Figure 1: Overhead view of a rope in a box.

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA, edemaine@mit.edu Joseph O'Rourke[†]

Update. This problem also appeared on Math-Overflow,¹ where Greg and Włodzimierz Kuperberg opine that it is open. At the suggestion of several people during the CCCG presentation, the poser started exploring the 2D version. If $k = \frac{1}{2r}$ is an even integer, then there are two natural strategies for coiling the rope within a box whose height renders it two-dimensional, as illustrated in Figure 2. Interestingly, the length of the core rope curve is identical for the two coilings:

$$L = 2(k-1)(r\pi/2) + 2(k-1)^2r.$$



Figure 2: Two 2D coilings in a $1 \times 1 \times 2r$ box. Here $r = \frac{1}{16}$, k = 8, and $L = \frac{7\pi}{16} + \frac{49}{8} \approx 7.5$.

When Sticks Fall, Will They Weave? Joseph O'Rourke Smith College orourke@cs.smith.edu

Imagine n z-vertical sticks uniformly spaced around a unit-radius circle in the xy-plane. At random times $t_1, t_2, \ldots, t_n \ge 0$, each stick is randomly ε perturbed from the vertical, and they fall under the influence of gravity. Will some sticks form a "teepee" suspended above the xy-plane?

Let us assume that the sticks are one-dimensional segments of height h, perhaps h = 2 so that they span the diameter, and that their base points are pinned to the plane via universal joints. It seems possible that a subset of sticks could fall to form a *weaving* with a cyclic on-top-of graph, as illustrated in Figure 3. Assuming a sufficiently large coefficient of friction μ between pairs of sticks, it

[†]Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu

¹ http://mathoverflow.net/questions/26525/.

seems conceivable that such a structure would not collapse to the plane. Is it possible that some sticks form a woven "teepee" structure above the plane? Or would all sticks ultimately flatten to the plane?



Figure 3: A weaving of four sticks.

Update. This problem also appeared on Math-Overflow,² where Scott Morrison observed that if all sticks are released at the same time t = 0, then they would hit one another with probability zero.

Linkless embeddings of graphs in \mathbb{R}^3 David Eppstein University of California, Irvine eppstein@ics.uci.edu

In one of the early papers on linkless embedding, Sachs [Sac83] asked a question that still remains open: is there an analogue of Fáry's theorem for three-dimensional drawing? That is, if a graph has a linkless or flat embedding with curved or polygonal edges, does it automatically have a linkless or flat embedding with straight line segment edges? If so, how can we find these straight drawings efficiently? If not, which graphs do have linkless straight drawings? An embedding of a graph into \mathbb{R}^3 is *linkless* if, for every pair of disjoint cycles C_1 and C_2 , there is a topological sphere separating C_1 from C_2 ; and an embedding is *flat* if every cycle in the graph forms the boundary of a topological disk that is disjoint from all the other vertices and edges of the graph. A flat embedding is always linkless, while a linkless embedding may not be flat; however, every graph with a linkless embedding also has a flat embedding.

Does every flat embedding have a homeomorphic straight embedding? Two embeddings are *homeomorphic* if there is a continuous deformation of space that takes one embedding to the other. Not every linkless embedding has a homeomorphic straight embedding: for instance, an embedding of the triangle K_3 that ties it into a trefoil knot is linkless, but cannot be straightened (the simplest representation of the trefoil with straight edges requires six edges in the cycle). However, this example is not a flat embedding.

Analogously to Wagner's theorem for planar graphs, the linklessly embeddable graphs may be characterized by a set of seven forbidden graph minors (the *Petersen family*, which includes K_6 and the Petersen graph) [RST95]. Based on this characterization, it is possible to recognize linklessly embeddable graphs and find flat embeddings for them in linear time [KKM10]. For planar graphs, another important result that goes beyond recognition and embedding is Fáry's theorem, which states that if a graph has a noncrossing embedding in the plane with arbitrary curves (or polygonal chains) for its edges, then it also has a noncrossing embedding with straight line segments for its edges. This result underlies many graph drawing techniques, because straight-line edges are easier for computers to draw and easier for humans to read.

References

- [KKM10] Ken-ichi Kawarabayashi, Stephan Kreutzer, and Bojan Mohar. Linkless and flat embeddings in 3-space and the unknot problem. In *Proc. Symp. on Computational Geometry (SoCG '10)*, pp. 97–106, 2010.
- [RST95] Neil Robertson, Paul D. Seymour, and Robin Thomas. Sachs' linkless embedding conjecture. Journal of Combinatorial Theory, Series B, 64 (2):185–227, 1995.
- [CG83] John Conway and C. McA. Gordon. Knots and links in spatial graphs. Journal of graph theory, 7(4):445–453, 1983.
- [Sac83] Horst Sachs. On a spatial analogue of Kuratowski's theorem on planar graphs—An open problem. Graph Theory, pages 230–241, 1983.

Covering points with rectangles Matias Korman Université Libre de Bruxelles mkormanc@ulb.ac.be

Given a set S of n points and an integer $k \leq n$, how efficiently can we find the axis-aligned rectangle of minimum area that covers n - k points of S, that is, all but k of the points? The motivation for the problem comes from clustering, where the k points to ignore are outliers which we would like to identify.

Several known algorithms solve this problem, with running times $O(n + k^3)$ [AB+11], $O(n + k^3)$

 $^{^2~{\}rm http://mathoverflow.net/questions/29660/.}$

 $k^2(n-k)$) [SK98], and $O((n-k)^2 n \log n)$ [AI+91] (when the rectangle can have any orientation). Observe that, unless k = o(n) or k = n - c for some constant c, all these algorithms run in cubic time. The question is whether subcubic time is possible for the general problem.

Several specializations of the problem render it much simpler. For example, if the aspect ratio of the rectangle is prescribed, the problem can be solved in $O(n \log n)$ time [Cha99]. Let L, R, T, B be the set of k leftmost, rightmost, topmost, and bottommost points of S, respectively. If $(L \cup R) \cap (T \cup B) = \emptyset$, each dimension can be solved independently, leading to an $O(n + k^2)$ -time algorithm.

References

- [AI+91] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *Journal* of Algorithms, 12:38–56, 1991.
- [AB+11] H.-K. Ahn, S. W. Bae, E. D. Demaine, M. L. Demaine, S.-S. Kim, M. Korman,
 I. Reinbacher, and W. Son. Covering points by disjoint boxes with outliers. *Computational Geometry: Theory and Applications*, 44(3):178 – 190, 2011.
- [Cha99] T. M. Chan. Geometric applications of a randomized optimization technique. Discrete and Computational Geometry, 22:547–567, 1999.
- [SK98] M. Segal and K. Kedem. Enclosing k points in the smallest axis parallel rectangle. Information Processing Letters, 65:95–99, 1998.

Counting points in circles Maarten Löffler University of California, Irvine mloffler@ics.uci.edu

Given *n* equal-radius circles whose centers form the points of a regular $\sqrt{n} \times \sqrt{n}$ grid, and given *n* points in the plane, how quickly can we count the number of points in each circle? This problem can be solved, in the more general case where the circle centers are not constrained to form a grid, in $O^*(n^{4/3})$ time via batched circular range queries. But does the grid structure help at all?

A similar question arises by dualizing the problem: given n equal-radius circles whose centers form a regular grid, and given n points, count the number of circles containing each point.

Domatic partition problems David Matula Southern Methodist University matula@lyle.smu.edu

The domatic partitioning problem asks to partition a graph into a maximum number of vertex-disjoint dominating sets. It is known to be NP-hard. The new problem is the *independent* domatic partition problem, which seeks to partition a graph into a maximum number of disjoint independent dominating sets. For more details, see [MLM10].

References

[MLM10] Dhia Mahjoub, Angelika Leskovskaya, and David W. Matula. Approximating the independent domatic partition problem in random geometric graphs—An experimental study. In *CCCG*, pages 195– 198, 2010.

Separating and covering points in the plane Filip Morić

Ecole Polytechnique Fédérale de Lausanne filip.moric@epfl.ch

- 1. Let *B* and *R* be sets of *n* blue and *n* red points in the plane in general position (i.e., no three points are collinear). What is the minimum number f(n)such that one can always find a simple polygon with at most f(n) sides that separates the blue and red points (i.e., the blue points are inside and the red points are outside of the polygon)? It is known that $n \leq f(n) \leq 3 \lceil \frac{n}{2} \rceil$. (Note that the problem is interesting only under the general position assumption, for if all the points were on a line in the order red, blue, red, blue, ..., then we would need at least a 2n-gon to separate them.)
- 2a. What is the smallest number g(n) such that any n points in the plane can be covered by a simple (non-self-intersecting) polygonal line with at most g(n) sides? Only trivial bounds are known: $n/2 \leq g(n) \leq n$.
- 2b. What is the smallest number h(n) such that any n points in the plane can be covered by a polygonal line (possibly self-intersecting) with at most h(n) sides? The known bounds are $n/2 \le h(n) \le n/2 + o(n)$, where the lower bound is obvious, while the upper bound is obtained by repeatedly using the Erdős-Szekeres theorem. Thus the gap in this version is quite small.

Update to 2b. At the GWOP 2011 workshop, E. Welzl proposed the following nice version of the problem. Call a set of n points in the plane *perfect* if it can be covered by a polygonal line (possibly self-intersecting) with at most $\lceil n/2 \rceil$ sides. For example, a set of points in convex position is perfect. The problem is to determine the maximum number p(n) such that any set of n points in the plane has a perfect subset of size p(n). By the Erdős-Szekeres theorem, $p(n) = \Omega(\log n)$. Can this bound be improved?

Orthogonal layering S. Mehdi Hashemi Amirkabir University of Technology hashemi@aut.ac.ir

Decompose a graph G into edge subsets E_1, E_2, \ldots, E_k such that each $G[E_i]$ is planar and maximum degree 4. What is the minimum orthogonal thickness $\hat{\Theta}(G)$ of G? The poser conjectures that $\hat{\Theta}(G) \leq \lceil \Delta/4 \rceil + 1$, where Δ is the maximum degree of G. See his paper [TH10] for more details.

References

[TH10] Maryam Tahmasbi and S. Mehdi Hashemi. Orthogonal thickness of graphs. In CCCG, pages 199–202, 2010.

Largest independent set in rectangle-Delaunay Sathish Govindarajan Indian Institute of Science, Bangalore gsat@csa.iisc.ernet.in

Define a rectangle-Delaunay graph for a set of n points in the plane (no two on a horizontal or vertical line) by connecting any two points that are opposite corners of an empty axis-parallel rectangle. This graph can have a quadratic number of edges. What is the size of the largest independent set in this graph, as a worst-case function of n?

This problem is related to conflict-free colorings. Erdős-Szekeres yields a lower bound of $\Omega(\sqrt{n})$, which the poser improved to $\Omega(n^{0.618})$. For random points in a square, Chen et al. [CPZT09] established an upper bound of $O(n(\log \log n)^2/\log n)$. (And this bound is nearly tight for random points in a square.) The poser conjectures n/ polylog n is the right bound.

References

[CPZT09] Xiaomin Chen, János Pach, Mario Szegedy, and Gábor Tardos. Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. *Random Structures & Algorithms*, 34(1):11– 23, Jan. 2009.

Where and How Chew's Second Delaunay Refinement Algorithm Works

Alexander Rand*

Abstract

Chew's second Delaunay refinement algorithm with offcenter Steiner vertices leads to practical improvement over Ruppert's algorithm for quality mesh generation, but the most thorough theoretical analysis is known only for Ruppert's algorithm. A detailed analysis of Chew's second Delaunay refinement algorithm with offcenters is given, improving the guarantee of well-graded output for any minimum angle threshold $\alpha^* \leq 28.60^{\circ}$.

1 Introduction

Ruppert's algorithm for quality triangular mesh generation [10] has a number of theoretical and practical advantages making it the prototypical Delaunay refinement setting: it is relatively simple to state, implement, and analyze. For non-acute input and a minimum angle threshold of about 20.70° , the algorithm is guaranteed to terminate and produce a mesh of optimal size up to a constant factor. Over the past 15 years, this elegant theory has been adjusted and refined to produce better and better meshes. From a theoretical standpoint, Miller, Pav, and Walkington gave an improved analysis of Ruppert's algorithm demonstrating that, under mild assumptions on the input, termination is guaranteed for a minimum angle threshold as high as 26.45° [7]. Off-center Steiner vertices provide an alternative to circumcenter insertion, reducing the mesh sizes produced

*Institute of Computational Engineering and Sciences, The University of Texas at Austin, arand@ices.utexas.edu



Figure 1: Using the boundary of Lake Michigan as input (left, 1537 vertices) and a minimum angle threshold of 25°, the results of Ruppert's algorithm (center, 3707 vertices) and Chew's second Delaunay refinement algorithm with off-centers (right, 2960 vertices) are shown.

in practice. Üngör introduced this concept and demonstrated its success with Ruppert's algorithm [13].

Chew's second Delaunay refinement algorithm [3] was originally studied for meshing surfaces embedded in 3D, but the restriction of this algorithm to the standard 2D mesh generation problem yields two specific advantages over Ruppert's algorithm: the algorithm is theoretically guaranteed to terminate for a larger minimum angle threshold (26.57°) and in practice the resulting meshes have fewer vertices [12]. Most of the improvements to Ruppert's algorithm have been applied to Chew's second Delaunay refinement algorithm and are similarly successful in practice; in fact, the default quality mesh generation algorithm in Triangle [11] is Chew's second Delaunay refinement algorithm with off-centers.

We improve the analysis of Chew's second Delaunay refinement algorithm with off-center vertices. By extending the Miller-Pav-Walkington analysis, we prove the termination of Chew's second Delaunay refinement algorithm for any minimum angle threshold less than 28.60°, and this guarantee holds not only for circumcenters but also for off-center Steiner vertices. Moreover, we generalize the Üngör off-center to a larger class of Steiner vertices characterized by a target angle and note that in some cases these vertices are outside existing selection discs. Finally, a simple example demonstrates the impact of the target angle parameter.

2 Preliminaries

The input to a 2D mesh generator is a consistent collection of straight segments and vertices. The goal of the mesh generator is to add vertices so that a triangulation (in this paper, the constrained Delaunay triangulation) of the final vertex set both conforms to the input segments and contains only high quality triangles.

Formally we follow [7]: a **planar straight-line graph** (PSLG), $\mathcal{G} = (\mathcal{P}, \mathcal{S})$, is a pair of sets of vertices \mathcal{P} and segments \mathcal{S} , such that the endpoints of each segment of \mathcal{S} are contained in \mathcal{P} and the intersection of any two segments of \mathcal{S} is also contained in \mathcal{P} . A PSLG $\mathcal{G}' = (\mathcal{P}', \mathcal{S}')$ is a **refinement** of the PSLG \mathcal{G} if $\mathcal{P} \subset \mathcal{P}'$ and each segment in \mathcal{S} is the union of segments in \mathcal{S}' .

Problem Statement. Given an input PSLG \mathcal{G} and a minimum angle threshold α^* compute a refinement \mathcal{G}' such that all angles of all triangles of the constrained Delaunay triangulation of \mathcal{G}' are larger than α^* .



Figure 2: A PSLG (left) with local feature size indicated at several points (gray) and a refinement (right) of the PSLG that gives a quality, conforming triangulation.

The local feature size at point \mathbf{x} with respect to PSLG \mathcal{G} , $lfs(\mathbf{x})$, is the radius of the smallest closed disk centered at \mathbf{x} which intersects two *disjoint* features of \mathcal{G} . Most Delaunay refinement algorithm analysis is based on relating the mesh size to the local feature size of the input PSLG. Throughout this paper, local feature size is always considered with respect to the input PSLG. Moreover, local feature size is 1-Lipschitz: $lfs(\mathbf{x}) \leq lfs(\mathbf{y}) + |\mathbf{x} - \mathbf{y}|$.

Before stating and analyzing Chew's second Delaunay refinement algorithm, we state one fact about constrained Delaunay triangulations which satisfy an empty circumdisk property with respect to *visible* vertices; for a complete definition see [2].

Proposition 1 Let \mathcal{T} be a constrained Delaunay triangulation of PLSG $(\mathcal{P}, \mathcal{S})$. Suppose that triangle $T \in \mathcal{T}$ has circumcenter \mathbf{c} and that \mathbf{c} is not visible to T. Then T lies inside the diametral disk of the constrained segment $S \in \mathcal{S}$ nearest to T that prevents visibility.

3 Chew's Second Delaunay Refinement Algorithm

Stated carefully as Algorithm 1, Chew's second Delaunay refinement algorithm has a few key differences from Ruppert's algorithm. The final constrained Delaunay triangulation is generated from three types of vertices, classified by why they were inserted into the mesh: input vertices, midpoints, and circumcenters.

Algorithm 1 Chew's second Delaunay refinement
Require: PSLG \mathcal{G} and angle threshold α^* .
Compute constrained Delaunay triangulation \mathcal{T} of \mathcal{G} .
while \mathcal{T} contains a poor quality triangle T do
if T encroaches a segments S then
Remove circumcenters from diametral disk of S .
Split S by adding its midpoint to \mathcal{T} .
else
Insert the circumcenter of T into \mathcal{T} .
end if
end while

Two particular steps above must be made precise. **Encroachment**. A segment S is encroached if there is a poor quality triangle T in the current triangulation such that T and the circumcenter of T lie on opposite sides of S, and T is visible to S. Note the "converse": if T and its circumcenter lie on the opposite sides of S, then some segment (but possibly not S) is encroached.

Vertex Removal. When adding the midpoint \mathbf{m} of a segment S, Chew's algorithm removes circumcenter vertices which lie in the diametral disk of S. In this treatment, we slightly relax this operation and fully specify a procedure for removing vertices. After inserting \mathbf{m} , the nearest visible neighbor to \mathbf{m} is removed if it is a circumcenter, and this is repeated until the nearest visible neighbor is not a circumcenter. Some circumcenters may remain in the diametral disk of S.

The termination of Chew's second Delaunay refinement algorithm and good grading of the resulting mesh follow from a proof that no two vertices are placed too close together. The insertion radius $r_{\mathbf{q}}$ of vertex \mathbf{q} is the distance from \mathbf{q} to the nearest visible vertex in the mesh immediately following the insertion of **q**. We call a mesh well-graded if there exists C depending only upon α^* such that for all vertices **q** inserted by the algorithm, $lfs(\mathbf{q}) \leq Cr_{\mathbf{q}}$. This is a natural measure of success of a mesh generation algorithm: it guarantees termination and that the size of the triangles in the mesh are proportional to the underlying size of the input geometry. Proof that a mesh generation algorithm produces a well-graded mesh is usually performed via induction using an appropriate previously inserted vertex (called the parent vertex) on which to base the estimate.

The **parent** of a vertex \mathbf{q} , denoted $\mathbf{p}(\mathbf{q})$, is defined to be a specific vertex near \mathbf{q} following insertion:

- (1) If \mathbf{q} is a circumcenter, then $\mathbf{p}(\mathbf{q})$ is the newest vertex on the shortest edge of triangle T of which \mathbf{q} is the circumcenter.
- (2) If q is a midpoint and the nearest visible neighbor to q is not contained in the input segment containing q, then p(q) is this nearest visible neighbor.
- (3) If \mathbf{q} is a midpoint and after deletion of some vertices no vertices remain in the diametral disk of S, let \mathcal{P}_r be the set containing all removed circumcenters. If either endpoint of S is newer than than any vertex in \mathcal{P}_r , the most recently inserted endpoint of S is the $\mathbf{p}(\mathbf{q})$. Otherwise, $\mathbf{p}(\mathbf{q})$ is the vertex in \mathcal{P}_r with the smallest insertion radius.

Define $\mathbf{p}_2(\mathbf{q}) := \mathbf{p}(\mathbf{p}(\mathbf{q})), \ \mathbf{p}_3(\mathbf{q}) := \mathbf{p}(\mathbf{p}(\mathbf{p}(\mathbf{q}))), \ \text{etc.}$ Next we prove Chew's second Delaunay refinement algorithm succeeds for non-acute input.

Theorem 2 ([12]) For $\alpha^* < \tan^{-1}(1/2) \approx 26.6^{\circ}$ and non-acute input, Chew's second Delaunay refinement algorithm terminates producing a well-graded, quality mesh.

This proof follows the argument in [12] using the slightly relaxed vertex removal procedure mentioned



Figure 3: Subcases 3b (left) and 3c (right) in Theorem 2.

previously. The cases are carefully enumerated so the proof can be augmented in later sections to provide an improved analysis and accept variants of the algorithm.

Proof. To prove that the resulting mesh is well-graded, we inductively find two constants $0 < C_c < C_m < \infty$ such that $lfs(\mathbf{q}) < C_c r_{\mathbf{q}}$ for any circumcenter and $lfs(\mathbf{q}) < C_c r_{\mathbf{q}}$ for any midpoint. We consider three cases corresponding to the definition of the parent vertex. <u>Case 1</u>. **q** is a circumcenter. Then,

$$\begin{aligned} \text{lfs}(\mathbf{q}) &\leq |\mathbf{q} - \mathbf{p}(\mathbf{q})| + \text{lfs}(\mathbf{p}(\mathbf{q})) \leq r_{\mathbf{q}} + C_m r_{\mathbf{p}(\mathbf{q})} \\ &\leq (1 + 2C_m \sin \alpha) r_{\mathbf{q}}. \end{aligned} \tag{1}$$

<u>Case 2</u>. **q** is a midpoint and a vertex other than **q** remains in the diametral disk of the segment which was split. Then $\mathbf{p}(\mathbf{q})$ must be an input vertex or midpoint. Then since the input is non-acute, this vertex belongs to an input feature which is disjoint from the input segment containing **q** and thus

$$lfs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}(\mathbf{q})| = r_{\mathbf{q}}.$$
(2)

<u>Case 3</u>. \mathbf{q} is a midpoint and the diametral disk of the newly split segment is empty (other than \mathbf{q}). Recalling Proposition 1, all the vertices of the encroaching triangle must lie inside the diametral disk of the segment containing \mathbf{q} .

<u>Subcase 3a.</u> $\mathbf{p}(\mathbf{q})$ is a midpoint. The assumption of non-acute input and the parent vertex definition imply that $\mathbf{p}(\mathbf{q})$ is an endpoint of the segment S. Recalling Proposition 1, let \mathbf{c} be a circumcenter that is older than $\mathbf{p}(\mathbf{q})$ and was removed from the diametral disk of S. Since \mathbf{c} was not removed when $\mathbf{p}(\mathbf{q})$ was inserted, the diametral disk of $\mathbf{p}(\mathbf{q})$ was not completely emptied and thus Case 2 applies to $\mathbf{p}(\mathbf{q})$. So $lfs(\mathbf{p}(\mathbf{q})) \leq r_{\mathbf{p}(\mathbf{q})} \leq$ $|\mathbf{p}(\mathbf{q}) - \mathbf{c}|$, and thus

$$lfs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}(\mathbf{q})| + lfs(\mathbf{p}(\mathbf{q})) \le r_{\mathbf{q}} + r_{\mathbf{p}(\mathbf{q})} \le 3r_{\mathbf{q}}.$$
(3)

<u>Subcase 3b.</u> $\mathbf{p}(\mathbf{q})$ is a circumcenter and at least two circumcenters were removed from the half of the diametral disk of S which is visible to $\mathbf{p}(\mathbf{q})$. Since all of these circumcenters were inserted after the endpoints of S (by the definition of the parent vertex), one of these vertices must have an insertion radius no larger than $r_{\mathbf{q}}$; see Figure 3(left). Then,

$$lfs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}(\mathbf{q})| + lfs(\mathbf{p}(\mathbf{q})) \le (1 + C_c)r_{\mathbf{q}}.$$
 (4)

<u>Subcase 3c</u>. $\mathbf{p}(\mathbf{q})$ is a circumcenter and $\mathbf{p}(\mathbf{q})$ was the only circumcenter removed from the half of the diametral disk of S visible to $\mathbf{p}(\mathbf{q})$. Then to form a skinny triangle with circumcenter on the opposite side of S, $\mathbf{p}(\mathbf{q})$ must belong to the shaded area in Figure 3(right). Then $r_{\mathbf{p}(\mathbf{q})} \leq r_{\mathbf{q}}/\cos \alpha^*$ and thus,

$$lfs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}(\mathbf{q})| + lfs(\mathbf{p}(\mathbf{q})) \le r_{\mathbf{q}} + C_c r_{\mathbf{p}(\mathbf{q})}$$
$$\le \left(1 + \frac{C_c}{\cos\alpha}\right) r_{\mathbf{q}}.$$
(5)

The requirements from the various cases (1)-(5) can be summarized by three conditions: $C_c \ge 1+2C_m \sin \alpha$, $C_m \ge 3$, and $C_m \ge 1 + \frac{C_c}{\cos \alpha}$. Suitable constants exist only if $\tan \alpha^* < 1/2$.

4 Off-Centers

Off-center Steiner vertices were developed as an alternative to circumcenter insertion to reduce the number of vertices inserted by Delaunay refinement algorithms [13]. We use the term off-center (or Υ -off-center to identify the parameter described below) to refer to the special class of Steiner points described by Üngör as opposed to the more general selection disks [1, 5] or selection regions [4, 6] in the literature.

If triangle T has a smallest angle less than $\alpha^*/2$, then inserting its circumcenter is guaranteed to create another poor-quality triangle since the newly inserted circumcenter and the shortest edge of T form a poor quality triangle. Üngör recognized that by selecting an alternative Steiner point, the mesh generator can control the quality of this particular newly formed triangle and, in practice, produce a smaller mesh.

First, we define the class of Υ -off-centers and remark how they generalize Üngör's definition. Let T be a poor quality constrained Delaunay triangle (i.e., the smallest angle of T, denoted α_T , is less than α^*), let $\overline{\mathbf{q}_1 \mathbf{q}_2}$ be the shortest edge of T, and let \mathbf{c} denote the circumcenter of T. The Υ -off-center \mathbf{c}' is an attempt to create a new triangle with smallest angle Υ . If \mathbf{q}_1 and \mathbf{q}_2 are the endpoints of the shortest edge of triangle T, the Υ -off-center \mathbf{c}' is defined as the unique point such that (i) $|\mathbf{q}_1 - \mathbf{c}'| = |\mathbf{q}_2 - \mathbf{c}'|$, (ii) $\angle \mathbf{q}_1 \mathbf{c} \mathbf{q}_2 = \Upsilon$, and (iii) $(\mathbf{c} - \mathbf{q}_1) \cdot (\mathbf{c}' - \mathbf{q}_1) > 0$. See Figure 4 for a depiction of the Υ -off-center region.

Ungör's original work suggested using $\Upsilon_T := \max(2\alpha_T, \alpha^*)$ which separates the points as much as possible without creating a poor quality triangle between the new off-center and the shortest edge of the split triangle. In this setting, the algorithm the was shown to terminate and produce a well-graded mesh.

Theorem 3 (Ungor [13]) Let minimum angle parameter $\alpha^* < \arcsin(1/(2\sqrt{2}))$ be given. Then Ruppert's algorithm with Υ -off-centers and $\Upsilon_T = \max(2\alpha_T, \alpha^*)$ terminates producing a well-graded, quality mesh.



Figure 4: For a poor quality triangle T, the set of admissible Υ -off-centers is shown with the triangle circumcenter **c** and a typical Υ -off-center **q**.

In Triangle [11], slightly larger values of Υ_T (about 5%) are used. In practice this makes "bunches" of nearly minimal quality triangles likely to appear near input edges and yields a mesh with fewer vertices. The proof of Theorem 3 can be extended to admit any $\Upsilon_T \in [2\alpha_T, \alpha^*]$ and (recalling Proposition 1) Chew's second Delaunay refinement algorithm. We provide a more detailed analysis which admits larger values of Υ_T .

Theorem 4 If $\alpha^* < \tan^{-1}(1/2)$ and $\Upsilon_T \in [2\alpha_T, 2\sin^{-1}(\cos(\alpha^*/2)))$, Chew's second Delaunay refinement algorithm with Υ -off-centers terminates producing a well-graded, quality mesh.

Proof. We will verify that the general structure of the proof of Chew's algorithm still applies, albeit with a few additional cases. Estimates on the insertion radii of Υ -off-centers must be revisited.

<u>Case 1</u>. Let \mathbf{q} denote an Υ -off-center associated with poor quality triangle T with shortest edge $\overline{\mathbf{v}_1 \mathbf{v}_2}$ and \mathbf{v}_1 is more recently inserted than \mathbf{v}_2 . Since the nearest vertex to \mathbf{q} may not be a vertex of T, we must deal with two subcases. In one of these subcases, the parent vertex of \mathbf{q} will be redefined.

<u>Subcase 1a.</u> \mathbf{v}_1 is the nearest vertex to \mathbf{q} . Then,

$$\begin{aligned} \operatorname{lfs}(\mathbf{q}) &\leq |\mathbf{q} - \mathbf{v}_1| + \operatorname{lfs}(v_1) \leq r_{\mathbf{q}} + C_m r_{\mathbf{v}_1} \\ &\leq \left(1 + 2C_m \sin \frac{\Upsilon_T}{2}\right) r_{\mathbf{q}}. \end{aligned} \tag{6}$$

<u>Subcase 1b.</u> $\mathbf{u}_1 \neq \mathbf{v}_1$ is the nearest vertex to \mathbf{q} . The edge $\overline{\mathbf{qu}_1}$ is shared by two new Delaunay triangles and let \mathbf{u}_2 denote the additional vertex of one of these triangles that is nearest to \mathbf{u}_1 . Since \mathbf{q} must be a Delaunay neighbor to \mathbf{v}_1 and \mathbf{v}_2 , \mathbf{u}_1 and \mathbf{u}_2 must both live in the (closed) diametral disk of $\overline{\mathbf{v}_1\mathbf{v}_2}$, and thus $|\mathbf{u}_1 - \mathbf{u}_2| \leq |\mathbf{v}_1 - \mathbf{v}_2|/\sqrt{2}$. Define the parent of \mathbf{c}' to be the newest vertex in $\{\mathbf{u}_1, \mathbf{u}_2\}$. Then

$$fs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}(\mathbf{q})| + lfs(\mathbf{p}(\mathbf{q})) \le r_{\mathbf{q}} + C_m r_{\mathbf{p}(\mathbf{q})}$$
$$\le \left(1 + \sqrt{2}C_m \sin\frac{\Upsilon_T}{2}\right) r_{\mathbf{q}}.$$
(7)

Cases 2 and 3 of Theorem 2 are identical in the Υ -offcenter algorithm. Now the worst case involves simultaneously satisfying Subcases 1a and 3c:

$$C_c \ge 1 + 2C_m \sin \frac{\Upsilon_T}{2}; \qquad C_m \ge 1 + \frac{C_c}{\cos \alpha^*}.$$

If $\Upsilon_T < 2 \sin^{-1}(\cos(\alpha^*/2))$, C_c and C_m exist. \Box

Observation 1 The region of admissible Υ -off-centers is not a subset of the selection disks in [1, 5]: the larger values of Υ_T lie outside the standard disk.

5 The Three Circumcenter Lemma

The critical cases in the proofs of Theorems 2 and 4 occur when a segment midpoint is inserted following encroachment due to a circumcenter. Circumcenters always have larger insertion radii than their parent vertices, while midpoints can have slightly smaller radii. The improved analysis of Ruppert's algorithm by Miller, Pav, and Walkington [7] demonstrated that several circumcenters must lie between certain midpoints in a sequence of parent vertices and thus insertion radii gains from the extra circumcenters can be used to offset the insertion radii reduction of the final midpoint. The result improved the admissible minimum angle threshold of Ruppert's algorithm from 20.70° to 26.45° .

Let \mathbf{q} be a midpoint inserted by a Delaunay refinement algorithm. The **circumcenter** (or Υ -off-center) sequence associated with \mathbf{q} is the sequence of points $\{\mathbf{p}_i(\mathbf{q})\}_{i=0}^n$, where n is the smallest positive index such that $\mathbf{p}_n(\mathbf{q})$ lies on a feature of the input PSLG. $\mathbf{q} = \mathbf{p}_0(\mathbf{q})$ is called the final vertex in the sequence and $\mathbf{p}_n(\mathbf{q})$ is called the initial vertex in the sequence. The crux of the Miller-Pav-Walkington analysis relies on studying circumcenter sequences that begin and end on the same input segment.

Lemma 5 (Miller-Pav-Walkington [7]) If a circumcenter sequence both (i) begins and ends on the same input segment and (ii) the insertion radius of the final vertex is no larger than that of the initial vertex, then the sequence contains at least three circumcenters.

The only property of circumcenters that is used in the proof of Lemma 5 is that circumcenters lie on the boundary of the Voronoi cell of their parent vertex. Thus the lemma can be extended to Υ -off-centers as stated below. For technical reasons to be made clear in the upcoming proof define $A(\alpha) := 2 \sin^{-1}((\cos(\alpha^*/2))^{1/3})$.

Corollary 6 Let $\Upsilon_T \in [2\alpha_T, A(\alpha_T))$. If a Υ -off-center sequence (i) begins and ends on the same input segment, (ii) the insertion radius of the final vertex is no larger than that of the initial vertex, and (iii) contains only vertices handled by Theorem 4 Subcase 1a, then the sequence contains at least three Υ -off-centers.

This section closes with a related technical lemma.

Lemma 7 Let $\Upsilon_T \in [2\alpha_T, A(\alpha_T))$ and let $\{\mathbf{p}_i(\mathbf{q})\}_{i=0}^n$ be an Υ -off-center sequence. Then there exists C_d such that $|\mathbf{q} - \mathbf{p}_n(\mathbf{q})| \leq C_d r_{\mathbf{q}}$.

1

6 Restricted Input Class

The core of the argument is given by restricting attention to PSLGs with no adjacent input segments.

Theorem 8 Suppose no segments in the input PSLG are adjacent. Let $\alpha \leq 28.60^{\circ}$ and select Υ -off-centers such that $\Upsilon_T \in [2\alpha_T, A(\alpha_T))$. Chew's second Delaunay refinement algorithm terminates producing a wellgraded, quality mesh.

Proof. The proof involves considering the interaction between cases in Theorem 4. The estimates for (sub)cases 1a, 1b, 2, 3a, and 3b are used without any changes. Let \mathbf{q} be a "subcase 3c"-vertex and let $\mathbf{P} = {\mathbf{p}_i(\mathbf{q})}_{i=0}^n$ be the associated Υ -off-center sequence.

<u>Case A</u>: There is at least one vertex $\mathbf{p}_j(\mathbf{q}) \in \mathbf{P}$ that is a "subcase 1b"-vertex. Since $\alpha^* < 30^\circ$, $r_{\mathbf{p}_1(\mathbf{q})} > r_{\mathbf{p}_2(\mathbf{q})} > \dots > r_{\mathbf{p}_n(\mathbf{q})}$. Then (applying Lemma 7),

$$\begin{aligned} \operatorname{lfs}(\mathbf{q}) &\leq |\mathbf{q} - \mathbf{p}_{j}(\mathbf{q})| + \operatorname{lfs}(\mathbf{p}_{j}(\mathbf{q})) \\ &\leq C_{d} r_{\mathbf{q}} + \left(1 + C_{m} \sqrt{2} \sin \frac{\Upsilon_{T}}{2}\right) r_{\mathbf{p}_{j+1}(\mathbf{q})} \\ &\leq \left(C_{d} + \left(1 + C_{m} \sqrt{2} \sin \frac{\Upsilon_{T}}{2}\right) \frac{1}{\cos \alpha^{*}}\right) r_{\mathbf{q}}. \end{aligned}$$
(8)

<u>Case B</u>: **P** contains no "subcase 1b" vertices and $r_{\mathbf{q}} > r_{\mathbf{p}_n(\mathbf{q})}$. Since all subsegments are derived by midpoint splits from an original segment, $r_{\mathbf{q}} \ge 2r_{\mathbf{p}_n(\mathbf{q})}$ and

$$lfs(\mathbf{q}) \le |\mathbf{q} - \mathbf{p}_n(\mathbf{q})| + lfs(\mathbf{p}_n(\mathbf{q})) \le (C_d + C_m/2)r_{\mathbf{q}}.$$
(9)

<u>Case C</u>: **P** contains no "subcase 1b" vertices and $r_{\mathbf{q}} \leq r_{\mathbf{p}_n(\mathbf{q})}$. By Corollary 6, $n \geq 4$. Υ -off-centers are constructed such that $2\sin(\Upsilon_T/2)r_{\mathbf{p}_i(\mathbf{q})} > r_{\mathbf{p}_{i+1}(\mathbf{q})}$ for $i \in \{1, 2, 3\}$. Then

$$\begin{aligned} \text{lfs}(\mathbf{q}) &\leq |\mathbf{q} - \mathbf{p}_4(\mathbf{q})| + \text{lfs}(\mathbf{p}_4(\mathbf{q})) \\ &\leq C_d r_{\mathbf{q}} + C_m 8 \sin^3 \left(\Upsilon_T / 2\right) r_{\mathbf{p}_1(\mathbf{q})} \\ &\leq \left(C_d + C_m \frac{8 \sin^3 \left(\Upsilon_T / 2\right)}{\cos \alpha^*}\right) r_{\mathbf{q}}. \end{aligned} \tag{10}$$

Requirement (10) is stronger than (8) and (9) so we focus our attention there. $A(\alpha)$ has been defined so that $8\sin^3(\Upsilon_T/2)/\cos\alpha^* < 1$ and thus a suitable constant C_m exists. The interval $[2\alpha_T, A(\alpha_T))$ is nonempty exactly when $8\sin^3\alpha^*/\cos\alpha^* < 1$ which is equivalent to our assumption $\alpha \leq 28.60^\circ$.

7 General Input

Acute angles between input segments pose a fundamental problem in Delaunay refinement and any application of the three circumcenter lemma requires some restrictions on the allowable adjacent input segments [7]. Perhaps the simplest protection strategy is to split adjacent segments at equal lengths proportional to the local



Figure 5: Meshes produced using Υ -off-centers, $\alpha^* = 28^{\circ}$. (top left) $\Upsilon = 0$ (i.e., circumcenter insertion) gives 4975 vertices. (top center) $\Upsilon = 27 \Rightarrow 6475$ vertices. (top right) $\Upsilon = 29 \Rightarrow 3432$ vertices. (bottom left) $\Upsilon = 40 \Rightarrow 3955$ vertices. (bottom center) $\Upsilon = 50 \Rightarrow 5346$ vertices. (bottom right) $\Upsilon = 55 \Rightarrow 8617$ vertices.

feature size and disallow the resulting adjacent subsegments to be split by the algorithm; a description of this "collar" protection strategy can be found in [9]. The advantage of this approach is that following initial grooming there are no adjacent input segments that can be refined which ensures the analysis of Theorem 8 holds.

Corollary 9 Let $\alpha \leq 28.60^{\circ}$ and select Υ -off-centers such that $\Upsilon_T \in [2\alpha_T, A(\alpha_T))$. Chew's second Delaunay refinement algorithm with "collar" vertex protection terminates producing a well-graded, quality mesh away from small input angles.

Another strategy for protecting small input angles (which we call the "wedge" method) disallows the refinement of poor quality triangles which lie between adjacent input segments [7]. This approach is especially important because no large angles are created even in the presence of very small input angles. The complete analysis of this scheme is rather involved and only appears in [8], but the crux of the analysis is the threecircumcenter lemma. Thus we claim that this algorithm also succeeds in creating a well-graded mesh.

Claim Let $\alpha \leq 28.60^{\circ}$ and select Υ -off-centers such that $\Upsilon_T \in [2\alpha_T, A(\alpha_T))$. Chew's second Delaunay refinement algorithm with "wedge" vertex protection terminates producing a well-graded, quality mesh away from small input angles.



Figure 6: Meshes produced using $\alpha^* = 5^\circ$ with $\Upsilon = 6^\circ$ (left, 1660 vertices) and $\Upsilon = 59.5^\circ$ (right, 6072 vertices).

8 Example

Using a 1537 vertex boundary of Lake Michigan as input, we give examples demonstrating the impact that Υ -off-centers have on the meshes generated. To denote a fixed target angle $\Upsilon = \gamma$ is used as a shorthand for the strategy $\Upsilon_T = \max(2\alpha_T, \gamma)$. Figures 5 and 6 contain meshes generated for the Lake Michigan example using various values of Υ and α^* . Figure 7 contains histograms of the smallest angles of all the triangles in meshes resulting from different Υ values and Figure 8 plots the number of mesh vertices as a function of Υ .

Acknowledgment

The author acknowledges useful discussions with Noel Walkington and Todd Phillips. All example meshes were generated using Jonathan Shewchuk's Triangle program (modified slightly).



Figure 7: Histograms of the smallest angle of each triangle of the mesh resulting from several algorithm variants and $\alpha = 25^{\circ}$.







Figure 8: The number of vertices in the resulting mesh using Υ -off-centers for various values of Υ and $\alpha = 25^{\circ}$.

References

- A. Chernikov and N. Chrisochoides. Generalized twodimensional Delaunay mesh refinement. SIAM J. Sci. Comput., 31(5):3387–3403, 2009.
- [2] L. P. Chew. Constrained Delaunay triangulations. In Proc. 3rd Symp. Comput. Geom., pages 215–222, 1987.
- [3] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In Proc. 9th Symp. Comput. Geom., pages 274–280, 1993.
- [4] H. Erten and A. Üngör. Quality triangulations with locally optimal Steiner points. SIAM J. Sci. Comput., 31:2103-2130, 2009.
- [5] P. Foteinos, A. Chernikov, and N. Chrisochoides. Fully generalized two-dimensional constrained Delaunay mesh refinement. *SIAM J. Sci. Comput.*, 32(5):2659–2686, 2010.
- [6] B. Hudson. Safe Steiner points for Delaunay refinement. In Res. Notes 17th Int. Meshing Roundtable, 2008.
- [7] G. L. Miller, S. E. Pav, and N. Walkington. When and why Delaunay refinement algorithms work. *Int. J. Comput. Geom. Appl.*, 15(1):25–54, 2005.
- [8] S. E. Pav. Delaunay Refinement Algorithms. PhD thesis, Carnegie Mellon University, May 2003.
- [9] A. Rand and N. Walkington. Collars and intestines: Practical conforming Delaunay refinement. Proc. 18th Int. Meshing Roundtable, pages 481–497, 2009.
- [10] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. J. Algorithms, 18(3):548-585, 1995.
- [11] J. R. Shewchuk. Triangle: A 2D quality mesh generator and Delaunay triangulator. http://www.cs.cmu.edu/ ~quake/triangle.html.
- [12] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Comput. Geom. Theory Appl.*, 22(1–3):86–95, 2002.
- [13] A. Üngör. Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. In Proc. 6th Latin Amer. Symp. Theor. Inform., pages 152–161, 2004.

Probabilistic Bounds on the Length of a Longest Edge in Delaunay Graphs of Random Points in *d*-Dimensions *

Esther M. Arkin[†]

Antonio Fernández Anta[‡]

.nta[‡] Joseph S. B. Mitchell[§]

Miguel A. Mosteiro[¶]

Abstract

Motivated by low energy consumption in geographic routing in wireless networks, there has been recent interest in determining bounds on the length of edges in the Delaunay graph of randomly distributed points. Asymptotic results are known for random networks in planar domains. In this paper, we obtain upper and lower bounds that hold with parametric probability in any dimension, for points distributed uniformly at random in domains with and without boundary. The results obtained are asymptotically tight for all relevant values of such probability and constant number of dimensions, and show that the overhead produced by boundary nodes in the plane holds also for higher dimensions. To our knowledge, this is the first comprehensive study on the lengths of long edges in Delaunay graphs.

1 Introduction

We study the length of a longest Delaunay edge for points randomly distributed in multidimensional Euclidean spaces. In particular, we consider the Delaunay graph for a set of n points distributed uniformly at random in a d-dimensional body of unit volume. It is known that the probability that uniformly distributed random points are not in general position ¹ is negligible and therefore it is safe to focus on generic sets of points [8], which we do throughout the paper.

The motivation to study such settings comes from the Random Geometric Graph (RGG) model in which n nodes are distributed uniformly at random in a disk or, more generally, according to some specified density function on d-dimensional Euclidean space [16]. The problem has attracted recent interest because of its applications in energy-efficient geometric routing and flooding in wireless sensor networks (see, e.g., [7, 11, 12, 13]).

Related Work. Kozma, Lotker, Sharir, and Stupp [11] show that the asymptotic length of a longest Delaunay edge depends on the sum, σ , of the distances to the boundary of its endpoints. More specifically, their bounds are $O(\sqrt[3]{(\log n)/n})$ if $\sigma \leq ((\log n)/n)^{2/3}$, $O(\sqrt{(\log n)/n})$ if $\sigma \ge \sqrt{(\log n)/n}$, and $O((\log n)/(n\sigma))$ otherwise. Kozma et al. also show, in the same setting, that the expected sum of the squares of all Delaunay edge lengths is O(1). In [5] the authors consider the Delaunay triangulation of an infinite random (Poisson) point set in d dimensional space. In particular, they study different properties of the subset of those Delaunay edges completely included in a cube $[0, n^{1/d}] \times \cdots \times [0, n^{1/d}]$. For the maximum length of a Delaunay edge in this setting, they observe that in expectation is in $\Theta(\log^{1/d} n)$.

The lengths of longest edges in geometric graphs induced by random point sets has also been studied for graphs related to the Delaunay, including Gabriel graphs [18] and relative neighborhood (RNG) graphs [17, 19]. In particular, Wan and Yi [18] show that for n points uniformly distributed in a unit-area disk, the ratio of the length of a longest Gabriel edge to $\sqrt{(\ln n)/(\pi n)}$ is asymptotically almost surely equal to 2, and the expected number of "long" Gabriel edges, of length at least $2\sqrt{(\ln n + \xi)/(\pi n)}$, is asymptotically almost surely equal to $2e^{-\xi}$, for any fixed ξ . In [9], while studying the maximum degree of Gabriel and Yao graphs, the authors observe that the probability that the maximum edge length is greater than $3\sqrt{(\log n)/n}$ tends to zero, bound that they claim becomes $O(((\log n)/n)^{1/d})$ for d dimensions. An overview of related problems can be found in [1].

Interest in bounding the length of a longest Delaunay edge in two-dimensional spaces has grown out of extensive algorithmic work [6, 4, 10] aimed at reducing the energy consumption of geographically routing messages in Radio Networks. Multidimensional Delaunay graphs

^{*}This research was partially supported by Spanish MICINN grant TIN2008-06735-C02-01, Comunidad de Madrid grant S2009TIC-1692, EU Marie Curie International Reintegration Grant IRG 210021, and the National Science Foundation (CCF-0937829, CCF-1018388). An earlier version of this work has been presented in [2].

[†]Department of Applied Mathematics and Statistics, Stony Brook University, USA, esther.arkin@stonybrook.edu

[‡]Institute IMDEA Networks, Madrid, Spain,

[§]Department of Applied Mathematics and Statistics, Stony Brook University, USA, joseph.mitchell@stonybrook.edu

[¶]Computer Science Department, Rutgers University, USA, and LADyR, GSyC, Universidad Rey Juan Carlos, Spain, mosteiro@cs.rutgers.edu

¹A set of d+1 points in *d*-dimensional Euclidean space is said to be *in general position* if no hyperplane contains all of them. We say that such a set is *generic*, or *degenerate* otherwise.

are well studied in computational geometry from the point of view of efficient algorithms to construct them (see [8] and references therein), but only limited results are known regarding probabilistic analysis of Delaunay graphs in higher dimensions [14].

Overview of Our Results. We study the probabilistic length of longest Delaunay edges for points distributed in geometric domains in two and more dimensions. Since the length of the longest Delaunay edge is strongly influenced by the boundary of the enclosing region, we study the problem for two cases, which we call *with boundary* and *without boundary*.

Our results include upper and lower bounds for ddimensional bodies with and without boundaries, that hold for a parametric error probability ε and are computed up to the constant factors (they are tight only asymptotically). In comparison, the upper bounds presented in [11] are only asymptotic, are restricted to two dimensions (d = 2), and apply to domains with boundary (disks), although results without boundary are implicitly given, since the results are parametric in the distance to the boundary.

Lower bounds without boundary and all upper bounds apply for any d > 1. Lower bounds with boundary are shown for $d \in \{2, 3\}$. The results shown are asymptotically tight for $e^{-cn} \leq \varepsilon \leq n^{-c}$, for any constant c > 0, and $d \in O(1)$. To the best of our knowledge, this is the first comprehensive study of this problem. The results obtained are summarized in Table 1. In order to compare upper and lower bounds for bodies with boundary, it is crucial to notice that we bound the volume of a circular segment (2D) and the volume of an spherical cap (3D), which can be approximated by polynomials of third and fourth degree respectively on the diameter of the base. Upper bounds are proved exploiting the fact that, thanks to the uniform density, it is very unlikely that a "large" volume is void of points. Lower bounds, on the other hand, are proved by showing that a configuration that yields a Delaunay edge of a certain length is not very unlikely.

In the following section, some necessary notation is introduced. Upper and lower bounds for enclosing bodies without boundaries are shown in Section 3, and the case with boundaries is covered in Section 4. We conclude with some open problems.

2 Preliminaries

The following notation will be used throughout. We will restrict attention to Euclidean (L_2) spaces. A *d*-sphere, $S = S_{r,c}$, of radius r is the set of all points in a (d + 1)-dimensional space that are located at distance r (the radius) from a given point c (the center). A *d*-ball, $B = B_{r,c}$, of radius r is the set of all points in

a d-dimensional space that are located at distance at most r (the radius) from a given point c (the center). The area of a d-sphere S (in (d + 1)-space) is its d-dimensional volume. The volume of a d-ball B (in d-space) is its d-dimensional volume. We refer to a unit sphere as a sphere of area 1 and a unit ball as a ball of volume 1. (This is in contrast with some definitions of a "unit" ball/sphere as a unit-radius ball/sphere; we find it convenient to standardize the volume/area to be 1 in all dimensions.)

Let P be a set of points on a d-sphere, S. Given two points $a, b \in P$, let ab be the arc of a great circle between them. Let $\delta(a, b)$ be the length of the arc ab, which is also known as the *orthodromic distance* between a and b on the sphere S. Let the orthodromic diameter of a subset $X \subseteq S$ be the greatest orthodromic distance between a pair of points in X. A spherical cap on S is the set of all points at orthodromic distance at most rfrom some center point $c \in S$. Let $A_d(x)$ be the area (d-volume) of a spherical cap of orthodromic diameter x, on a d-sphere of surface area 1. A ball cap of B is the intersection of a d-ball B with a closed halfspace, bounded by a hyperplane h, in *d*-space; the *base* of a ball cap is the (d-1)-ball that is the intersection of h with the ball B. Let $V_d(x)$ be the d-volume of a ball cap of base diameter x, of a d-ball of volume 1. For any pair of points a, b, let d(a, b) be the Euclidean distance between a and b, i.e. $d(a,b) = ||\overline{ab}||_2$. Let D(P) be the Delaunay graph of a set of points P.

The following definitions of a Delaunay graph, D(P), of a finite set P of points in d-dimensional bodies follow the standard definitions of Delaunay graphs (see, e.g., Theorem 9.6 in [8]).

Definition 1 Let P be a generic set of points on a d-sphere S.

- (i) A set $F \subseteq P$ of d + 1 points define the vertices of a Delaunay face of D(P) if and only if there is a d-dimensional spherical cap $C \subset S$ such that F is contained in the boundary, ∂C , of C and no points of P lie in the interior of C (relative to the sphere S).
- (ii) Two points $a, b \in P$ form a Delaunay edge, an arc of D(P), if and only if there is a d-dimensional spherical cap C such that $a, b \in \partial C$ and no points of P lie in the interior of C (relative to the sphere S).

Definition 2 Let P be a generic set of points in a d-ball B.

(i) A set $F \subseteq P$ of d + 1 points define the vertices of a Delaunay face of D(P) if and only if there is a d-ball B' such that F is contained in the boundary, $\partial B'$, of B' and no points of P lie in the interior of B'.

	d	Upper Bound:	Lower Bound:
		w.p. $\geq 1 - \varepsilon, \nexists \widehat{ab} \in D(P)$	w.p. $\geq \varepsilon, \exists \ \widehat{ab} \in D(P)$
Without boundary	d	$A_d(\delta(a,b)) \ge \frac{\ln\left(\binom{n}{2}\binom{n-2}{d-1}/\varepsilon\right)}{n-d-1}$	$A_d(\delta(a,b)) \ge \frac{\ln((e-1)/(e^2\varepsilon))}{n-2+\ln((e-1)/(e^2\varepsilon))}$
	1	$\delta(a,b) \ge \frac{\ln\left(\binom{n}{2}/\varepsilon\right)}{n-2}$	$\delta(a,b) \geq \frac{\ln\left((e-1)/(e^2\varepsilon)\right)}{n-2 + \ln\left((e-1)/(e^2\varepsilon)\right)}$
	2	$\delta(a,b) \ge \frac{\cos^{-1}\left(1 - \frac{2\ln\left(\binom{n}{2}(n-2)/\varepsilon\right)}{n-3}\right)}{\sqrt{\pi}}$	$\delta(a,b) \geq \frac{\cos^{-1}\left(1 - \frac{2\ln\left((e-1)/(e^2\varepsilon)\right)}{n-2 + \ln\left((e-1)/(e^2\varepsilon)\right)}\right)}{\sqrt{\pi}}$
With boundary	d	$V_d(d(a,b)) \ge \frac{\ln\left(\binom{n}{2}\binom{n-2}{d-1}/\varepsilon\right)}{n-d-1}$	_
	2	$d(a,b) \ge \sqrt[3]{rac{16}{\sqrt{\pi}}} rac{\ln\left(\binom{n}{2}(n-2)/\varepsilon\right)}{n-3}$	$d(a,b) \ge \rho_2/2 : V_2(\rho_2) = \frac{\ln(\alpha_2/\varepsilon)}{(n-2+\ln(\alpha_2/\varepsilon))}$ $\implies d(a,b) \ge \sqrt[3]{\frac{\ln(\alpha/\varepsilon)}{2\sqrt{\pi}(n-2+\ln(\alpha/\varepsilon))}}$
	3	$d(a,b) \ge \sqrt[4]{\frac{96}{\pi^{3/2}} \frac{\ln\left(\binom{n}{2}\binom{n-2}{2}/\varepsilon\right)}{n-4}}$	$d(a,b) \ge \rho_3/2 : V_3(\rho_3) = \frac{\ln(\alpha_3/\varepsilon)}{(n-2+\ln(\alpha_3/\varepsilon))}$ $\implies d(a,b) \ge \sqrt[4]{\sqrt[3]{\frac{48}{\pi^4}} \frac{\ln(\alpha_3/\varepsilon)}{(n-2+\ln(\alpha_3/\varepsilon))}}$

Table 1: Summary of results. α_2, α_3 are constants.

(ii) Two points $a, b \in P$ form a Delaunay edge, an arc of D(P), if and only if there is a d-ball B' such that $a, b \in \partial B'$ and no points of P lie in the interior of B'.

The following inequalities [15] are used throughout

$$e^{-x/(1-x)} \le 1 - x \le e^{-x}$$
, for $0 < x < 1$. (1)

3 Enclosing Body without Boundary

The following theorems show upper and lower bounds on the length of arcs in the Delaunay graph on a dsphere.

3.1 Upper Bound

Theorem 3 Consider the Delaunay graph D(P) of a set P of $n > d + 1 \ge 2$ points distributed uniformly and independently at random in a unit d-sphere, S. Then, for $0 < \varepsilon < 1$, the probability is at least $1 - \varepsilon$ that there is no arc $\hat{ab} \in D(P)$, $a, b \in P$, such that

$$A_d(\delta(a,b)) \ge \frac{\ln\left(\binom{n}{2}\binom{n-2}{d-1}/\varepsilon\right)}{n-d-1}.$$
 (*)

Proof. Let E_{ε} be the event that "there exists an arc $\widehat{ab} \in D(P)$, $a, b \in P$, with inequality (*) satisfied" Our goal is to prove that $P(E_{\varepsilon}) \leq \varepsilon$.

Let us consider a fixed pair of points, $a, b \in P$. We let $E_{a,b}$ be the event that $\widehat{ab} \in D(P)$. For any subset $Q \subset P$ of d + 1 points containing a and b, let C_Q denote the spherical cap through Q and let F_Q denote the event that the interior of C_Q contains no points of P (i.e., $int(C_Q) \cap P = \emptyset$).

Thus, we can write $E_{a,b} = \bigcup_Q F_Q$ as the union, over all $\binom{n-2}{(d+1)-2} = \binom{n-2}{d-1}$ subsets $Q \subset P$ with |Q| = d+1and $a, b \in Q$, of the events F_Q . Then, by the union bound, we know that $P(E_{a,b}) \leq \sum_Q P(F_Q)$. Further, in order for event F_Q to occur, all points of P except the d+1 points of Q must lie outside the spherical cap C_Q through Q; thus, $P(F_Q) = (1 - \mu_d(C_Q))^{n-(d+1)}$, where $\mu_d(C_Q)$ denotes the d-volume of C_Q .

We see that $P(F_Q) \leq (1 - A_d(\delta(a, b)))^{n-(d+1)}$, since, for any subset $Q \supset \{a, b\}$, the *d*-volume $\mu_d(C_Q)$ is at least as large as the *d*-volume, $A_d(\delta(a, b))$, of the spherical cap having orthodromic diameter $\delta(a, b)$. (In other words, $A_d(\delta(a, b))$ is the *d*-volume of the smallest volume spherical cap whose boundary passes through *a* and *b*.)

Altogether, we get

$$P(E_{a,b}) \le \sum_{Q} P(F_Q) = \sum_{Q} (1 - \mu_d(C_Q))^{n - (d+1)}$$
$$\le \binom{n-2}{d-1} (1 - A_d(\delta(a,b)))^{n - (d+1)}.$$

Now, the event of interest is

$$E_{\varepsilon} = \bigcup_{a,b \in P:(*) \text{ holds}} E_{a,b}.$$

The inequality (*) is equivalent to

$$(n-d-1)A_d(\delta(a,b)) \ge \ln\left(\binom{n}{2}\binom{n-2}{d-1}/\varepsilon\right)$$

which is equivalent to

$$\left(e^{-A_d(\delta(a,b))}\right)^{(n-d-1)} \le \frac{\varepsilon}{\binom{n}{2}\binom{n-2}{d-1}}$$

Since, by Inequality 1, $e^{-x} \ge 1-x$, the above inequality implies that

$$\left(1 - A_d(\delta(a, b))\right)^{(n-d-1)} \le \frac{\varepsilon}{\binom{n}{2}\binom{n-2}{d-1}},$$

which implies that

$$\binom{n}{2}\binom{n-2}{d-1}\left(1-A_d(\delta(a,b))\right)^{(n-d-1)} \le \varepsilon.$$

Using the union bound, we get

$$P(E_{\varepsilon}) = P\left(\bigcup_{a,b\in P:(*) \text{ holds}} E_{a,b}\right) \le \sum_{a,b\in P:(*) \text{ holds}} P(E_{a,b})$$

Since each term $P(E_{a,b})$ in the above summation is bounded above by $\binom{n-2}{d-1}(1 - A_d(\delta(a,b)))^{n-(d+1)}$, and there are at most $\binom{n}{2}$ terms in the summation, we get

$$P(E_{\varepsilon}) \leq \sum_{a,b \in P:(*) \text{ holds}} P(E_{a,b})$$

$$\leq {\binom{n}{2}} {\binom{n-2}{d-1}} \left(1 - A_d(\delta(a,b))\right)^{(n-d-1)} \leq \varepsilon.$$

The following corollaries for d = 1 and d = 2 can be obtained from Theorem 3 using the corresponding surface areas.

Corollary 4 In the Delaunay graph D(P) of a set Pof n > 2 points distributed uniformly and independently at random on a unit circle (1-sphere), with probability at least $1 - \varepsilon$, for $0 < \varepsilon < 1$, there is no arc $\hat{ab} \in D(P)$, $a, b \in P$, such that

$$\delta(a,b) \ge \frac{\ln\left(\binom{n}{2}/\varepsilon\right)}{n-2}$$

Corollary 5 In the Delaunay graph D(P) of a set P of n > 3 points distributed uniformly and independently at random on a unit sphere (2-sphere), with probability

at least $1 - \varepsilon$, for $0 < \varepsilon < 1$, there is no arc $\widehat{ab} \in D(P)$, $a, b \in P$, such that

$$\delta(a,b) \ge \frac{1}{\sqrt{\pi}} \cos^{-1} \left(1 - \frac{2 \ln\left(\binom{n}{2}(n-2)/\varepsilon\right)}{n-3} \right).$$

Proof. The surface area of a spherical cap of a 2-sphere is $2\pi Rh$, where R is the radius of the sphere and h is the height of the cap. For a unit 2-sphere is $R = 1/(2\sqrt{\pi})$. Then, the perimeter of a great circle is $2\pi/(2\sqrt{\pi}) = \sqrt{\pi}$. Thus, the central angle of a cap whose orthodromic diameter is ρ is $2\pi \rho/\sqrt{\pi} = 2\sqrt{\pi}\rho$. Let the angle between the line segment \overline{ab} and the radius of the sphere be α . Then,

$$\alpha = \begin{cases} \pi/2 - \sqrt{\pi}\rho & \text{if } \rho \le \sqrt{\pi}/2\\ \sqrt{\pi}\rho - \pi/2 & \text{if } \rho > \sqrt{\pi}/2 \end{cases}$$

And the height of the cap is $h = 1/(2\sqrt{\pi}) - 1/(2\sqrt{\pi})\sin(\pi/2 - \sqrt{\pi}\rho) = (1 - \cos(\sqrt{\pi}\rho))/(2\sqrt{\pi})$. Therefore, the surface area of a spherical cap of a 2-sphere whose orthodromic diameter is ρ is $(1 - \cos(\sqrt{\pi}\rho))/2$. Replacing in Theorem 3, the claim follows.

3.2 Lower Bound

Theorem 6 In the Delaunay graph D(P) of a set Pof n > 2 points distributed uniformly and independently at random in a unit d-sphere, with probability at least ε , there is an arc $\widehat{ab} \in D(P)$, $a, b \in P$, such that $A_d(\delta(a, b)) \ge A_d(\rho_1)$, where

$$A_d(\rho_1) = \frac{\ln\left((e-1)/(e^2\varepsilon)\right)}{n-2+\ln\left((e-1)/(e^2\varepsilon)\right)},$$

for any $0 < \varepsilon < 1$ such that $A_d(2\rho_1) \le 1 - 1/(n-1)$.

Proof. In order to prove this claim, we consider a configuration given by a specific pair of points and a specific empty spherical cap circumscribing them, that would yield a Delaunay arc between those points. Then, we relate the probability of existence of such configuration to the distance between the points. Finally, we relate this quantity to the desired parametric probability. The details follow.

For any pair of points $a, b \in P$, by Definition 1, for the arc \widehat{ab} to be in D(P), there must exist a *d*-dimensional spherical cap C such that a and b are located on the boundary of the cap base and the cap surface of C is void of points from P. We compute the probability of such an event as follows. Let $\rho_2 > \rho_1$ be such that $A_d(2\rho_2) - A_d(2\rho_1) = 1/(n-1)$. Consider any point $a \in P$. The probability that some other point b is located so that $\rho_1 < \delta(a, b) \leq \rho_2$ is $1 - (1 - 1/(n-1))^{n-1} \geq 1 - 1/e$, by Inequality 1.

The spherical cap with orthodromic diameter $\delta(a, b)$ is empty with probability $(1 - A_d(\delta(a, b)))^{n-2}$. To lower bound this probability we consider separately the spherical cap with orthodromic diameter ρ_1 and the remaining annulus of the spherical cap with orthodromic diameter $\delta(a, b)$. The probability that the latter is empty is lower bounded by upper bounding the area $A_d(\delta(a, b)) - A_d(\rho_1) \leq A_d(2\rho_2) - A_d(2\rho_1) = 1/(n-1)$. Then, $(1 - 1/(n-1))^{n-2} \geq 1/e$, by Inequality 1.

Finally, the probability that the spherical cap with orthodromic diameter ρ_1 is empty is, by Inequality 1,

$$(1 - A_d(\rho_1))^{n-2} \ge \exp\left(-\frac{A_d(\rho_1)(n-2)}{1 - A_d(\rho_1)}\right),$$
$$= \exp\left(-\ln\left(\frac{e-1}{e^2\varepsilon}\right)\right) = \frac{e^2\varepsilon}{e-1}.$$

Therefore,

$$Pr\left(\widehat{ab} \in D(P)\right) \ge \left(1 - \frac{1}{e}\right) \frac{1}{e} \frac{e^2\varepsilon}{e - 1} = \varepsilon.$$

The following corollaries for d = 1 and d = 2 can be obtained from Theorem 6 using the corresponding surface areas.

Corollary 7 In the Delaunay graph D(P) of a set Pof n > 2 points distributed uniformly and independently at random in a unit circle (1-sphere), with probability at least ε , for any $e^{1-n-4/n} \le \varepsilon < 1$, there is an arc $ab \in D(P)$, $a, b \in P$, such that

$$\delta(a,b) \ge \frac{\ln\left((e-1)/(e^2\varepsilon)\right)}{n-2+\ln\left((e-1)/(e^2\varepsilon)\right)}$$

Corollary 8 In the Delaunay graph D(P) of a set Pof n > 2 points distributed uniformly and independently at random in a unit sphere (2-sphere), with probability at least ε , for any $e^{-n+2\sqrt{n-1}-1} \le \varepsilon < 1$, there is an arc $\widehat{ab} \in D(P)$, $a, b \in P$, such that

$$\delta(a,b) \ge \frac{1}{\sqrt{\pi}} \cos^{-1} \left(1 - \frac{2\ln((e-1)/(e^2\varepsilon))}{n-2 + \ln((e-1)/(e^2\varepsilon))} \right).$$

Proof. As shown in the proof of Corollary 5, the surface area of a spherical cap of a 2-sphere whose orthodromic diameter is ρ is $(1 - \cos(\sqrt{\pi}\rho))/2$. Replacing in Theorem 6, the claim follows.

4 Enclosing Body with Boundary

The following theorems show upper and lower bounds on the length of edges in the Delaunay graph in a *d*ball. The proofs, omitted here for brevity, can be found in the full version of this work [3].

4.1 Upper Bound

Theorem 9 In the Delaunay graph D(P) of a set P of $n > d + 1 \ge 2$ points distributed uniformly and independently at random in a unit d-ball, with probability at least $1-\varepsilon$, for $0 < \varepsilon < 1$, there is no edge $(a, b) \in D(P)$, $a, b \in P$, such that

$$V_d(d(a,b)) \ge \frac{\ln\left(\binom{n}{2}\binom{n-2}{d-1}/\varepsilon\right)}{n-d-1}$$

The following corollaries for d = 2 and d = 3 can be obtained from Theorem 9 using the corresponding volumes.

Corollary 10 In the Delaunay graph D(P) of a set P of n > 3 points distributed uniformly and independently at random in a unit disk (2-ball), with probability at least $1 - \varepsilon$, for $\binom{n}{2}(n-2)e^{-\sqrt{2}(n-3)/\pi} < \varepsilon < 1$, there is no edge $(a, b) \in D(P)$, $a, b \in P$, such that

$$d(a,b) \ge \sqrt[3]{\frac{16}{\sqrt{\pi}} \frac{\ln\left(\binom{n}{2}(n-2)/\varepsilon\right)}{n-3}}$$

Corollary 11 In the Delaunay graph D(P) of a set Pof n > 4 points distributed uniformly and independently at random in a unit ball (3-ball), with probability at least $1 - \varepsilon$, for $\binom{n}{2}\binom{n-2}{2}e^{-2(n-4)/(3\sqrt{\pi})} < \varepsilon < 1$, there is no edge $(a,b) \in D(P)$, $a, b \in P$, such that

$$d(a,b) \ge \sqrt[4]{\frac{96}{\pi^{3/2}}} \frac{\ln\left(\binom{n}{2}\binom{n-2}{2}/\varepsilon\right)}{n-4}.$$

4.2 Lower Bound

In this section we give lower bounds for d = 2 and d = 3. As in the case without boundary, we prove our lower bounds showing a configuration given by a specific pair of points and a specific empty body circumscribing them, that would yield a Delaunay edge between those points. Then, we relate the probability of existence of such configuration to the distance between the points and to the desired parametric probability.

Theorem 12 For d = 2, given the Delaunay graph D(P) of a set P of n > 2 points distributed uniformly and independently at random in a unit d-ball, with probability at least ε , there is an edge $(a, b) \in D(P)$, $a, b \in P$, such that $d(a, b) \ge \rho_1/2$, where

$$V_d(\rho_1) = rac{\ln(\alpha/\varepsilon)}{(n-2+\ln(\alpha/\varepsilon))}$$

where $\alpha = (1 - e^{-1/16})(1 - e^{-1/32})e^{-1}$, for any $0 < \varepsilon \le \alpha/e^2$ such that $V_d(\rho_1) \le 1/2 - 1/n$. Which implies that

$$d(a,b) \ge \sqrt[3]{rac{\ln(\alpha/\varepsilon)}{2\sqrt{\pi}(n-2+\ln(\alpha/\varepsilon))}}.$$

Theorem 13 For d = 3, given the Delaunay graph D(P) of a set P of n > 4 points distributed uniformly and independently at random in a unit d-ball, with probability at least ε , there is an edge $(a,b) \in D(P)$, $a, b \in P$, such that $d(a,b) \ge \rho_1/2$, where

$$V_d(\rho_1) = \frac{\ln\left(\alpha/\varepsilon\right)}{\left(n - 2 + \ln\left(\alpha/\varepsilon\right)\right)}$$

where $\alpha = (1 - e^{-1/6})(1 - e^{-1/12})e^{-12}$, for any $0 < \varepsilon \le \alpha/e$ such that $V_d(\rho_1) \le 1/2 - 1/n$. Which implies that

$$d(a,b) \ge \sqrt[4]{\sqrt[3]{\frac{48}{\pi^4}}} \frac{\ln(\alpha/\varepsilon)}{(n-2+\ln(\alpha/\varepsilon))}$$

5 Future Directions, Open Problems

It would be interesting to extend this study to other norms, such as L_1 or L_{∞} . Also, Theorems 12 and 13 were proved by showing that the existence of a configuration that yields a Delaunay edge of some length is not unlikely. Different configurations were used for each, but a configuration that works for both cases exists (although yielding worse constants). We conjecture that (modulo some constant) the same bound can be obtained in general for any d > 1. Both questions are left for future work.

References

- D. Aldous and S. J. Connected spatial networks over random points and a route-length statistic. *Statistical Science*, 25(3):275–288, 2010.
- [2] E. M. Arkin, A. Fernández Anta, J. S. B. Mitchell, and M. Mosteiro. The length of the longest edge in multidimensional delaunay graphs (extended abstract). In *Proceedings of the 20th Annual Fall Workshop on Computational Geometry*, 2010.
- [3] E. M. Arkin, A. Fernández Anta, J. S. B. Mitchell, and M. Mosteiro. Probabilistic Bounds on the Length of a Longest Edge in Delaunay Graphs of Random Points in *d*-Dimensions. arXiv:1106.4927v1 [cs.CG], June 2011.
- [4] C. Avin. Fast and efficient restricted Delaunay triangulation in random geometric graphs. *Internet Mathematics*, 5(3):195–210, 2008.
- [5] M. Bern, D. Eppstein, and F. Yao. The expected extremes in a delaunay triangulation. *International Journal of Computational Geometry and Applications*, 1(1):79–91, 1991.
- [6] P. Bose, P. Carmi, M. H. M. Smid, and D. Xu. Communication-efficient construction of the plane localized Delaunay graph. In Proc. of the 9th Latin American Theoretical Informatics Symposium, pages 282– 293, 2010.

- [7] P. Bose and P. Morin. Online Routing in Triangulations. In Proc. of the 10th International Symposium on Algorithms and Computation, page 113. Springer Verlag, 1999.
- [8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. Springer-Verlag, second edition, 2000.
- [9] L. Devroye, J. Gudmundsson, and M. P. On the Expected Maximum Degree of Gabriel and Yao Graphs. arXiv:0905.3584v1 [cs.CG], May 2009.
- [10] A. F. and L. Rodrigues. Single-step creation of localized Delaunay triangulations. Wireless Networks, 15(7):845– 858, 2009.
- [11] G. Kozma, Z. Lotker, M. Sharir, and G. Stupp. Geometrically aware communication in random wireless networks. In Proc. 23rd Ann. ACM Symp. on Principles of Distributed Computing, pages 310–319, 2004.
- [12] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In Proc. of the 11th Canadian Conference on Computational Geometry, 1999.
- [13] E. Lebhar and Z. Lotker. Unit disk graph and physical interference model: Putting pieces together. In Proc. of the 23rd International Symposium on Parallel & Distributed Processing, pages 1–8. IEEE, 2009.
- [14] C. Lemaire and J. Moreau. A probabilistic result on multi-dimensional Delaunay triangulations, and its application to the 2D case. *Comput. Geom.*, 17(1-2):69– 96, 2000.
- [15] D. S. Mitrinović. *Elementary Inequalities*. P. Noordhoff Ltd. - Groningen, 1964.
- [16] M. Penrose. Random Geometric Graphs. Oxford University Press, May 2003.
- [17] P. Wan, L. Wang, F. Yao, and C. Yi. On the Longest RNG Edge of Wireless Ad Hoc Networks. In Proc. of the 28th International Conference on Distributed Computing Systems, pages 329–336. IEEE, 2008.
- [18] P. Wan and C. Yi. On the longest edge of gabriel graphs in wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 111–125, 2007.
- [19] C. Yi, P. Wan, L. Wang, and C. Su. Sharp thresholds for relative neighborhood graphs in wireless Ad Hoc networks. *IEEE Transactions on Wireless Communications*, 9(2):614–623, 2010.

Outerplanar graphs and Delaunay triangulations

Ashraful Alam *

Igor Rivin^{\dagger}

Ileana Streinu[‡]

Abstract

Over 20 years ago, Dillencourt [1] showed that all outerplanar graphs can be realized as Delaunay triangulations of points in convex position. His proof is elementary, constructive and leads to a simple algorithm for obtaining a concrete Delaunay realization. In this note, we provide two new, alternate, also quite elementary proofs.

1 Introduction

The Delaunay triangulation of a point set in convex position is, combinatorially, an outerplanar graph. Dillencourt [1] has shown, constructively, that the other direction is also true: any graph which arises from a triangulation of the interior of a simple polygon can be realized as a Delaunay triangulation. Dillencourt's proof uses a simple and natural criterion on the angles of triangles in a Delaunay triangulation, and gives an $O(n^2)$ time incremental algorithm to calculate their angles and infer a realization. Lambert [2] adapted this method into a linear time algorithm, whose implementation in Java is available at http://www.cse.unsw.edu.au/l̃ambert/java/realize/

The general question, of characterizing and reconstructing arbitrary Delaunay triangulations (in twoor higher dimensions), is substantially more difficult. A closely related problem, going back to Steiner (see Grünbaum [6], page 284), asks for a characterization of the graphs of inscribable or circumscribable polyhedra: those whose vertices lie on a sphere, resp. whose faces are tangent to a sphere. Such graphs are said to be of inscribable or circumscribable type. The best result to date is due to Rivin [4], who proved necessary and sufficient conditions for a polyhedral graph to be of inscribable or circumscribable type. Dillencourt and Smith [3] linked inscribability of a graph to its realizability as a Delaunay triangulation, and gave a criterion relating Hamiltonicity to inscribability. In this paper, we present two new simple and elementary proofs of the Delaunay realizability of outerplanar graphs. We are not aware of them previously appearing in the literature. The first one is an easy consequence of Dillencourt and Smith's [3] criterion relating Hamiltonianicity and inscribability. The second one, which occupies most of this note, uses Rivin's [4] inscribability criterion and constructs an explicit "witness" of this inscribability, in the form of certain weights assigned to the edges of the graph.

Preliminaries

A graph G = (V, E) is a collection of vertices V = $\{1, \dots, n\}$ and edges E, where an edge $e = ij \in E$ is a pair of vertices $i, j \in V$. A graph is *planar* if it can be drawn in the plane in a way such that no two edges cross, except perhaps at endpoints. A planar drawing of a planar graph is called a *plane graph*. It subdivides the plane into regions called faces. A plane graph has one unbounded face, called the *outer face*. A plane graph is denoted by its vertices, edges, faces and the outer face: G = (V, E, F, f). A plane graph where *all* vertices lie on the outer face is called an *outerplanar graph*. A stellated outerplanar graph is obtained from an outerplanar graph by adding one vertex and connecting it to all the original vertices. We call this the *stellating* vertex, and all edges emanating from this vertex the stellating edges.

Two paths between two vertices are *independent* if they do not share any vertices except the end-points. A graph is connected if there is a path between any two vertices and it is *k*-connected if there are *k* independent paths between any two vertices. A cutset of a graph is the minimal set of edges whose removal makes the graph disconnected. A cutset is coterminous if all the edges emanate from a single point.

A graph is *polyhedral* if it is planar and 3-connected. In this case, the faces of a plane realization are uniquely determined up to the choice of the outer face. By Steinitz theorem (see Grünbaum [6]), any polyhedral graph can be realized as a convex polyhedron. A polyhedral graph is *inscribable* if its corresponding convex polyhedron is combinatorially equivalent to the edges and vertices of the convex hull of a set of noncoplanar points on the surface of the sphere.

Given a set P of points in the Euclidean plane, a triangulation of these points is a planar graph where all

^{*}Department of Computer Science, University of Massachusetts Amherst, MA, ashraful@cs.umass.edu. Research supported by NSF CCF-1016988 and DARPA HR0011-09-0003 grants.

[†]Department of Mathematics, Temple University, Philadelphia, rivin@math.temple.edu

[‡]Department of Computer Science, Smith College, Northampton, MA, streinu@smith.edu. Research supported by NSF CCF-1016988 and DARPA HR0011-09-0003 grants.

faces, with the possible exception of the outer face, are triangles. A Delaunay triangulation of P is a triangulation where the circumscircle of any triangle does not contain any other points of P.

Our result

We give two new proofs of Dillencourt's theorem:

Theorem 1 Any outerplanar graph can be realized as a Delaunay triangulation.

2 The first proof

The first proof that an outerplanar graph can be realized as a Delaunay triangulation relies on two elegant results due to Dillencourt and Smith [3] and to Rivin [4, 5]. They relate inscribability, realization as Delauney triangulation and Hamiltonicity.

A Hamiltonian cycle in a graph is a simple spanning cycle. Any graph that has a Hamiltonian cycle is called *Hamiltonian*. A graph is *1-Hamiltonian* if removing any vertex from the graph makes it Hamiltonian.

Dillencourt and Smith [3] proved:

Theorem 2 A 1-Hamiltonian planar graph is of inscribable type.

Next, we need this result of Rivin [4, 5].

Theorem 3 A plane graph G = (V, E, F, f), with f as the unbounded face, is realizable as a Delaunay triangulation for some point set P if and only if the graph G'obtained from G by stellating f is of inscribable type.

To complete our first proof, we just need to show that:

Lemma 4 A stellated outerplanar graph G' is 1-Hamiltonian.

Proof. Let $\{1, 2, \dots, n\}$ be the vertices of the underlying outerplanar graph G of G', in counterclockwise order on the outer face, with modulo n indices, and let s be the stellating vertex. If we remove a vertex $i, 1 \leq i \leq n$, then we find a Hamiltonian cycle $i+1, i+2, \dots, i-1, s, i+1$. If we remove vertex s, we get the original outerplanar graph G which is Hamiltonian.

In the next section, we present our main result, which is a more technical proof for the same result. It is based on a very general criterion of Rivin, and has the advantage of illustrating specific properties (besides Hamiltonicity) of Delaunay triangulation realizations for outerplanar graphs.

3 The main proof

Rivin [4, 5] gave this very general criterion for the Steiner's problem:

Theorem 5 A planar graph G' = (V, E) is of inscribable type if and only if it satisfies the following conditions:

- 1. G' is a 3-connected planar graph.
- 2. A set of weights W can be assigned to the edges of G' such that:
 - (a) For each edge $e, 0 < w(e) \le 1/2$.
 - (b) For each vertex v, the sum of all weights of edges incident to v is 1.
 - (c) For each non-coterminous cutset $C \subseteq E$, the sum of all the weights of edges of C must exceed 1.

Combining this with Theorem 3, we have to prove that if G' is a stellated outerplanar graph, then a weight assignment as in Theorem 5 exists. But first, let us verify that any stellated outerplanar graph is 3-connected. The following lemma is straightforward:

Lemma 6 Any outerplanar graph is 2-connected.

Proof. In an outerplanar graph, all the vertices lie on the unbounded face f. If we label the vertices as $1, 2, \dots, n$ in the order in which they appear on the outer face, there are two independent paths between any pair of vertices i and j: one from $i, i + 1, \dots, j$ and another is $i, i - 1, \dots, j$.

This leads immediately to the verification of the first condition in Theorem 5:

Lemma 7 Any stellated outerplanar graph G' is planar and 3-connected.

Proof. Planarity is straightforward, since G' was obtained from an outerplanar graph G by stellating (with a new vertex s) its unbounded face. We show now that there exists three independent paths between any two vertices i and j of G'. Let i and j be two vertices of the outerplanar graph G. Lemma 6 showed that there are two independent paths between i and j. A third independent path is i, s, j where s is the stellating vertex. To complete the proof we show the existence of three independent paths between s and any other vertex i of G: they are (s, i), (s, i - 1, i) and (s, i + 1, i), where index arithmetic is done modulo n in the range $1, \dots, n$. Notice that we implicitly assume that G has at least three vertices, otherwise the theorem is trivial.

In the rest of the paper, we describe a weight assignment for G' which satisfies Rivin's criteria. In section 3.1 we give an inductive scheme to compute the weights, and prove that they satisfy the first two properties (2a and 2b) in Theorem 5. The proof is completed in section 3.2, where we verify the third property (2c) in Theorem 5.

3.1 Weight assignment

Instead of assigning weights on the edges of a stellated outerplanar graph G', we will assign them on the edges of the dual graph G'_D of G', so first we look closer at the structure of the dual of a stellated outerplanar graph.

The duals of the stellating edges of G' form a cycle, and the remaining ones form a tree (denoted by T_D) whose leaves lie on the cycle (see Fig 1). Furthermore, the tree is partitioned into a path whose vertices are not leaves (the *backbone*) and edges incident to the tree leaves, called leaf-edges. We thus partition the edges of G' into three classes, colored blue (cycle), green (backbone) and black (leaf edges). Primal and dual edges get the same color, see Fig 1(c). One end of the leaf edge is connected to a backbone edge and another end is connected to a cycle edge.

The edges of a face of G'_D are duals of edges incident to a vertex in G'. If there are *n* cycle edges, there will be *n* faces in G'_D . Let these faces be f_1, f_2, \dots, f_n in counter-clockwise order. Clearly, it suffices to make sure that the sum of all weights of cycle edges and sum of all weights of the edges of each face of G'_D are separately equal to 1. In addition, we have to make sure that the remaining two conditions 2a and 2b of Theorem 5 are also satisfied.



Figure 1: (a) An outerplanar graph (b) A stellated outerplanar graph obtained from (a). Blue edges are stellating edges and blue vertex is the stellating vertex. (c) Dual graph of the stellated outerplanar graph. Dual edges are colored according to their primal edges. Dark blue, green and black edges are cycle, backbone and leaf edges respectively.

The weight assignment is carried out in two steps, the *contraction step* and the *expansion step*. During contraction, all the backbone edges are contracted to obtain a very specific type of dual graph, on which a simple weight assignment is possible. The edges are then expanded back, and adjustments to the initial weights are locally performed, while maintaining Rivin's conditions.

Contraction:

In this step we contract all the backbone edges of T_D . Then all the cycle edges and leaf edges of G'_D remain unchanged, but all the faces become triangular (see Fig 2). Next, we assign a weight of 1/n to each cycle edge. Here n is the number of cycle edges of G'_D . Next we assign each leaf edge (one of the remaining two edges of a triangular face) a weight of $\frac{n-1}{2n}$. This weight assignment satisfies Rivin's conditions 2a and 2b for n > 1.



Figure 2: (a)A dual of a stellated outerplanar graph. Bold edges are backbone edges. (b) Dual graph after contraction of backbone edges. (c) Expansion of a single backbone edge.

Expansion:

Now we incrementally expand back the backbone edges of T_D . Consider a backbone edge e_b which is shared by face f_i and face f_j . We assign a weight of $0 < \epsilon < 1/2$ to e_b , for some positive ϵ to be determined later. This creates an imbalance into the sum of weights for the edges of faces f_i and f_j . We remove this imbalance by subtracting $\frac{\epsilon}{2}$ from the cycle edges of f_i and f_j , and subtracting $\frac{\epsilon}{4}$ from each of the two leaf edges of f_i and f_j , respectively. Although this restores the balance for weights for faces f_i and f_j , it creates an imbalance for faces $f_{i-1}, f_{i+1}, f_{j-1}, f_{j+1}$ and cycle edges of G'_D . To fully balance the weights, we add $\frac{\epsilon}{4}$ to the cycle edges of these four faces. This assignment of weights meets Rivin's conditions. The process is repeated for each expanded backbone edge. See Figure 3.



Figure 3: Three possible cases when a backbone edge is expanded. Expanded face is shared by (a) four distinct faces, (b) two distinct faces and one common face and (c) two common faces. When a backbone edge is expanded, only weights of these neighbor faces have to be adjusted; weights of other faces remain unchanged. Here $w(l) = \frac{n-1}{2n}$

The weight assignment scheme leads to a maximum possible weight of 1/n for each edge. When n > 2, this is always smaller than 1/2. When we expand a backbone edge, we subtract a value from *some* of the edges of G'_D , and *always* add $\frac{\epsilon}{4}$ to two cycle edges of the adjacent cells. The maximum amount that we subtract is $\epsilon/2$ from a cycle edge of G'_D . Therefore, the only case when the weight on any edge becomes negative is when we subtract more than the initial value of the edge.



Figure 4: A stellated outerplanar graph and its dual (shown in red edges) where the backbone is a single vertex.

An extreme case occurs when, for each expansion of a backbone edge, $\epsilon/2$ is subtracted from the same edge. This is only possible when our original outerplanar graph G has all chords emanating from a single vertex. Consider a face in G'_D corresponding to such vertex in G. For each expansion of the backbone edge, the weight of the cycle edge of that face is decremented by $\epsilon/2$. Similarly, for each expansion, the weight of each of the two cycle edges is increased by $\frac{\epsilon}{4}$. It remains to prove that the weight of each edge of G'_D satisfies condition 2a of Theorem 5, if ϵ lies within a certain range.

Lemma 8 The weight of each edge e lies within the range $0 < w(e) \le 1/2$.

Proof. We prove this first for the extreme case where a face f of G'_D shares all the backbone edge, as in Fig 4. First we show the upper bound of w(e). Each time a backbone edge is expanded, the weight of each of the two cycle edges which are the edges of two adjacent faces of f is increased by $\frac{\epsilon}{4}$. Therefore, each such edge takes extra at most $\frac{\epsilon}{4}(k+1)$ from the weights, where k is the number of backbone edges in G'_D or chords in G'. In an outerplanar graph, k is exactly n-3. To maintain the upper bound of w(e), we need $\frac{1}{n} + \frac{\epsilon(n-2)}{4} \leq \frac{1}{2}$ or $\epsilon \leq \frac{2}{n}$.

Now we prove the lower bound of w(e). Each time a backbone edge is expanded, $\frac{\epsilon}{2}$ and $\frac{\epsilon}{4}$ are subtracted from the cycle and leaf edges of the face f respectively. Therefore, it suffices to show that the final weight of cycle edge remains positive. After adjusting weights for all backbone edges, the final weight of a boundary edge is $w(e) = \frac{1}{n} - \frac{(n-3)\epsilon}{2}$, as there are (n-3) chords in the outerplanar graph. Since the base weight 1/n is always less than 1/2, w(e) < 1/2. To make w(e) positive, we have to choose ϵ such that $\frac{(n-3)\epsilon}{2} < \frac{1}{n}$ or $\epsilon < \frac{2}{n(n-3)}$. This completes the proof, with weights assignments w(e) lying within the required range when $\epsilon < \min\{\frac{2}{n(n-3)}, \frac{2}{n}\}$.

3.2 Proof of non-coterminous cutset condition

A non-coterminous cutset is a cutset where all the edges of the cutset do not emanate from a single vertex. A non-coterminous cutset, like any other cutset, divides a connected graph into two components, where each component consists of at least two vertices. Since each vertex in the primal graph is represented by a face in the dual, the non-coterminous cutset in the primal is represented by a non-facial cycle in the dual. To prove the non-coterminous condition 2c of Theorem 5, we show now that for any non-facial cycle in the dual, the sum of weights of the edges of the cycle is strictly greater than 1.

For simplicity, let us consider a non-coterminous cutset where the non-facial cycle contains two adjacent faces in the dual, as in Fig 5. Let e be the edge shared by this two faces. According to condition 2b, the sum of the weights of the edges of each of these two faces is 1. Therefore, the sum of the weights of edges of the non-facial cycle is 2 - 2w(e). Since the weight of any edge is less than $\frac{1}{2}$, the weight of the cutset is strictly greater than 1. It is important to note that this cutset divides the primal graph into two components where one component has only two vertices joined by an edge. The two faces of the non-facial cycle represents these two vertices in the dual and edge e is the dual of the connecting edge in the primal.



Figure 5: The dotted lines show the non-facial cycle in the dual graph. Edge e is the extra edge in the cycle.

Now consider a non-facial cycle ${\cal C}$ which contains n

faces f_1, f_2, \dots, f_n . An edge e_x is called an *extra edge*, if it is shared by two faces f_i and f_j , where $1 \leq i \neq j \leq n$. Denote by k the extra edges in C and by w_{max} the largest weight possible on any of these k edges (which is essentially less than $\frac{1}{2}$). In order to satisfy the condition 2c, we need $n - 2kw_{max} > 1$. We need an upper bound of w_{max} (lower bound is trivially greater than 0). In order to find that, we need an upper bound of k too.

Recall that the extra edges in any non-facial cycle represent the edges of one of the two components. Since a non-coterminous cycle divides the primal graph into exactly two components, one of the components has at least as many vertices than the other one. Therefore, the number of vertices of the smaller component is $v \leq \frac{n}{2}$ and the number of edges in that component satisfy $e \leq 2v - 3 = n - 3$. Therefore the upper bound of k is n - 3. Substituting this value in the equation above, we get $w_{max} < \frac{n-1}{2(n-3)}$, where w_{max} is clearly less than $\frac{1}{2}$.

Finally, we need to convert w_{max} in terms of ϵ . Initially, in the contracted form, the cycle edges are assigned $\frac{1}{n}$ each and leaf edges are assigned $\frac{n-1}{2n}$ each. Both of these initial weights are within the bound of w_{max} . Whenever a backbone edge of face f_i is expanded, the weights of cycle and leaf edges of f_i are decreased. The only edges whose weights are increased are the cycle edges of face f_{i-1} and f_{i+1} . Hence it is possible for these edges to violate only the w_{max} condition. We get the bound of w_{max} based on these edges. The maximum weights of such cycle edges are encountered when the outerplanar graph is similar to the extreme case stated in lemma 8. In that case, only one face f_i shares all the backbone edges and the cycle edges of faces f_{i-1} and f_{i+1} incur maximum additional weights. Since there can be at most n-3 possible chords in the primal or backbone edges in the dual, the weight of each of these two cycle edge is $\frac{1}{n} + \frac{(n-3)\epsilon}{4}$. Therefore, the weights of these edges has to satisfy the condition that $\frac{1}{n} + \frac{(n-3)\epsilon}{4} \leq w_{max}$ or $\frac{1}{n} + \frac{(n-3)\epsilon}{4} < \frac{n-1}{2(n-3)}$. Solving

that $\frac{1}{n} + \frac{4}{4} = \alpha \max (1 + \frac{4}{n} + \frac{2(n-3)}{2(n-3)})$ this equation, we get $\epsilon < \frac{2(n^2 - 3n + 6)}{n(n-3)^2}$. To satisfy both conditions 2a and 2c, we will choose an ϵ such that $0 < \epsilon < \min\{\frac{2}{n(n-3)}, \frac{2}{n}, \frac{2(n^2 - 3n + 6)}{n(n-3)^2}\}$

This concludes our main proof.

Acknowledgement

This work was initiated at the 25th Bellairs Winter Workshop on Computational Geometry held January 1 – January 7, 2010.

References

 M. B. Dillencourt Realizability of Delaunay triangulations. Information Processing Letters, 33(6):283-287, 1990

- [2] T. Lambert An optimal algorithm for realizing a Delaunay triangulation *Information Processing Letters*, 62:245-250, 1997.
- [3] M. B. Dillencourt and W. D. Smith Graph-theoretical conditions for inscribability and Delaunay realizability. *Discrete Mathematics*, 161(1-3):63–77, December, 1996.
- [4] I. Rivin Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume *The Annals of Mathematics*, 139(3):533-580, May, 1994.
- [5] I. Rivin A characterization of ideal polyhedra in hyperbolic 3-space. *The Annals of Mathematics*, 143(1):51-70, January, 1996.
- [6] B. Grunbaum Convex Polytopes. Wiley Interscience, 1967.

Toward the Tight Bound of the Stretch Factor of Delaunay Triangulations^{*}

Ge Xia[†]

Liang Zhang[‡]

Abstract

In this paper, we investigate the tight bound of the stretch factor of the Delaunay triangulation by studying the stretch factor of the chain (Xia 2011). We define a sequence $\Gamma = (\Gamma_1, \Gamma_2, \Gamma_3, ...)$ where Γ_i is the maximum stretch factor of a chain of *i* circles, and show that Γ is strictly increasing. We then present an improved lower bound of 1.5932 for the stretch factors of the Delaunay triangulation. This bound is derived from a sequence of chains sharing a set of properties. We conjecture that these properties are also shared by a chain with the worst stretch factor.

1 Introduction

Let S be a finite set of points in the Euclidean plane. A *Delaunay triangulation* of S is a triangulation in which the circumscribed circle of every triangle contains no point of S in its interior. An alternative equivalent definition is: An edge xy is in the Delaunay triangulation of S if and only if there exists a circle through points x and y whose interior is devoid of points of S. A Delaunay triangulation of S is the dual graph of the Voronoi diagram of S.

Let D be a Delaunay triangulation of S. For two points p and q in S, denote by d(p,q) the length of the shortest path from p to q following the edges of the triangles in D and by ||pq|| the Euclidean line distance between p and q. Then the *stretch factor* (also known as *dilation* or *spanning ratio*) of a Delaunay triangulation of S is the maximum value of d(p,q)/||pq|| over all pairs of points p, q in S.

Proving the tight bound for the stretch factor of Delaunay triangulations has been a long standing open problem in computational geometry, with important applications in areas such as wireless communications. The stretch factor of Delaunay triangulations has an obvious lower bound of $\pi/2 \approx 1.571$ [3], which occurs when the points lie on a circle whose diameter is pq. Recently, Bose et. al. [2] gave an improved lower bound of $1.581 > \pi/2$ by constructing a configuration where the points are distributed on the boundary of two half circles separated by a small distance. They also showed a slightly better lower bound of 1.5846. In term of upper bounds, Dobkin, Friedman, and Supowit [5] in 1987 showed that the stretch factor of the Delaunay triangulation is at most $(1 + \sqrt{5})\pi/2 \approx 5.08$. This upper bound was improved by Keil and Gutwin [6] in 1989 to $2\pi/(3\cos(\pi/6)) \approx 2.42$. For the special case when the point set S is in convex position, Cui, Kanj and Xia [4] proved that the Delaunay triangulation of S has stretch factor at most 2.33. Recently, Xia [7] proposed to study the stretch factor of the Delaunay triangulation by focusing on the stretch factor of a chain of circles in the plane. With this approach, Xia [7] proved that the stretch factor of the Delaunay triangulation is less than 1.998.

Following the same approach as that in [7], we investigate the tight bound on the stretch factor of the Delaunay triangulation by studying the stretch factor of the chain. We define a sequence $\Gamma = (\Gamma_1, \Gamma_2, \Gamma_3, \ldots)$ where Γ_i is the maximum stretch factor of a chain of *i* circles. We prove that Γ is strictly increasing, which implies that the tight bound of the stretch factor of the chain is the limit of Γ . We then proceed to investigate what kind of chains achieve the stretch factors in Γ . To that end, we define a family of chains $\mathbb{C} = \{\mathcal{C}_3, \mathcal{C}_5, \mathcal{C}_7, \ldots\},\$ each having odd number of circles and satisfying certain structural properties. We conjecture that for all odd numbers $n \geq 3$, the stretch factor of the chain $\mathcal{C}_n \in \mathbb{C}$ is Γ_n . If this conjecture is true, then the problem of finding the tight bound is reduced to computing the limit of the stretch factor of the chains in \mathbb{C} .

Even without proving the conjecture, studying the stretch factor of the chains in \mathbb{C} is still interesting. It yields improved lower bound. To illustrate this, we compute the chains $\mathcal{C}_n \in \mathbb{C}$ for $n = 3, 5, 7, \ldots, 31$. This yields a lower bound of 1.5932 for the stretch factors of the Delaunay triangulation, improving the previous lower bound of 1.5846 by Bose et al. [2].

The paper is organized as follows. The necessary definitions are given in Section 2. In Section 3, we show that Γ is strictly increasing by proving that one can always increase the stretch factor of a chain by adding a circle to it. In Section 4, we present an improved lower bound of the stretch factors of the Delaunay triangulation. We conclude the paper in Section 5 with a conjecture and a question that are key to finding the tight bound of the

^{*}Partial results of this paper have been presented in the poster session of FWCG 2010.

[†]Department of Computer Science, Lafayette College, xiag@lafayette.edu. Supported by a Lafayette College research grant.

 $^{^{\}ddagger}Lafayette College, \texttt{zhangl@lafayette.edu}.$ Supported by the Lafayette College EXCEL Scholars program.

stretch factor of the Delaunay triangulation.

2 Preliminaries

We label the points in the plane by lower case letters, such as p, q, u, v, etc. For any two points p, q in the plane, denote by pq a line in the plane passing through p and q, by \overline{pq} the line segment connecting p and q, and by \overline{pq} the ray from p to q. The Euclidean distance between p and q is denoted by ||pq||. The length of a path P in the plane is denoted by |P|. Any angle denoted by $\angle poq$ is measured from \overrightarrow{op} to \overrightarrow{oq} in the *counterclockwise* direction. Unless otherwise specified, the angles are defined in the range $(-\pi, \pi]$.

Definition 1 We say that a sequence of distinct finite circles¹ $\mathcal{C} = (O_1, O_2, \ldots, O_n)$ in the plane is a $chain^2$ if it has the following three properties. **Prop**erty (1): Every two consecutive circles O_i, O_{i+1} intersect, $1 \leq i \leq n-1$. Let a_i and b_i be the shared points on their boundary (in the special case where O_i, O_{i+1} are tangent, $a_i = b_i$). Without loss of generality, assume a_i 's are on one side of C and b_i 's are on the other side. Denote by $C_i^{(i+1)}$ the arc on the boundary of O_i that is in O_{i+1} , and by $C_{i+1}^{(i)}$ the arc on the boundary of O_{i+1} that is in O_i . We refer to $C_i^{(i+1)}$ and $C_{i+1}^{(i)}$ as "connecting arcs". **Property (2)**: The connecting arcs $C_i^{(i-1)}$ and $C_i^{(i+1)}$ on the same circle O_i do not overlap, for $2 \le i \le n-1$; i.e., $C_i^{(i-1)}$ and $C_i^{(i+1)}$ do not share any point other than a boundary point. Property (3): There is a ray \overrightarrow{r} that crosses line segments $\overline{a_i b_i}$ for all $1 \leq i \leq n-1$ in that order. Let u be the entry-point of \overrightarrow{r} on O_1 and v the exit-point of \overrightarrow{r} on O_n . We call u, v a pair of terminal points (or simply terminals) of the chain \mathcal{C} . See Figure 1 for an illustration.

For notational convenience, define $a_0 = b_0 = u$ and $a_n = b_n = v$. Every circle O_i has two arcs on its boundary between the line segments $\overline{a_{i-1}b_{i-1}}$ and $\overline{a_ib_i}$, denoted by A_i and B_i . Without loss of generality, assume that a_{i-1}, a_i are the ends of A_i and b_{i-1}, b_i are the ends of B_i , for $1 \le i \le n$. This means that $P_A = A_1 \dots A_n$ is a path from u to v on one side of the chain and $P_B = B_1 \dots B_n$ is a path from u to v on the other side of the chain. An arc A_i or B_i may degenerate to a point, in which case $a_{i-1} = a_i$ or $b_{i-1} = b_i$, respectively.

We define the shortest path between u and vin C, denoted by $P_{\mathcal{C}}(u, v)$, to be the shortest path from u to v while traveling along arcs in $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_n\}$ and line segments in



Figure 1: A chain C. The connecting arcs are green (gray in black and white printing). The connecting arcs on the boundary of the same circle are disjoint. Points u and v are a pair of terminals of C.

 $\{\overline{a_1b_1}, \ldots, \overline{a_{n-1}b_{n-1}}\}$. Its length, $|P_{\mathcal{C}}(u, v)|$, is the total length of the edges in $P_{\mathcal{C}}(u, v)$. For example, in Figure 1, $P_{\mathcal{C}}(u, v)$ is the shortest path from u to v while traveling along the thick arcs and lines.

Now we can define the *stretch factor* of a chain \mathcal{C} to be the maximum value of

$$|P_{\mathcal{C}}(u,v)|/||uv||,\tag{1}$$

over all terminals u, v of C. The stretch factor of a chain is analogous to that of a Delaunay triangulation. From [7], the maximum stretch factor of the chain is an upper bound of the maximum stretch factor of the Delaunay triangulation. We believe that these two quantities are in fact equal (we discuss this in details in Section 4).

3 On the Tight Bound

Let $\Gamma = (\Gamma_1, \Gamma_2, \Gamma_3, \ldots)$ be a sequence where Γ_i is the maximum stretch factor of a chain of *i* circles. In this section we show that Γ is strictly increasing. This implies that the tight bound of the stretch factor of the chain is the limit of Γ : $\lim_{i\to\infty} \Gamma_i$.

Theorem 2 For all $n \ge 1$, $\Gamma_{n+1} > \Gamma_n$.

Proof. Let \mathcal{C} be a chain with stretch factor $\rho = \Gamma_n$. Without loss of generality, assume that \mathcal{C} has the *minimum* number of circles among all chains whose stretch factor is Γ_n . We will add a circle O_{n+1} to \mathcal{C} such that the new chain \mathcal{C}' has a stretch factor $> \rho$.

Refer to Figure 2. Let u, v be terminals of C with stretch factor ρ . Without loss of generality, assume that a_{n-1} is above uv and b_{n-1} is below uv. By flipping around uv, we can assume that o_n is on or below uv. We can also assume that v is not on the boundary of O_{n-1} because otherwise, we can remove O_n from C and still have the same stretch factor—a contradiction to

 $^{^1\}mathrm{In}$ this paper, a circle is considered to be a closed disk in the plane

²Note that our definition of a chain is slightly different from the chain defined in [7]. Our chain has an additional Property (3).



Figure 2: Illustration of adding a new circle O_{n+1} to C.

the fact that C is a chain of minimum number of circles with stretch factor ρ .

This means that there exist two points a_n and b_n sufficiently close to v on the boundary of O_n such that

$$|\widehat{va_n}| - |\widehat{vb_n}| = ||a_n b_n||, \tag{2}$$

where $\widehat{va_n}$ (resp. vb_n) is the arc on the boundary of O_n between v and a_n (resp. between v and b_n).

Add a new circle O_{n+1} going through a_n and b_n whose center is o_{n+1} . Denote the new chain by C'. Let v' be the point on O_{n+1} such that

$$|\widehat{v'a_n}| - |\widehat{v'b_n}| = ||a_nb_n||, \tag{3}$$

where $\widehat{v'a_n}$ (resp. $\widehat{v'b_n}$) is the arc on the boundary of O_{n+1} between v' and a_n (resp. between v' and b_n).

Refer to Figure 2. Let α be the angle from $\overrightarrow{o_n o_{n+1}}$ to $\overrightarrow{o_n a_n}$, β the angle from $\overrightarrow{o_n v}$ to $\overrightarrow{o_n o_{n+1}}$, and γ the angle from \overrightarrow{uv} to $\overrightarrow{o_n o_{n+1}}$.

Let $\Delta_O = ||o_n o_{n+1}||$, $\Delta_P = |P_{\mathcal{C}'}(u, v')| - |P_{\mathcal{C}}(u, v)|$, and $\Delta_D = ||uv'|| - ||uv||$. By a standard, if lengthy, calculation, we have the following from [7]:

$$\lim_{\Delta_O \to 0} \frac{\Delta_P}{\Delta_O} = \sin \alpha - \alpha \cos \alpha, \tag{4}$$

and

$$\lim_{\Delta_O \to 0} \frac{\Delta_D}{\Delta_O} = \cos \gamma - \cos \alpha (\cos(\beta - \gamma) + \beta \sin(\beta - \gamma)).$$
(5)

Let r_n be the radius of O_n . Note that $|\widehat{va_n}| = (\alpha + \beta)r_n$, $|\widehat{vb_n}| = (\alpha - \beta)r_n$, and $||a_nb_n|| = 2\sin\alpha r_n$. From (2), we have $(\alpha + \beta)r_n - (\alpha - \beta)r_n = 2\sin\alpha r_n$. This

means $\beta = \sin \alpha$. We have

$$\lim_{\Delta_O \to 0} \frac{\Delta_P}{\Delta_D} = \frac{\sin \alpha - \alpha \cos \alpha}{\cos \gamma - \cos \alpha (\cos(\beta - \gamma) + \beta \sin(\beta - \gamma))} = \frac{\sin \alpha - \alpha \cos \alpha}{\cos \gamma - \cos \alpha (\cos(\sin \alpha - \gamma) + \sin \alpha \sin(\sin \alpha - \gamma))}.$$
(6)

Set α small enough, say $\alpha = 0.01$. Then

$$\lim_{\Delta_O \to 0} \frac{\Delta_P}{\Delta_D} \left| \{ \alpha = 0.01, \gamma = 0 \} > 66.$$
 (7)

Refer to Figure 2. Let $\Delta_V = ||vv'||$. When $\gamma = 0$, we have

$$\Delta_D | \{ \gamma = 0 \} = -\cos(\angle v'vu')\Delta_V.$$
(8)

When $\gamma > 0$, we have

$$\Delta_D | \{\gamma > 0\} = -\cos(\angle v'vu)\Delta_V. \tag{9}$$

Let q and q' be the exit-point of $\overrightarrow{o_n o_{n+1}}$ on the boundary of O_n and O_{n+1} , respectively. Then $|\widehat{vq}| = ||a_n b_n||/2 =$ $|\widehat{v'q'}|$, where \widehat{vq} is the arc between v and q on the boundary of O_n and $\widehat{v'q'}$ is the arc between v' and q' on the boundary of O_{n+1} . Since α is small and $\Delta_O \to 0$, we have $r_n > r_{n+1}$. It is easy to see that $0 < \angle v'vu <$ $\angle v'vu'$. Also $\angle v'vu' < \pi$ because the distance from v' to line $o_n o_{n+1}$ is less than the distance from v to $o_n o_{n+1}$. This means $-\cos(\angle v'vu) < -\cos(\angle v'vu')$. From (8) and (9), we have

$$\Delta_D \mid \{\gamma > 0\} < \Delta_D \mid \{\gamma = 0\}.$$

Therefore

$$\lim_{\Delta_O \to 0} \frac{\Delta_P}{\Delta_D} \left| \{ \alpha = 0.01, \gamma > 0 \} \right|$$

$$> \lim_{\Delta_O \to 0} \frac{\Delta_P}{\Delta_D} \left| \{ \alpha = 0.01, \gamma = 0 \} \right|$$

$$> 66. \tag{10}$$

Since o_n is on or below uv, $\gamma > 0$. When $\Delta_O \to 0$ and α small enough, we have $\frac{\Delta_P}{\Delta_D} > 66$. From [7], the stretch factor of the chain is less than 2. So $\frac{|P_C(u,v)|}{||uv||} = \rho < 2$. Therefore, we have

$$\frac{|P_{\mathcal{C}'}(u,v')|}{||uv'||} = \frac{|P_{\mathcal{C}}(u,v)| + \Delta_P}{||uv|| + \Delta_D} > \frac{|P_{\mathcal{C}}(u,v)|}{||uv||} = \rho.$$

This completes the proof.

4 Improved Lower bounds

A natural question is what kind of chains achieve the worst stretch factor Γ_n . We present chains of 3, 5, 7, 15, 31 circles (see Figure 3) with stretch factors 1.5894,



(a) A chain a 3 circles with stretch factor $\rho_3 \approx 1.5894$.



(b) A Delaunay triangulation based on a chain of 3 circles with the same stretch factor as $\rho_3 \approx 1.5894$. The orange "shield" points are added to prevent short-cuts outside of the chain.



(c) A chain of 5 circles with stretch factor $\rho_5 \approx 1.5919$.



(e) A chain of 15 circles with stretch factor $\rho_{15} \approx 1.5931$.



(d) A chain of 7 circles with stretch factor $\rho_7 \approx 1.5926$.



(f) A chain of 31 circles with stretch factor $\rho_{31} \approx 1.5932$.

Figure 3: Illustration of the chains with improved lower bounds. The green line connects terminals u and v. The red dots are the centers of the circles in the chain. The blue lines in (b) show the Delaunay triangulation.

1.5919, 1.5926, 1.5931, and 1.5932, respectively. The exact configuration of the chain of 31 circles is given in the Appendix.

For each chain given in Figure 3, we can create a Delaunay triangulation with the same stretch factor as follows: place points of S densely on the outer boundary of C_n . With a small perturbation, one can ensure that the edges of the Delaunay triangulation inside the chain do not provide a short-cut for any shortest path between the terminals u and v in the chain, as shown in Figure 3(b). In order to prevent short-cuts outside of the chains, we use the technique of Bose et al. [2] by adding "shield" points, shown as the orange points in Figure 3(b). This yields a lower bound of 1.5932 on the stretch factor of the Delaunay triangulation, improving the previous best lower bound of 1.5846 by Bose et al. [1].

Theorem 3 There exists a set S of points in the plane, such that the Delaunay triangulation of S has a stretch factor of at least 1.5932.

5 Toward the Tight Bound

The chains C_n in Figure 3 all share the following properties: let n = 2k + 1 and let u and v be the terminals of C_n , then

- 1. for all $1 \le i \le k$, O_{k+1+i} and O_{k+1-i} are symmetric around a line l passing through o_{k+1} , the center of O_{k+1} ,
- 2. O_k , O_{k+1} and O_{k+2} share a point on l,
- 3. the radii of $O_{k+1}, O_{k+2}, \ldots, O_{2k+1}$ are in decreasing order and the radii of $O_1, O_2, \ldots, O_{k+1}$ are in increasing order,
- 4. for any $1 \le i \le n-1$, $\overline{a_i b_i}$ is contained in a shortest path from u to v, and
- 5. both $P_A = A_1 \dots A_n$ and $P_B = B_1 \dots B_n$ are shortest paths from u to v.

We conjecture that these properties are shared by a chain with the worst stretch factor:

Conjecture 4 For all $n = 2k + 1 \ge 3$ there is a chain of n circles with stretch Γ_n that satisfies Properties 1–5.

Note that we can assume Property 5 is true because of the following observation.

Proposition 5 ([7]) There is a chain C^* of n circles with stretch factor Γ_n , in which both P_A and P_B are shortest paths.



Figure 4: A chain of 4 circles with stretch factor 1.5907.

Proposition 5 was proved in [7] using a technique that transforms any chain of n circles into C^* without reducing the stretch factor. A similar technique of transformation may be helpful in proving other properties.

If Conjecture 4 is true, the task of finding the tight bound of the stretch factor of the Delaunay triangulation is reduced to answering the following question.

Question: What is the worst stretch factor of a chain satisfying Conditions 1–5.

Even without proving Conjecture 4, the answer to this question will give an improved lower bound of the stretch factor of the Delaunay triangulation.

Finally, note that a chain of even number of circles with the maximum stretch factor may not have the symmetry exhibited by the chains of odd number of circles. See Figure 4 for a chain of 4 circles with stretch factor 1.5907. This chain is not symmetric and all circles in it have different sizes.

References

- [1] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma. The spanning ratio of the Delaunay triangulation is greater than $\pi/2$. In *Proceedings of CCCG*, 2009.
- [2] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma. Almost all delaunay triangulations have stretch factor greater than $\pi/2$. Computational Geometry, 44(2):121 127, 2011.
- [3] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

- [4] S. Cui, I.A. Kanj, and G. Xia. On the stretch factor of delaunay triangulations of points in convex position. *Computational Geometry*, 44(2):104 – 109, 2011.
- [5] D. Dobkin, S. Friedman, and K. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete and Comp. Geom.*, 5(4):399–407, 1990.
- [6] J. Keil and C. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete and Comp. Geom.*, 7:13–28, 1992.
- [7] G. Xia. Improved upper bound on the stretch factor of delaunay triangulations. to appear in *Proceedings* of the 27th Annual Symposium on Computational Geometry (SoCG 2011).

Appendix

In the following, we give the exact configurations of chains of 31 circles, as shown in Figure 3 (f). Let n = 31. ρ_n is the stretch factor, b_1 is the angle from $\overrightarrow{o_1 u}$ to $\overrightarrow{o_2 o_1}$, and b_n is the angle from $\overrightarrow{o_{n-1} o_n}$ to $\overrightarrow{o_n v}$. For all $1 \leq i \leq n$, (x_i, y_i) are the x- and y-coordinates of o_i —the center of O_i , and r_i is the radius of O_i .

 $\rho_{31} = 1.59321532337905$

- $(x_1, y_1) = (-82.83285023949975, -25.121488078241036)$ $r_1 = 27.2227454174619$
- $(x_2, y_2) = (-72.85751097488247, -28.607891622305775)$ $r_2 = 37.5929692832941$
- $\begin{aligned} (x_3,y_3) &= (-65.06105288129035,-30.244664465966718) \\ r_3 &= 45.2705987926872 \end{aligned}$
- $(x_4, y_4) = (-58.46281391202502, -30.8967699166666095)$ $r_4 = 51.5306676497054$
- $(x_5, y_5) = (-52.71657386093427, -30.908699522242472) \\ r_5 = 56.8317425129397$
- $(x_6, y_6) = (-47.641009930136384, -30.47042857093014)$ $r_6 = 61.4105580627107$
- $(x_7, y_7) = (-43.12496351134913, -29.70369563073232)$ $r_7 = 65.4084222645816$
- $(x_8, y_8) = (-39.09176839012416, -28.69398435657018)$ $r_8 = 68.9185983263288$
- $(x_9, y_9) = (-35.4841587781849, -27.505033297273425)$ $r_9 = 72.0068096919405$
- $(x_{10}, y_{10}) = (-32.25667996954523, -26.186384074676056)$ $r_{10} = 74.7216529992949$

 $(x_{11}, y_{11}) = (-29.371513326910353, -24.777713407816105)$ $r_{11} = 77.1003848112902$

- $(x_{12}, y_{12}) = (-26.795869450721117, -23.311439841674048)$ $r_{12} = 79.1724821194007$
- $(x_{13}, y_{13}) = (-24.50025982738938, -21.814336462991804)$ $r_{13} = 80.9619450886396$
- $(x_{14}, y_{14}) = (-22.457239646974063, -20.308501263020666)$ $r_{14} = 82.488877282518$
- $(x_{15}, y_{15}) = (-20.639566299045978, -18.8111754808014)$ $r_{15} = 83.7712177530083$
- $(x_{16}, y_{16}) = (0.0, 0.0)$ $r_{16} = 100.0$ $(x_{17}, y_{17}) = (20.639566299045978, -18.8111754808014)$ $r_{17} = 83.7712177530083$ $(x_{18}, y_{18}) = (22.457239646974063, -20.308501263020666)$ $r_{18} = 82.488877282518$ $(x_{19}, y_{19}) = (24.50025982738938, -21.814336462991804)$ $r_{19} = 80.9619450886396$ $(x_{20}, y_{20}) = (26.795869450721117, -23.311439841674048)$ $r_{20} = 79.1724821194007$ $(x_{21}, y_{21}) = (29.371513326910353, -24.777713407816105)$ $r_{21} = 77.1003848112902$ $(x_{22}, y_{22}) = (32.25667996954523, -26.186384074676056)$ $r_{22} = 74.7216529992949$ $(x_{23}, y_{23}) = (35.4841587781849, -27.505033297273425)$ $r_{23} = 72.0068096919405$ $(x_{24}, y_{24}) = (39.09176839012416, -28.69398435657018)$ $r_{24} = 68.9185983263288$ $(x_{25}, y_{25}) = (43.12496351134913, -29.70369563073232)$ $r_{25} = 65.4084222645816$ $(x_{26}, y_{26}) = (47.641009930136384, -30.47042857093014)$ $r_{26} = 61.4105580627107$ $(x_{27}, y_{27}) = (52.71657386093427, -30.908699522242472)$ $r_{27} = 56.8317425129397$ $(x_{28}, y_{28}) = (58.46281391202502, -30.896769916666095)$ $r_{28} = 51.5306676497054$ $(x_{29}, y_{29}) = (65.06105288129035, -30.244664465966718)$ $r_{29} = 45.2705987926872$
- $\begin{aligned} (x_{30}, y_{30}) &= (72.85751097488247, -28.607891622305775) \\ r_{30} &= 37.5929692832941 \end{aligned}$
- $(x_{31}, y_{31}) = (82.83285023949975, -25.121488078241036)$ $r_{31} = 27.2227454174619$

 $b_1 = b_{31} = 0.22563636621218$
Rigid components in fixed-lattice and cone frameworks*

Matthew Berardi[†]

Brent Heeringa[‡]

Justin Malestein[†]

Louis Theran[†]

Abstract

We study the fundamental algorithmic rigidity problems for generic frameworks periodic with respect to a fixed lattice or a finite-order rotation in the plane. For fixedlattice frameworks we give an $O(n^2)$ algorithm for deciding generic rigidity and an $O(n^3)$ algorithm for computing rigid components. If the order of rotation is part of the input, we give an $O(n^4)$ algorithm for deciding rigidity; in the case where the rotation's order is 3, a more specialized algorithm solves all the fundamental algorithmic rigidity problems in $O(n^2)$ time.

1 Introduction

The geometric setting for this paper involves two variations on the well-studied *planar bar-joint* rigidity model: fixed-lattice periodic frameworks and cone frameworks. A fixed-lattice periodic framework is an infinite structure, periodic with respect to a lattice, where the allowed continuous motions preserve the lengths and connectivity of the bars as well as the periodicity with respect to a fixed lattice. See Figure 1(a). A cone framework is also made of fixed-length bars connected by universal joints, but it is finite and symmetric with respect to a finite order rotation; the allowed continuous motions preserve the bars' lengths and connectivity and symmetry with respect to a fixed rotation center. Cone frameworks get their name from the fact that the quotient of the plane by a finite order rotation is a flat cone with opening angle $2\pi/k$ and the quotient framework, embedded in the cone with geodesic "bars", captures all the geometric information [13]. Figure 2(a) shows an example.

A fixed-lattice framework is *rigid* if the only allowed motions are translations and *flexible* otherwise. A coneframework is *rigid* if the only allowed motions are isometries of the cone, which is just rotation around the cone point, and *flexible* otherwise. A framework is *minimally rigid* if it is rigid, but ceases to be so if any of the bars are removed. **Generic rigidity** The combinatorial model for the fixed-lattice and cone frameworks introduced above is given by a colored graph (G, γ) : G = (V, E) is a finite directed graph and $\gamma = (\gamma_{ij})_{ij \in E}$ is an assignment of a group element $\gamma_{ij} \in \Gamma$ (the "color") to each edge ij for a group Γ . For fixed-lattice frameworks, the group Γ is \mathbb{Z}^2 , representing translations; for cone frameworks it is $\mathbb{Z}/k\mathbb{Z}$ with $k \geq 2$ a natural number. See Figure 1(b) and Figure 2(b).

The colors can be seen as efficiently encoding a map ρ from the oriented cycle space of G into Γ ; ρ is defined, in detail, in Section 2. If the image of ρ restricted to a subgraph G' contains only the identity element, we define the Γ -image of ρ to be trivial otherwise it is non-trivial.



Figure 1: Periodic frameworks and colored graphs: (a) part of a periodic framework, with the representation of the integer lattice \mathbb{Z}^2 shown in gray and the bars shown in black; (b) one possibility for the the associated colored graph with \mathbb{Z}^2 colors on the edges. (Graphics from [12].)

In 2009 Elissa Ross announced the following theorem:

Theorem 1 ([12],[15]) A generic fixed-lattice periodic framework with associated colored graph (G, γ) is minimally rigid if and only if: (1) G has n vertices and 2n-2 edges; (2) all non-empty subgraphs G' of G with m' edges and n' vertices and trivial \mathbb{Z}^2 -image satisfy $m' \leq 2n' - 3$; (3) all non-empty subgraphs G' with nontrivial \mathbb{Z}^2 -image satisfy $m' \leq 2n' - 2$.

The colored graphs appearing in the statement of Theorem 1 are defined to be *Ross graphs*; if only conditions (2) and (3) are met, (G, γ) is *Ross-sparse*. Ross graphs generalize the well-known *Laman graphs* which

^{*}Extended abstract. Full proofs are in [2].

[†]Department of Mathematics, Temple University, {mberardi,justmale,theran}@temple.edu

[‡]Department of Computer Science, Williams College, heeringa@cs.williams.edu



Figure 2: Cone-Laman graphs: (a) a realization of the framework on a cone with opening angle $2\pi/3$ (graphic from Chris Thompson); (b) a $\mathbb{Z}/3\mathbb{Z}$ -colored graph (edges without colors have color 0); (c) the developed graph with $\mathbb{Z}/3\mathbb{Z}$ -symmetry (dashed edges are lifts of dashed edges in (b)).

are uncolored, have m = 2n - 3 edges, and satisfy (2). By Theorem 1 the maximal rigid sub-frameworks of a generic fixed-lattice framework on a Ross-sparse colored graph (G, γ) correspond to maximal subgraphs of Gwith m' = 2n' - 2; we define these to be the *rigid components* of (G, γ) .

Malestein and Theran [13] proved a similar statement for cone frameworks:

Theorem 2 ([13]) A generic cone framework with associated colored graph (G, γ) is minimally rigid if and only if: (1) G has n vertices and 2n - 1 edges; (2) all non-empty subgraphs G' of G with m' edges and n' vertices and trivial $\mathbb{Z}/k\mathbb{Z}$ -image satisfy $m' \leq 2n' - 3$; (3) all non-empty subgraphs G' with non-trivial $\mathbb{Z}/k\mathbb{Z}$ -image satisfy $m' \leq 2n' - 1$.

The graphs appearing in the statement of Theorem 2 are called *cone-Laman graphs*. We define *cone-Laman sparse* colored graphs and their rigid components similarly to the analogous definitions for Ross-sparse graphs, with 2n' - 1 replacing 2n' - 2.

Ross and cone-Laman graphs are examples of the " Γ graded-sparse" colored graphs introduced in [12, 13]. They are all matroidal families [12, 13], which guarantees that greedy algorithms work correctly on them.

Main results In this paper we investigate the algorithmic theory of crystallographic rigidity of fixed-lattice and cone frameworks. Given a colored graph (G, γ) ,

we are interested in the rigidity properties of an associated generic framework. Lee and Streinu [9] define three fundamental algorithmic rigidity questions: **Decision** Is the input rigid?; **Extraction** Find a maximum subgraph of the input corresponding to independent length constraints; **Components** Find the maximal rigid subframeworks of a flexible input.

We give algorithms for these problems with running times shown in the following table

	Decision	Extraction	Components
Fixed-lattice	$O(n^2)$	$O(n^3)$	$O(n^3)$
Cone $k \neq 3$	$O(n^4)$	$O(n^5)$	$O(n^5)$
Cone $k = 3$	$O(n^2)$	$O(n^2)$	$O(n^2)$

Novelty Previously, the only known efficient combinatorial algorithms for any of these problems were pointed out in [12, 13]: the Edmonds Matroid Union algorithm yields an algorithm with running times $O(n^4)$ for **Deci**sion and $O(n^5)$ **Extraction**. Recently, Ross presented a **Decision** algorithm for Ross graphs very similar to ours [15]. A folklore randomized algorithm based on Gaussian elimination gives an $O(n^3 \operatorname{polylog}(n))$ algorithm for **Decision** and **Extraction** of most rigidity problems, but this doesn't easily generalize to **Components**.

The $O(n^2)$ running time of **Decision** for fixed-lattice frameworks equals that from the *pebble game* [3, 8, 9] for the corresponding problem in finite frameworks. Although there are faster **Decision** algorithms [5] for finite frameworks, the pebble game is the standard tool in the field due to its elegance and ease of implementation. Our algorithms for cone frameworks with order 3 rotation are a reduction to the pebble games of [3, 8, 9].

The $O(n^3)$ running time for **Extraction** and **Components** in fixed-lattice frameworks is worse by a factor of O(n) than the pebble games for finite frameworks. However, it is equal to the $O(n^3)$ running time from [9] for the "redundant rigidity" problem. Computing *fundamental Laman circuits* (definition in Section 2) plays an important role (though for different reasons) in both of these algorithms.

Roadmap and key ideas Our main contribution is a pebble game algorithm for Ross graphs, from which we can deduce the corresponding results for general cone-Laman graphs. Intuitively, the algorithmic rigidity problems should be harder for Ross graphs than for Laman graphs, since the number of edges allowed in a subgraph depends on whether the \mathbb{Z}^2 -image of the subgraph is trivial or not. To derive an efficient algorithm we use three key ideas (detailed definitions are given in Section 2):

• The Lee-Streinu-Theran [11] approach of playing several copies of the pebble game for (k, ℓ) -graphs

[9] with different parameters to handle different sparsity counts for different types of subgraphs.

- A *new* structural characterization of the edge-wise minimal colored graphs which violate the Ross counts (Section 3).
- A *linear time* algorithm for computing the Γ image of a given subgraph (Section 4).

Our algorithms for general cone-Laman graphs then use the Ross graph **Decision** algorithm as a subroutine. When the order of the rotation is 3, we can reduce the cone-Laman rigidity questions to Laman graph rigidity questions directly, resulting in better running times.

Motivation Periodic frameworks, in which the lattice *can* flex, arise in the study of *zeolites*, a class of microporous crystals with a wide variety of industrial applications, notably in petroleum refining. Because zeolites exhibit *flexibility* [16], computing the degrees of freedom in *potential* [14, 18] zeolite structures is a well-motivated algorithmic problem.

Other related work The general subject of periodic and crystallographic rigidity has seen a lot of progress recently [4, 12, 13], see [7] for a list of announcements. Bernd Schulze [17] has studied Laman graphs with a free $\mathbb{Z}/3\mathbb{Z}$ action in a different context and Elissa Ross's recent thesis studies rigidity of infinite periodic frameworks.[15].

2 Preliminaries

In this section, we introduce the required background in colored graphs, hereditary sparsity, and introduce a data structure for least common ancestor queries in trees that is an essential tool for us.

Colored graphs and the map ρ A pair (G, γ) is defined to be a *colored graph* with Γ a group, G = (V, E) a finite, directed graph on *n* vertices and *m* edges, and $\gamma = (\gamma_{ij})_{ij \in E}$ is an assignment of a group element $\gamma \in \Gamma$ to each edge.

Let (G, γ) be a colored graph, and let C be a cycle in G with a fixed traversal order. We define $\rho(C)$ to be

$$\rho(C) = \sum_{\substack{ij \in C \\ ij \text{ traversed} \\ \text{forwards}}} \gamma_{ij} - \sum_{\substack{ij \in C \\ ij \text{ traversed} \\ \text{backwards}}} \gamma_{ij}$$

Since Γ is always abelian in this paper, we need not be concerned with the particular order of summation, and since we are interested in whether $\rho(C)$ is trivial or not, we are not concerned with sign. For a subgraph G' of G, we define $\rho(G')$ to be *trivial* if its image on cycles spanned by G' contains only the identity and *non-trivial* otherwise. We need the following fact about ρ .

Lemma 3 ([12, Lemma 2.2]) Let (G, γ) be a colored graph. Then $\rho(G)$ is trivial if and only if, for any spanning forest T of G, ρ is trivial on every fundamental cycle induced by T.

 (k, ℓ) -sparsity and pebble games The hereditary sparsity counts defining Ross and cone-Laman graphs generalize to (k, ℓ) -sparse graphs which satisfy $m' \leq kn' - \ell$ for all subgraphs; if in addition the total number of edges is $m = kn - \ell$, the graph is a (k, ℓ) -graph. We also need the notion of a (k, ℓ) -circuit, which is an edgeminimal graph that is not (k, ℓ) -sparse; these are always $(k, \ell - 1)$ -graphs [9]. If G is any graph, a (k, ℓ) -basis of G is a maximal subgraph that is (k, ℓ) -sparse; if G' is a (k, ℓ) -basis of G and $ij \in E(G) - E(G')$, the fundamental (k, ℓ) -circuit of ij with respect to G' is the unique (k, ℓ) -circuit in G' + ij. See [9] for a detailed development of this theory. As is standard in the field, we use "(2, 3)-" and "Laman" interchangeably.

Although (k, ℓ) -sparsity is defined by exponentially many inequalities, it can be checked in quadratic time using the *pebble game* [9], an incremental approach that builds a (k, ℓ) -sparse graph G one edge at a time. Here, we will use the pebble game as a "black box" to: (1) Check if an edge ij is in the span of any (k, ℓ) -component of G in O(1) time [9, 10]; (2) Assuming that G plus a new edge ij is (k, ℓ) -sparse, add the edge ij to G and update the components in amortized $O(n^2)$ time [9]; (3) Compute the fundamental circuit with respect to a given (k, ℓ) -sparse graph G in O(n) time [9].

Least common ancestors in trees Let T be a rooted tree with root r and let i and j be any vertices in T. The *least common ancestor* (shortly, LCA) of i and jis defined to be the vertex where the (unique, since Tis a tree) paths from i to r and j to r first converge. If either i or j is r, then this is just r. A fundamental result of Harel and Tarjan [6] is that LCA queries can be answered in O(1) time after O(n) preprocessing.

3 Combinatorial lemmas

In this section we prove structural properties of Ross and cone-Laman graphs that are required by our algorithms.

Ross graphs Let (G, γ) be a colored graph and suppose that G is a (2, 2)-graph. We can verify that (G, γ) is Ross by checking the \mathbb{Z}^2 -images of a relatively small set of subgraphs.

Lemma 4 ([2]) Let (G, γ) be a colored graph and suppose that G is a (2,2)-graph. Then (G, γ) is a Ross

graph if and only if for any Laman basis L of G, the fundamental Laman circuit with respect to L of every edge $ij \in E - E(L)$ has non-trivial \mathbb{Z}^2 -image.

Figure 3 shows two examples. The point is that we can pick any Laman basis L of G. The main idea is that G being a (2, 2)-graph forces all Laman circuits to be edge-disjoint, from which we can deduce all of them are fundamental Laman circuits of every Laman basis.



Figure 3: Examples of Ross and non-Ross graphs (edges without colors have color (0,0)): (a) a Ross graph; the underlying graph is itself a Laman circuit; (b) the underlying graph is a (2,2)-graph, but the uncolored K_4 subgraph has trivial image, so this is not a Ross graph. Note that K_4 is a Laman circuit, illustrating Lemma 4

Cone-Laman graphs Because cone-Laman graphs have an underlying (2, 1)-graph, the statement of Lemma 4, with (2, 1)- replacing (2, 2)- does *not* hold for cone-Laman graphs. The analogous statement, proven in the full version is:

Lemma 5 ([2]) Let (G, γ) be a colored graph. Then (G, γ) is a cone-Laman graph if and only if: (1) G is a (2,1)-graph; (2) for any (2,2)-basis R of G, the fundamental (2,2)-circuit G' with respect to R of $ij \in E(G) - E(R)$ becomes a Ross graph after removing any edge from G'; (3) for any Laman-basis L of G, the fundamental Laman-circuits with respect to L have nontrivial Γ -image.

Order three rotations In the special case where the group $\Gamma = \mathbb{Z}/3\mathbb{Z}$, which corresponds to a cone with opening angle $2\pi/3$, we can give a simpler characterization of cone-Laman graphs in terms of their *development*. The development \tilde{G} is defined by the following construction: \tilde{G} has three copies of each vertex $i: i_0, i_1$ and i_2 ; a directed edge ij with color γ then generates three undirected edges $i_k j_{k+\gamma}$ (addition is modulo 3). See Figure 2(c)) for an example. The development has a free $\mathbb{Z}/3\mathbb{Z}$ -action; a subgraph of \tilde{G} is defined to be symmetric if it is fixed by this action.

Lemma 6 ([2]) Let (G, γ) be a colored graph with $\Gamma = \mathbb{Z}/3\mathbb{Z}$. Then (G, γ) is a cone-Laman graph if and only

if its development G is a Laman graph. Moreover, the rigid components of (G, γ) correspond to the symmetric rigid components of \tilde{G} .

4 Computing the Γ -image of ρ

We now focus on the problem of deciding whether the Γ -image of the map ρ , defined in Section 2, is trivial on a colored graph (G, γ) . The case in which G is not connected follows easily by considering connected components one at a time, so we assume from now on that G is connected. Let (G, γ) be a colored graph and T be a spanning tree of G with root r. For a vertex *i*, there is a unique path P_i in T from r to *i*. We define σ_{ri} to be

$$\sigma_{ri} = \sum_{\substack{jk \in P_i \\ jk \text{ traversed forwards}}} \gamma_{jk} - \sum_{\substack{jk \in P_i \\ jk \text{ traversed backwards}}} \gamma_{jk}$$

The notation σ_{ri} extends in a natural way: for a a vertex j on P_i , we define σ_{ij} to be $\sigma_{ri} - \sigma_{rj}$; if σ_{ji} is defined, we define $\sigma_{ij} = -\sigma_{ji}$. The key observation is the following lemma:

Lemma 7 Let (G, γ) be a connected colored graph, let T be a rooted spanning tree of G, let ij be an edge of G not in T, and let a be the least common ancestor of i and j. Then, if C is the fundamental cycle of ij with respect to T, $\rho(C) = \sigma_{ai} + \gamma_{ij} - \sigma_{ja}$.

Proof. Traversing the fundamental cycle of ij so that ij is crossed from i to j means: going from i to j, from j to the LCA a of i and j towards the root, and then from a to i away from the root.

We now show how to compute whether the Γ -image of a colored graph is trivial in linear time. The idea used here is closely related to a folklore $O(n^2)$ algorithm for all-pairs-shortest paths in trees.¹

Lemma 8 Let (G, γ) be a connected colored graph with n vertices and m edges. There is an O(n+m) time algorithm to decide whether the Γ -image of $\rho(G)$ is trivial.

The rest of this section gives the proof of Lemma 8. We first present the algorithm.

Input: A colored graph (G, γ) **Question:** Is $\rho(G)$ trivial? **Method:**

- Pick a spanning tree T of G and root it.
- Compute σ_{ri} for each vertex *i* of *G*.
- For each edge *ij* not in *T*, compute the image of its fundamental cycle in *T*.
- Say 'yes' if any of these images are not the identity and 'no' otherwise.

 $^{^1\}mathrm{We}$ thank David Eppstein for clarifying the tree APSP trick's origins on MathOverflow.

Correctness This is an immediate consequence of Lemma 3, since the algorithm checks all the fundamental cycles with respect to a spanning tree.

Running time Finding the spanning tree with BFS is O(m) time, and once the tree is computed, the σ_{ri} can be computed with a single pass over it in O(n) time. Lemma 7 says that the image of any fundamental cycle with respect to T can be computed in O(1) time once the LCA of the endpoints of the non-tree edge is known. Using the Harel-Tarjan data structure, the total cost of LCA queries is O(n+m), and the running time follows.

The pebble game for Ross graphs We have all the pieces in place to describe our algorithm for the rigidity problems in Ross graphs.

Algorithm: Rigid components in Ross graphs

Input: A colored graph (G, γ) with *n* vertices and *m* edges.

Output: The rigid components of (G, γ) .

Method: We will play the pebble game for (2, 3)-sparse graphs and the pebble game for (2, 2)-sparse graphs in parallel. To start, we initialize each of these separately, including data structures for maintaining the (2, 2)- and (2, 3)-components.

Then, for each colored edge $ij \in E$:

- (A) If ij is in the span of a (2, 2)-component in the (2, 2)-sparse graph we are maintaining, we discard ij and proceed to the next edge.
- (B) If ij is not in the span of any (2,3)-component, we add ij to both the (2,2)-sparse and (2,3)-sparse graphs we are building, and update the components of each.
- (C) Otherwise, we use the (2, 3)-pebble game to identify the smallest (2, 3)-block G' spanning ij. We add ijto this subgraph G' and compute its \mathbb{Z}^2 -image. If this is trivial, we discard ij and proceed to the next edge.
- (D) If the image of G' was non-trivial, add ij to the (2, 2)-sparse graph we are maintaining and update its rigid components.

The output is the (2,2)-components in the (2,2)-sparse graph we built.

Correctness By definition, the rigid components of a Ross graph are its (2, 2)-components. Step (A) ensures that we maintain a (2, 2)-sparse graph; steps (B) and (C), by Lemma 4 imply that when new (2, 2)-blocks are formed *all* of them have non-trivial \mathbb{Z}^2 -image, which is what is required for Ross-sparsity. Step (D) ensures that the rigid components are updated at every step.

The matroidal property implies that a greedy algorithm is correct.

Running time By [9, 10], steps (**A**), (**B**), and (**D**) require $O(n^2)$ time over the entire run of the algorithm (the analysis of the time taken to update components is amortized). Step (**C**), by [9] and Lemma 7 requires O(n) time. Since $\Omega(m)$ iterations may enter step (**C**), this becomes the bottleneck, resulting in an O(nm) running time, which is $O(n^3)$.

Modifications for other rigidity problems We have presented and analyzed an algorithm for computing the rigid components in Ross graphs. Minor modifications give solutions to the **Decision** and **Extraction** problems. For **Extraction**, we just return the (2, 2)-sparse graph we built; the running time remains $O(n^3)$. For **Decision**, we simply stop and say 'no' if any edge is ever discarded. Since we process at most O(n) edges, the running time becomes $O(n^2)$.

5 Pebble games for cone-Laman graphs

We now describe our algorithms for cone-Laman graphs.

Order-three rotations We start with the special case when the group $\Gamma = \mathbb{Z}/3\mathbb{Z}$. In this case, the following algorithm's correctness is immediate from Lemma 6. The running time follows from [3, 9, 10] and the fact that the development can be computed in linear time.

Input: A colored graph (G, γ) with *n* vertices and *m* edges.

Output: The rigid components of (G, γ) . Method:

- (A) Compute the development \tilde{G} of (G, γ) .
- (B) Use the (2,3)-pebble game to compute the rigid components of \tilde{G} .
- (C) Return the subgraphs of G corresponding to the symmetric rigid components in \tilde{G} .

General cone-Laman graphs For colored graphs with $\Gamma = \mathbb{Z}/k\mathbb{Z}$, we don't have an analogue of Lemma 6, and the development may not be polynomial size. However, we can modify our pebble game for Ross graphs to compute the rigid components. Here is the algorithm: **Input:** A colored graph (G, γ) with *n* vertices and *m* edges, and an integer *k*.

Output: The rigid components of (G, γ) .

Method: We initialize a (2, 1)-pebble game, a (2, 2)-pebble game, and a (2, 3)-pebble game. Then, for each edge $ij \in E(G)$:

- (A) If ij is in the span of a (2, 1)-component in the (2, 1)-sparse graph we are maintaining, we discard ij and proceed to the next edge.
- (B) If ij is not in the span of any (2, 3)-component, we add ij to all three sparse graphs we are building, update the components of each, and proceed to the next edge.
- (C) If ij is not in the span of any (2, 2)-component, we check that its fundamental Laman circuit in the (2,3)-sparse graph has non-trivial $Z/k\mathbb{Z}$ -image. If not, discard ij. Otherwise, add ij to the (2, 1)- and (2, 2)-sparse graphs and update components.
- (D) Otherwise ij is not in the span of any (2, 1)component. We find the minimal (2, 2)-block G'spanning ij and check if G' + ij becomes a Ross
 graph after removing any edge. If so, add ij to the (2, 1)-graph we are building. Otherwise discard ij.

The output is the (2,1)-components in the (2,1)-sparse graph we built.

Analysis The proof of correctness follows from Lemma 5 and an argument similar to the one used to show that the pebble game for Ross graphs is correct. Each loop iteration takes $O(n^3)$ time, from which the claimed running times follow.

6 Conclusions and remarks

We studied the three main algorithmic rigidity questions for generic fixed-lattice periodic frameworks and cone frameworks. We gave algorithms based on the pebble game for each of them. Along the way we introduced several new ideas: a linear time algorithm for computing the Γ -image of a colored graph, a characterization of Ross graphs in terms of Laman circuits, and a characterization of cone-Laman graphs in terms of the development for k = 3 and Ross graphs for general k.

Implementation issues The pebble game has become the standard algorithm in the rigidity modeling community because of its elegance, ease of implementation, and reasonable implicit constants. The original data structure of Harel and Tarjan [6], unfortunately, is too complicated to be of much use except as a theoretical tool. More recent work of Bender and Farach-Colton [1] gives a vastly simpler data structure for O(1)-time LCA that is not much more complicated than the *union pair-find* data structure of [10] used in the pebble game. This means that the algorithm presented here is implementable as well.

References

- M. A. Bender and M. Farach-Colton. The LCA problem revisited. In Proc. LATIN'00, pages 88–94, 2000.
- [2] M. Berardi, B. Heeringa, J. Malestein, and L. Theran. Rigid components in fixed-lattice and cone frameworks. Preprint, arXiv:1105.3234, 2011.
- [3] A. R. Berg and T. Jordán. Algorithms for graph rigidity and scene analysis. In *ESA 2003*, volume 2832 of *LNCS*, pages 78–89. 2003.
- [4] C. Borcea and I. Streinu. Periodic frameworks and flexibility. Proc. of the Royal Soc. A, 466:2633–2649, 2010.
- [5] H. N. Gabow and H. H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7(5-6):465–497, 1992.
- [6] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. SIAM J. Comput., 13:338– 355, May 1984.
- [7] B. Jackson, J. Owen, and S. Power. London mathematical society workshop: Rigidity of frameworks and applications. http://www.maths.lancs.ac.uk/~power/ LancRigidFrameworks.htm, 2010.
- [8] D. J. Jacobs and B. Hendrickson. An algorithm for twodimensional rigidity percolation: the pebble game. J. Comput. Phys., 137(2):346–365, 1997.
- [9] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Math.*, 308(8):1425–1437, 2008.
- [10] A. Lee, I. Streinu, and L. Theran. Finding and maintaining rigid components. In *Proc. CCCG'05*, 2005.
- [11] A. Lee, I. Streinu, and L. Theran. Graded sparse graphs and matroids. *Journal of Universal Computer Science*, 13(11):1671–1679, 2007.
- [12] J. Malestein and L. Theran. Generic combinatorial rigidity of periodic frameworks. Preprint, arXiv:1008.1837, 2010.
- [13] J. Malestein and L. Theran. Generic combinatorial rigidity of crystallographic frameworks. Preprint, 2011.
- [14] I. Rivin. Geometric simulations a lesson from virtual zeolites. *Nature Materials*, 5(12):931–932, Dec 2006.
- [15] E. Ross. The Rigidity of Periodic Frameworks as Graphs on a Torus. PhD thesis, York University, May 2011.
- [16] A. Sartbaeva, S. Wells, M. Treacy, and M. Thorpe. The flexibility window in zeolites. *Nature Materials*, Jan 2006.
- [17] B. Schulze. Symmetric versions of Laman's theorem. Discrete Comput. Geom., 44(4):946–972, 2010.
- [18] M. Treacy, I. Rivin, E. Balkovsky, and K. Randall. Enumeration of periodic tetrahedral frameworks. II. Polynodal graphs. *Microporous and Mesoporous Materials*, 74:121–132, 2004.

Orientations of Simplices Determined by Orderings on the Coordinates of their Vertices^{*}

Emeric Gioan^{1,2}

Kevin Sol^2

Gérard Subsol 1,2

Abstract

We address the problem of testing when orderings on coordinates of n points in an (n-1)-dimensional affine space, one ordering for each coordinate, suffice to determine if these points are the vertices of a simplex (i.e. are affinely independent), and the orientation of this simplex, independently of the real values of the coordinates. In other words, we want to know when the sign (or the non-nullity) of the determinant of a matrix whose columns correspond to affine points is determined by orderings given on the values on each row. We completely solve the problem in dimensions 2 and 3, providing a direct combinatorial characterization, together with a formal calculus method, that can be seen also as a decision algorithm, which relies on testing the existence of a suitable inductive cofactor expansion of the determinant. We conjecture that the method we use generalizes in higher dimensions. The motivation for this work is to be part of a study on how oriented matroids encode shapes of 3-dimensional objects, with applications in particular to the analysis of anatomical data for physical anthropology and clinical research.

Keywords: simplex orientation, determinant sign, chirotope, coordinate ordering, combinatorial algorithm, formal calculus, oriented matroid, 3D model.

1 Introduction

We consider n points in an (n-1)-dimensional real affine space. For each of the n-1 coordinates, an ordering is given, applied on the n values of the points with respect to this coordinate. We address the problem of testing if these points are the vertices of a simplex (i.e. are affinely independent, i.e. do not belong to a same hyperplane), and of determining the orientation of this simplex, assuming only that their coordinates satisfy the given orderings, independently of their real values.

More formally, we consider the following generic matrix (where each e_i is the label of a point and each b_i is the index of a coordinate)

$$M_{\mathcal{E},\mathcal{B}} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_{e_1,b_1} & x_{e_2,b_1} & \dots & x_{e_n,b_1} \\ x_{e_1,b_2} & x_{e_2,b_2} & \dots & x_{e_n,b_2} \\ \vdots & \vdots & & \vdots \\ x_{e_1,b_{n-1}} & x_{e_2,b_{n-1}} & \dots & x_{e_n,b_{n-1}} \end{pmatrix}$$

together with orderings given on the values on each row, and we want to know when the sign (or the non-nullity) of its determinant is determined by these orderings only.

Equivalently, we consider the above formal matrix and the affine algebraic variety of $\mathbb{R}^{n \times (n-1)}$ whose equation is $det(M_{\mathcal{E},\mathcal{B}}) = 0$. Then we look for which regions of $\mathbb{R}^{n \times (n-1)}$, delimited by the hyperplanes $x_{e_i,b_k} = x_{e_j,b_k}$, for all $1 \leq i, j \leq n$ and all $1 \leq k \leq n-1$, have a non-empty intersection with this variety (obviously, regions delimited by these hyperplanes are in canonical bijection with coordinate linear orderings).

In this paper, we completely solve the problem in dimensions 2 (Section 4) and 3 (Section 5), providing a direct combinatorial characterization, together with a combinatorial formal calculus method, that can be seen also as a decision algorithm, to test if the orientation is determined or not. More precisely, our method relies on testing the existence of a suitable inductive cofactor expansion of the determinant, from which a combinatorial formal calculus is able to determine the sign of the determinant. We conjecture that such a characterization generalizes in higher dimensions (Section 3).

The motivation for this work is to be part of a study on how oriented matroids [1] encode shapes of 3dimensional objects, with applications in particular to the analysis of anatomical data for physical anthropology and clinical research [3]. In these applications, we usually study a set of models belonging to a given group (e.g. a set of 3D landmark points located on human or primate skulls) and we look for the significant properties encoded by the combinatorial structure. The above results allow us to distinguish chirotopes (i.e. simplex orientations) which are determined by the "generic" form (e.g. in any skull, the mouth is below the eyes) from those which are specific to anatomical variations. As an example, some results on anatomical 3D data are presented in Section 6.

^{*}Research supported by the OMSMO Project LIRMM France and the TEOMATRO Grant ANR-10-BLAN 0207.

¹CNRS

²LIRMM, Univ. Montpellier 2, France. {lastname}@lirmm.fr

2 Formalism and terminology of the problem

We warn the reader that we use on purpose a rather abstract formalism throughtout the paper (formal variables instead of real values, indices within arbitrary ordered sets instead of integers). This will allow us to get easier and non-ambiguous constructions and definitions.

Let us fix an (ordered) set $\mathcal{E} = \{e_1, \ldots, e_n\}$, with size n, of *labels*, and an (ordered) canonical basis $\mathcal{B} = \{b_1, \ldots, b_{n-1}\}$, with size n-1, of the (n-1)-dimensional real space \mathbb{R}^{n-1} . We denote $M_{\mathcal{E},\mathcal{B}}$ - or M for short when the context is clear - the formal matrix whose entry at column i and row j+1, for $1 \leq i \leq n$ and $1 \leq j \leq n-1$, is the formal variable x_{e_i,b_j} , as represented in Section 1. The determinant $det(M_{\mathcal{E},\mathcal{B}})$ of this formal matrix is a multivariate polynomial on these formal variables, and the main object studied in this paper.

Let \mathcal{P} be a set of n points, labeled by \mathcal{E} , in \mathbb{R}^{n-1} considered as an affine space. We denote $M_{\mathcal{E},\mathcal{B}}(\mathcal{P})$ - or $M(\mathcal{P})$ for short - the matrix whose columns give the coordinates of points in \mathcal{P} w.r.t. the basis \mathcal{B} , that is specifying real values for the formal variables x_{e_i,b_j} in the matrix $M_{\mathcal{E},\mathcal{B}}$ above. For $e \in \mathcal{E}$ and $b \in \mathcal{B}$, we denote $x_{e,b}(\mathcal{P})$ the real value given to the formal variable $x_{e,b}$ in \mathcal{P} . We may sometimes denote $x_{e,b}$ for short instead of $x_{e,b}(\mathcal{P})$ when the context is clear. We call orientation of \mathcal{P} , or chirotope of \mathcal{P} in the oriented matroid terminology, the sign of $det(M(\mathcal{P}))$, belonging to the set $\{+, -, 0\}$. It is the sign of the real evaluation of the polynomial det(M) at the real values given by \mathcal{P} . This sign is not equal to zero if and only if \mathcal{P} forms a simplex (basis of the affine space).

We call ordering configuration on $(\mathcal{E}, \mathcal{B})$ - or configuration for short - a set \mathcal{C} of n-1 orderings $\langle b_1, \ldots, \langle b_{n-1} \rangle$ on \mathcal{E} , one ordering for each element of \mathcal{B} . In general, such an ordering can be any partial ordering. If every ordering $\langle b, b \in \mathcal{B}$, is linear, then \mathcal{C} is called a *linear ordering configuration*. An element of \mathcal{E} which is the smallest or the greatest in a linear ordering on \mathcal{E} is called *extreme* in this ordering. We call *reversion* of an ordering the ordering obtained by reversing every inequality in this ordering.

Given a configuration \mathcal{C} on $(\mathcal{E}, \mathcal{B})$ and a set of n points \mathcal{P} labeled by \mathcal{E} , we say that \mathcal{P} satisfies \mathcal{C} if, for all $b \in \mathcal{B}$, the natural order (in the set of real numbers \mathbb{R}) of the coordinates b of the points in \mathcal{P} is compatible with the ordering $<_b$ of \mathcal{C} , that is precisely :

$$\forall b \in \mathcal{B}, \ \forall e, f \in \mathcal{E}, \ e <_b f \Rightarrow x_{e,b}(\mathcal{P}) < x_{f,b}(\mathcal{P}).$$

One may observe that the set of all \mathcal{P} satisfying \mathcal{C} forms a convex polyhedron, more precisely: a (full dimensional) region of the space $\mathbb{R}^{n \times (n-1)}$, delimited by some hyperplanes of equations of type $x_{e,b} = x_{f,b}$ for $b \in \mathcal{B}$ and $e, f \in \mathcal{E}$.

We say that a configuration C is *fixed* if all the sets of points \mathcal{P} satisfying C form a simplex and have the same orientation. In this case, the sign of $det(M(\mathcal{P}))$ is the same for all \mathcal{P} satisfying \mathcal{C} . Then we call sign of det(M) this sign, belonging to $\{[+], [-]\}$ accordingly, and we denote it $\sigma_{\mathcal{C}}(det(M))$. If \mathcal{C} is non-fixed, then its sign is $\sigma_{\mathcal{C}}(det(M)) = [\pm]$.

The following lemma is easy to prove.

Lemma 1 The following propositions are equivalent:

(a) The configuration C is non-fixed, that is $\sigma_{\mathcal{C}}(det(M)) = [\pm].$

(b) There exist two sets of points \mathcal{P}_1 and \mathcal{P}_2 satisfying \mathcal{C} and forming simplices that do not have the same orientation, that is $det(M(\mathcal{P}_1)) > 0$ and $det(M(\mathcal{P}_2)) < 0$;

(c) There exists a set of points \mathcal{P} satisfying \mathcal{C} and such that the points of \mathcal{P} belong to one hyperplane, that is $det(M(\mathcal{P})) = 0$.

Two configurations on $(\mathcal{E}, \mathcal{B})$ are called *equivalent* if they are equal up to a permutation of \mathcal{B} , a permutation of \mathcal{E} (relabelling), and some reversions of orderings (symmetries from the geometrical viewpoint). Note that changing a configuration into an equivalent one comes, in a matricial setting, to change the orderings of rows, of columns, and to multiply some rows by -1. Obviously those operations do not change the non-nullity of the determinant, hence two equivalent configurations are fixed or non-fixed simultaneously.

Now, given an ordering configuration C, the aim of the paper is to determine if C is fixed or non-fixed.

3 Computable fixity criteria and conjectures

3.1 From partial orderings to linear orderings

We recall that a linear extension of an ordering on a set \mathcal{E} is a linear ordering on \mathcal{E} compatible with this ordering. A *linear extension* of an ordering configuration \mathcal{C} on $(\mathcal{E}, \mathcal{B})$ is a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$ obtained by replacing each ordering on \mathcal{E} in C by one of its linear extensions.

Lemma 2 Let C be a configuration on $(\mathcal{E}, \mathcal{B})$. If there exists a set \mathcal{P} of n points satisfying C and contained in an hyperplane, then there exists a set of n points \mathcal{P}' contained in an hyperplane and a linear extension C' of C satisfied by \mathcal{P}' .

From the previous (easy) lemma, we get (directly) the above proposition.

Proposition 1 Let C be a configuration on $(\mathcal{E}, \mathcal{B})$. The configuration C is non-fixed if and only if there exists a non-fixed linear extension of C. The configuration C is fixed if and only if every linear extension of C is fixed.

The above result allows to test only the fixity of linear ordering configurations to deduce the fixity of any configuration. In what follows, we will concentrate on linear ordering configurations.

3.2 Formal fixity

Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$. We consider formal expression of type $x_{e,b}-x_{f,b}$ for $e, f \in \mathcal{E}$, $e \neq f$, and $b \in \mathcal{B}$, which we may sometimes denote $x_{e-f,b}$ for short. Such a formal expression gets a *formal sign w.r.t.* C denoted $\sigma_{\mathcal{C}}(x_{e,b}-x_{f,b})$ and belonging to $\{[+], [-]\}$, the following way:

$$\begin{aligned} \sigma_{\mathcal{C}}(x_{e,b} - x_{f,b}) &= & + & \text{if} \quad f <_b e; \\ \sigma_{\mathcal{C}}(x_{e,b} - x_{f,b}) &= & - & \text{if} \quad e <_b f. \end{aligned}$$

Recall that the polynomial $det(M_{\mathcal{E},\mathcal{B}})$ is a multivariate polynomial on variables $x_{e,b}$ for $b \in \mathcal{B}$ and $e \in \mathcal{E}$. Assume a particular formal expression of $det(M_{\mathcal{E},\mathcal{B}})$ is a sum of multivariate monomials where each variable is replaced by some $x_{e,b} - x_{f,b}$, for $b \in \mathcal{B}$ and $e, f \in \mathcal{E}$. Various expressions of this type can be obtained by suitable transformations and determinant cofactor expansions from the matrix M, as we will do more precisely below. This particular expression of $det(M_{\mathcal{E},\mathcal{B}})$ gets a formal sign w.r.t. \mathcal{C} belonging to $\{[+], [-], [?]\}$, by replacing each expression of type $x_{e,b} - x_{f,b}$ with its formal sign $\sigma_{\mathcal{C}}(x_{e,b} - x_{f,b})$ and applying the following formal calculus rules:

$+ \cdot +$	=	_ · _	=	+ ,
$+ \cdot -$	=	- · +	=	,
+ + +	=	+	=	+,
- + -	=	+	=	_ ,
+ + -	=	-++	=	?,

and the result of any operation involving a ? term or factor is also ?.

We say that C is formally fixed if $det(M_{\mathcal{E},\mathcal{B}})$ has such a formal expression whose formal sign is not ?.

Example. Consider the following matrix $M = M_{\mathcal{E},\mathcal{B}}$ for $\mathcal{E} = \{a, b, c\}$ and $\mathcal{B} = \{1, 2\}$:

$$M = \begin{pmatrix} 1 & 1 & 1 \\ x_{a,1} & x_{b,1} & x_{c,1} \\ x_{a,2} & x_{b,2} & x_{c,2} \end{pmatrix}$$

and consider the configuration \mathcal{C} defined by:

A formal expression of det(M) is:

This

$$det(M) = x_{b-a,1} \cdot x_{c-a,2} - x_{b-a,2} \cdot x_{c-a,1}$$

whose formal sign w.r.t. C is

 $\underbrace{+} \cdot \underbrace{-} - \underbrace{-} \cdot \underbrace{+}_{e} = \underbrace{?}_{e}$ Another formal expression of det(M) is:

 $det(M) = x_{b-a,1} \cdot x_{c-b,2} - x_{b-a,2} \cdot x_{c-b,1}$ whose formal sign w.r.t. \mathcal{C} is

$$\begin{array}{c} + \cdot + - - \cdot + = + \\ \text{second expression shows that } \mathcal{C} \text{ is formally fixed} \end{array}$$

Observation 1 If C is formally fixed, then C is fixed.

More precisely, given an expression as above whose formal sign w.r.t. C is [+] or [-], the evaluation of this determinant for any set of real values \mathcal{P} satisfying C necessarily provides a real number whose sign is consistent with the formal sign of this expression. In this case, this resulting sign does not depend on the chosen expression as soon as it is not [?], and $\sigma_C(det(M))$ equals this sign.

Conversely, one may wonder if for every fixed configuration there would exist a suitable expression of the determinant showing formally that C is fixed by this way. That is, equivalently: if every formal expression of $det(M_{\mathcal{E},\mathcal{B}})$ has formal sign ?, then $\sigma_C(det(M)) = \pm$. We strongly believe in this result, which we state as a conjecture, and which we will prove for $n \leq 4$.

Conjecture 1 Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$. Then C is fixed if and only if C is formally fixed.

3.3 Formal fixity by expansion

Let C be a configuration on $(\mathcal{E}, \mathcal{B})$, and $\mathcal{E}' = \mathcal{E} \setminus \{e\}$, $\mathcal{B}' = \mathcal{B} \setminus \{b\}$ for some $e \in \mathcal{E}, b \in \mathcal{B}$. We call configuration induced by C on $(\mathcal{E}', \mathcal{B}')$ the configuration on $(\mathcal{E}', \mathcal{B}')$ obtained by restricting every ordering $\langle_{b'}, b' \in \mathcal{B}'$, of C to \mathcal{E}' . Moreover, we say that all the configurations induced by C on \mathcal{E}' are fixed if, for every $b \in \mathcal{B}$, the configuration induced by C on $(\mathcal{E}', \mathcal{B} \setminus \{b\})$ is a fixed configuration.

Let $M = M_{\mathcal{E},\mathcal{B}}$ as previously, with $\mathcal{E} = \{e_1, ..., e_n\}_{<}$ and $\mathcal{B} = \{b_1, ..., b_{n-1}\}_{<}$. Let $e_i, e_j \in \mathcal{E}$, with $e_i \neq e_j$. Consider the matrix obtained from M by substracting the *j*-th column (corresponding to e_j), from the *i*-th column (corresponding to e_i), that is:

$$\begin{pmatrix} 1 & \dots & 1 & 0 \\ x_{e_1,b_1} & \dots & x_{e_{i-1},b_1} & x_{e_i,b_1} - x_{e_j,b_1} \\ x_{e_1,b_2} & \dots & x_{e_{i-1},b_2} & x_{e_i,b_2} - x_{e_j,b_2} \\ \vdots & \vdots & \vdots \\ x_{e_1,b_{n-1}} & \dots & x_{e_{i-1},b_{n-1}} & x_{e_i,b_{n-1}} - x_{e_j,b_{n-1}} \\ & & 1 & \dots & 1 \\ & & & x_{e_{i+1},b_1} & \dots & x_{e_n,b_1} \\ & & & & x_{e_{i+1},b_2} & \dots & x_{e_n,b_2} \\ & & & \vdots & & \vdots \\ & & & & x_{e_{i+1},b_{n-1}} & \dots & x_{e_n,b_{n-1}} \end{pmatrix}$$

The determinant of this matrix equals det(M). The cofactor expansion formula for the determinant of this matrix with respect to its *i*-th column yields:

$$det(M_{\mathcal{E},\mathcal{B}}) = \sum_{k=1}^{n-1} (-1)^{i+k+1} \cdot (x_{e_i,b_k} - x_{e_j,b_k}) \\ \cdot det(M_{\mathcal{E}\setminus\{e_i\},\mathcal{B}\setminus\{b_k\}})$$

which we call expression of det(M) by expansion with respect to (e_i, e_j) .

Then the above particular expression of det(M) gets a *formal sign* w.r.t. C the following way. First, replace each expression of type $x_{e,b} - x_{f,b}$ with its formal sign w.r.t. C in $\{ +, - \}$, and replace each $det(M_{\mathcal{E} \setminus \{e_i\}, \mathcal{B} \setminus \{b_k\}}), 1 \leq k \leq n-1$, with its sign $\sigma_{\mathcal{C}_k}(det(M_{\mathcal{E} \setminus \{e_i\}, \mathcal{B} \setminus \{b_k\}})) \in \{ +, -, \pm \}$, where \mathcal{C}_k is the configuration induced by C on $(\mathcal{E} \setminus \{e_i\}, \mathcal{B} \setminus \{b_k\})$. This leads to the formal expression:

$$\sum_{k=1}^{n-1} (-1)^{i+k+1} \cdot \sigma_{\mathcal{C}}(x_{e_i,b_k} - x_{e_j,b_k}) \\ \cdot \sigma_{C_k} \left(det \left(M_{\mathcal{E} \setminus \{e_i\}, \mathcal{B} \setminus \{b_k\}} \right) \right).$$

Then, provide the formal sign of this expression by using the same formal calculus rules as previously, completed with the following one:

$$+\cdot\pm$$
 = $-\cdot\pm$ = ?.

If there exists such an expression of det(M) by expansion whose formal sign is + or -, then C is called formally fixed by expansion.

Observation 2 If C is formally fixed by expansion, then C is fixed.

The above observation is similar to Observation 1: if \mathcal{C} is formally fixed by expansion then $\sigma_{\mathcal{C}}(det(M))$ is given as the formal sign of any expression certifying that \mathcal{C} is formally fixed by expansion. Notice that if \mathcal{C} is formally fixed by expansion then all those configurations \mathcal{C}_k induced by \mathcal{C} are fixed, since one must have $\sigma_{\mathcal{C}_k}(det(M_{\mathcal{E}\setminus\{e_i\},\mathcal{B}\setminus\{b_k\}})) \in \{[+], [-]\}.$

Conjecture 2 Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$. Then C is fixed if and only if C is formally fixed by expansion.

We point out that if Conjecture 1 is true in dimension n-1, then Conjecture 2 in dimension n implies Conjecture 1 in dimension n. Indeed, in this case, the fixity of the (n-1)-dimensional configurations corresponding to cofactors can be determined using formal expressions.

Finally, the point of this paper is to deal with the property of being formally fixed by expansion as an inductive criterion for fixity. In what follows, we will prove Conjecture 2 for n = 4, together with more precise and direct characterizations in this case.

3.4 A non-fixity criterion

The following Lemma 3 will be our main tool to prove that a configuration is non-fixed. We point out that, when n = 4, the sufficient condition for being non-fixed provided by Lemma 3 turns out to be a necessary and sufficient condition (see Theorem 4). However, the authors feel that this equivalence result is too hazardous to be stated as a general conjecture in dimension n.

Lemma 3 Let C be a configuration on $(\mathcal{E}, \mathcal{B})$. If the configuration C' induced by C on $(\mathcal{E} \setminus \{e\}, \mathcal{B} \setminus \{b\})$ for some $e \in \mathcal{E}$ and $b \in \mathcal{B}$ satisfies the following properties: C' is non-fixed and e is extreme in the ordering $<_b$ of C, then C is non-fixed.

4 Results in dimension 2

In this section we fix n = 3 and $\mathcal{E} = \{A, B, C\}$. In order to lighten notations of variables $x_{e,b}$ for $e \in \mathcal{E}$ and $b \in \mathcal{B}$, we rather denote:

$$M = \left(\begin{array}{rrrr} 1 & 1 & 1 \\ x_A & x_B & x_C \\ y_A & y_B & y_C \end{array}\right)$$

We will denote also $\mathcal{B} = \{x, y\}$ and $\langle x, \langle y \rangle$ the orderings in a configuration.

The following theorems are easy to prove. First it is easy to check that, up to equivalence of configurations, there exist exactly two linear ordering configurations:

$$\begin{array}{ll} A <_x B <_x C & A <_x B <_x C \\ A <_y B <_y C & B <_y C <_y A \end{array}$$

They correspond respectively to the following grid representations:



Theorem 1 Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$ with n = 3, $\mathcal{E} = \{A, B, C\}$ and $\mathcal{B} = \{x, y\}$. The following properties are equivalent:

a) C is non-fixed;

b) the two orderings on \mathcal{E} in \mathcal{C} are either equal or equal to reversions of each other;

c) up to equivalence,
$$C$$
 is equal to $\begin{array}{c} A <_x B <_x C \\ A <_y B <_y C \end{array}$

Theorem 2 Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$ with n = 3, $\mathcal{E} = \{A, B, C\}$ and $\mathcal{B} = \{x, y\}$. The following properties are equivalent:

a) C is fixed;

b) C is formally fixed;

c) up to equivalence, C is equal to $\begin{array}{c} A <_x B <_x C \\ B <_y C <_y A \end{array}$

Now that we have listed fixed and non-fixed linear ordering configurations, we are able to determine all fixed and non-fixed configurations using Proposition 1 (up to equivalence, and omitting those obviously non-fixed for which two elements of \mathcal{E} are comparable in no ordering in the configuration):

$$\begin{array}{c|cccc} A <_x C & & \\ B <_x C & & \\ B <_y A & & \\ C <_y A & & \\ \end{array} \quad \text{non-fixed}$$

5 Results in dimension 3

In this section we fix n = 4 and $\mathcal{E} = \{A, B, C, D\}$. In order to lighten notations of variables $x_{e,b}$ for $e \in \mathcal{E}$ and $b \in \mathcal{B}$, we rather denote:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ z_A & z_B & z_C & z_D \end{pmatrix}$$

We will denote also $\mathcal{B} = \{x, y, z\}$ and $\langle x, \langle y, \rangle \langle z \rangle$ the orderings in a configuration.

As noticed in Section 3, in order to prove that a configuration \mathcal{C} is formally fixed by expansion, we need to find an element $e \in E$ such that all the configurations induced by \mathcal{C} on $\mathcal{E} \setminus \{e\}$ are fixed. The proposition below characterizes such induced configurations.

Proposition 2 Let C be a configuration on $(\mathcal{E}, \mathcal{B})$ with $n = 4, \mathcal{E} = \{A, B, C, D\}$ and $\mathcal{B} = \{x, y, z\}$. All the configurations induced by C on $\{A, B, C\}$ are fixed if and only if C is equivalent to a configuration whose orderings

	B	$<_x$	C	$<_x$	A
satisfy:	C	$<_y$	A	$<_y$	B
	A	$<_z$	B	$<_z$	C

We will now state Theorem 3 which is the main result of the paper. Its detailed proof is about five pages long. Briefly, it consists in separating configurations having a triplet such that all induced configurations w.r.t. this triplet are fixed, and the other ones. In the first group, characterized by Proposition 2, we prove a sufficient condition for fixity. Then we prove that every configuration in the first group not satisfying this condition, and every configuration in the second group, is non-fixed, by analysing several cases, and always using Lemma 3 together with Theorem 1. So, it turns out that Lemma 3 completely characterizes non-fixed configurations, which proves also Theorem 4.

Theorem 3 Let C be a configuration on $(\mathcal{E}, \mathcal{B})$ with n = 4, $\mathcal{E} = \{A, B, C, D\}$ and $\mathcal{B} = \{x, y, z\}$. The following propositions are equivalent:

a) C is fixed;

b) C is formally fixed;

- c) C is formally fixed by expansion;
- d) up to equivalence, C satsifies:

and there exists $X \in \{A, B, C\}$ such that either $X <_b D$ for every $b \in \mathcal{B}$, or $D <_b X$ for every $b \in \mathcal{B}$.

Theorem 4 Let C be a linear ordering configuration on $(\mathcal{E}, \mathcal{B})$ with n = 4. Then C is non-fixed if and only if conditions of Lemma 3 are satisfied, that is: there exist $e \in \mathcal{E}$ and $b \in \mathcal{B}$ such that the configuration C' induced by C on $(\mathcal{E} \setminus \{e\}, \mathcal{B} \setminus \{b\})$ is non-fixed and e is extreme in the ordering $<_b$ of C.

We computed the result provided by Theorem 3 to list the fixed linear ordering configurations when n = 4. Up to equivalences, there are exactly 4 such configurations, within 21 linear ordering configurations:

$B <_x C <_x A <_x D$ $C <_y A <_y B <_y D$ $A <_z B <_z C <_z D$	$B <_x C <_x D <_x A$ $C <_y A <_y B <_y D$ $A <_z B <_z C <_z D$
$B <_x D <_x C <_x A$ $C <_y A <_y B <_y D$ $A <_z B <_z C <_z D$	$B <_x C <_x D <_x A$ $C <_y D <_y A <_y B$ $A <_z B <_z C <_z D$

The interest of the results of this section is to provide a combinatorial characterization as well as an (easily computable) algorithm deciding if a configuration is fixed or not. Also, we point out that our result statements deal with being fixed or not, but not with the exact value + or - of the considered fixed configuration. This sign can be derived easily from the construction stating the fixity. As well, this sign can be obtained by choosing any set of points \mathcal{P} satisfying the configuration and evaluating the sign of the real number $det(M(\mathcal{P}))$. Finally, from the list of fixed linear ordering configurations given above, one may compute the list of all fixed (partial) ordering configurations using Proposition 1, but we do not give this list here.

6 An example from applications to anatomical data

Let us consider ten anatomical landmark points in \mathbb{R}^3 chosen by experts on the 3D model of a skull from [2], as shown on Figure 1. We choose a canonical basis $(O, \vec{x}, \vec{y}, \vec{z})$ such that the axis \vec{x} goes from the right of the skull to its left, the axis \vec{y} goes from the bottom of the skull to its top, and the axis \vec{z} goes from the front of the skull to its back. The specificity of this 3D model as being a skull implies that some coordinate ordering relations are satisfied by those points: for instance the point 9 (right internal ear) will always be on the right, above and behind w.r.t. point 5 (right part of the chin). Figures 2 and 3 show respectively those points from the front and from the right of the model, together with a grid representing those coordinate ordering relations. By this way, the ordering configurations are represented on Figures 2 and 3, with \mathcal{E} any set of four points, and \mathcal{B} corresponding to the three axis $\{x, y, z\}$.



Figure 1: Ten anatomic points on a skull model [2]





Figure 3: View from the right

We are usually given a set of such models, coming from various individuals (with possibly some pathologies) and species (e.g. primates and humans), by experts of those fields who are interested in characterizing and classifying mathematically those models. In this paper, our aim is to detect which configurations are fixed, independently of the real values of the landmarks, meaning that the relative positions of points satisfying these configurations do not depend on some anatomical variabilities (e.g. being a primate or a human skull), but just on the generic shape of the model (i.e. being a skull).

Example 1. Fixed linear ordering configurations providing a fixed partial ordering configuration: the configuration on $\mathcal{E} = \{2, 5, 8, 9\}$ is fixed.

This configuration is given by the orderings:

$$\begin{array}{c} 9 <_x 5 <_x 2 <_x 8 \\ 5 <_y 8 <_y 9 <_y 2 \\ 2 <_z 8 <_z 9 \quad \text{and} \quad 5 <_z 8 <_z 9 \end{array}$$

Its two linear extensions, respectively C_1 and C_2 , are the following:

$9 <_x 5 <_x 2 <_x 8$	$9 <_x 5 <_x 2 <_x 8$
$5 <_y 8 <_y 9 <_y 2$	$5 <_y 8 <_y 9 <_y 2$
$2 <_z 5 <_z 8 <_z 9$	$5 <_z 2 <_z 8 <_z 9$
Let us write these orderin	ngs in another way:
$2 <_z 5 <_z 8 <_z 9$	$5 <_z 2 <_z 8 <_z 9$
$5 <_y 8 <_y 9 <_y 2$	$5 <_y 8 <_y 9 <_y 2$
$9 <_x 5 <_x 2 <_x 8$	$9 <_x 5 <_x 2 <_x 8$
Dry this more me and that	up to a pormutatio

By this way, we see that, up to a permutation of \mathcal{B} , that is for $\{i, j, k\} = \{x, y, z\}$, and if we choose A = 9, B = 2, C = 8 and D = 5, then the orderings in those configurations both satisfy:

$$B <_i C <_i A$$
$$C <_j A <_j B$$
$$A <_k B <_k C$$

as required by Theorem 3. Moreover, for each of these orderings we have D smaller than C (i.e. $5 <_x 8, 5 <_y 8, 5 <_z 8$). So, by Theorem 3, those two configurations are fixed. Then, C is fixed by Proposition 1.

Example 2. A non-fixed ordering configuration implied by a non-fixed linear ordering configuration: the configuration on $\mathcal{E} = \{1, 3, 7, 10\}$ is non-fixed.

It is given by the orderings:

One of its linear extensions is \mathcal{C}' :

$$\begin{array}{c} 7 <_x 3 <_x 1 <_x 10 \\ 7 <_y 10 <_y 3 <_y 1 \\ 3 <_z 1 <_z 7 <_z 10 \end{array}$$

whose configuration induced on ({7,3,1}, {x,y}) is
$$\begin{array}{c} 7 <_x 3 <_x 1 \\ 7 <_y 3 <_y 1 \end{array}$$

which is non-fixed by Theorem 1. Then 10 is extreme in the ordering $<_z$ of configuration \mathcal{C}' , hence \mathcal{C}' is non-fixed by Lemma 3, and so is \mathcal{C} by Proposition 1.

References

- A. Björner, M. Las Vergnas, B. Sturmfels, N. White, G. Ziegler, Oriented matroids 2nd ed., Encyclopedia of Mathematics and its Applications 46, Cambridge University Press, Cambridge, UK 1999.
- [2] J. Braga, J. Treil. Estimation of pediatric skeletal age using geometric morphometrics and three-dimensional cranial size changes. Int. J. Legal. Med. (2007) 121:439– 443.
- [3] E. Gioan, K. Sol, G. Subsol, Y. Heuzé, J. Richstmeier, J. Braga, F. Thackeray. A new 3D morphometric method based on a combinatorial encoding of 3D point configurations: application to skull anatomy for clinical research and physical anthropology. Presented at 80th Annual Meeting of the American Association of Physical Anthropologists, Minneapolis (U.S.A.), April 2011. Abstract published in American Journal of Physical Anthropology, p. 280, Vol. 144 Issue S52, 2011.

Pushing the boundaries of polytopal realizability

David Bremner^{*}

Antoine Deza[†]

William Hua[‡]

Lars Schewe[§]

Abstract

Let $\Delta(d, n)$ be the maximum possible diameter of the vertex-edge graph over all d-dimensional polytopes defined by n inequalities. The *Hirsch bound* holds for particular n and d if $\Delta(d, n) \leq n - d$. Francisco Santos recently resolved a question open for more than five decades by showing that $\Delta(d, 2d) = d + 1$ for d = 43; the dimension was then lowered to 20 by Matchske, Santos and Weibel. This progress has stimulated interest in related questions. The existence of a polynomial upper bound for $\Delta(d, n)$ is still an open question, the best bound being the quasi-polynomial one due to Kalai and Kleitman in 1992. Another natural question is for how large n and d the Hirsch bound holds. Goodev showed in 1972 that $\Delta(4, 10) = 5$ and $\Delta(5,11) = 6$, and more recently, Bremner and Schewe showed $\Delta(4,11) = \Delta(6,12) = 6$. Here we show that $\Delta(4, 12) = \Delta(5, 12) = 7$ and present strong evidence that $\Delta(6, 13) = 7$.

1 Introduction

Finding a good bound on the maximal diameter $\Delta(d, n)$ of the 1-skeleton (vertex-edge graph) of a polytope in terms of its dimension d and the number of its facets nis one of the basic open questions in polytope theory [9]. Although some bounds are known, the behaviour of the function $\Delta(d, n)$ is largely unknown. The Hirsch conjecture, formulated in 1957 and reported in [4], states that $\Delta(d, n)$ is linear in n and d: $\Delta(d, n) \leq n-d$. The conjecture was recently disproved by Santos [18] by exhibiting a counterexample for $\Delta(d, 2d)$ with d = 43 which was further improved to d = 20 [17]. The conjecture is known to hold in small dimensions, i.e. for $d \leq 3$ [14], along with other specific pairs of d and n (Table 1). However, the asymptotic behaviour of $\Delta(d, n)$ is not well understood: the best upper bound — due to Kalai and Kleitman — is quasi-polynomial [11].

The behaviour of $\Delta(d, n)$ is not only a natural question of extremal discrete geometry, but is historically closely connected with the theory of the simplex method. The approach of using abstract models [6, 7, 12] to study linear optimization has recently achieved the exciting result of a subexponential lower bound for Zadeh's rule [7], another long standing open problem. On the positive side, several authors have recently shown upper bounds for interesting special cases of the simplex method [21] and the diameter problem [16].

In this article we will show that $\Delta(4, 12) = \Delta(5, 12) =$ 7 and present strong evidence for $\Delta(6, 13) =$ 7. The first of these new values continues the pattern of $\Delta(4, n) =$ n-5 for $n \ge 10$. It would be very interesting to establish a general sub-Hirsch bound for d = 4. The considered computational approaches might help to narrow the gap between the smallest entries for d and n - d yielding a counterexample and the largest ones for which the Hirsch conjecture still holds.

Our approach is computational and builds on the approach used by Bremner and Schewe [3]. As in [3] we reduce the determination of $\Delta(d, n)$ to a set of simplicial complex realizability problems. Section 2 introduces our computational framework and some related background. A common theme in the SAT literature is that the hardest instances to solve are those that are "almost satisfiable"; we find a similar classification of our realizability problems. Compared to [3], this work involves significantly more computation, and we discuss a simple but effective parallelization strategy in Section 2. Finally we discuss our new bounds in Section 3. Again comparing with [3], the results here have the feature that they do not rely on having a priori upper bounds on the value of $\Delta(d, n)$ to be computed, but rather on inductive computation of $\Delta(d, n)$ using bounds on $\Delta(d-1, n-1).$

2 General approach

In this section we give a summary of our general approach. For more on the theoretical background, the reader is referred to [3].

It is easy to see via a perturbation argument that $\Delta(d, n)$ is always achieved by some simple polytope.

^{*}Faculty of Computer Science, University of New Brunswick, Box 4400, Fredericton NB, Canada. bremner@unb.ca

[†]Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada and Equipe Combinatoire et Optimisation, Université Pierre et Marie Curie, Paris, France. deza@mcmaster.ca

[‡]Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada. huaw@mcmaster.ca

[§]Department of Mathematics, Friedrich-Alexander-University, Erlangen-Nuremberg, Germany. lars.schewe@math.uni-erlangen.de

				n-2c	l	
		0	1	2	3	4
	4	4	5	5	6	7+
	5	5	6	7-8	7+	8 +
d	6	6	7-9	8 +	9+	9+
	$\overline{7}$	7-10	8 +	9+	10 +	11 +
	8	8+	9+	10 +	11 +	12 +

Table 1: Previously known bounds on $\Delta(d, n)$ [3, 8, 10, 15].

By a reduction applied from [15], we only need to consider *end-disjoint* paths: paths where the end vertices do not lie on a common facet (*facet-disjointness*). It will be convenient both from an expository and a computational view to work in a polar setting where we consider the lengths of facet-paths on the boundary of simplicial polytopes. We apply the term *end-disjoint* equally to the corresponding facet paths, where it has the simple interpretation that two end facets do not intersect.

For any set $Z = \{x_1 \dots x_{r-2}, y_1 \dots y_4\} \subset \mathbb{R}^r$, as a special case of the Grassmann-Plücker relations [1, §3.5] on determinants we have

$$det(X, y_1, y_2) \cdot det(X, y_3, y_4) + det(X, y_1, y_4) \cdot det(X, y_2, y_3)$$
(1)
$$- det(X, y_1, y_3) \cdot det(X, y_2, y_4) = 0$$

where $X = \{x_1 \dots x_{d-1}\}$. We are in particular interested in the case where r = d+1 and Z represents (d+3)points in \mathbb{R}^d in homogeneous coordinates; the various determinants are then signed volumes of simplices. In the case of points drawn from the vertices of a simplicial polytope, we may assume without loss of generality that these simplices are never flat, i.e. determinant 0. Thus if we define $\chi(v_1 \dots v_{d+1}) = \operatorname{sign}(\operatorname{det}(v_1 \dots v_{d+1}))$ it follows from (1) that

$$\{ \chi(X, y_1, y_2) \chi(X, y_3, y_4), \\ -\chi(X, y_1, y_3) \chi(X, y_2, y_4), \\ \chi(X, y_1, y_4) \chi(X, y_2, y_3) \} = \{-1, +1\}.$$

Any alternating map $\chi : E^{d+1} \to \{-,+\}$ satisfying these constraints for all (d+3)-subsets is called a *uni*form chirotope; this is one of the many axiomatizations of *uniform oriented matroids* [1]. In the rest of this paper we call uniform chirotopes simply chirotopes. A facet is a d-set $F \subset E$ such that for all $g \in E \setminus F$, $\chi(F,g)$ has the same sign. An *interior point* of a chirotope is some $g \in E$ that is not contained in any facet. We are mainly concerned with *convex* chirotopes, i.e. those without interior points.

A combinatorial facet-path is a simplicial complex with a path as dual graph, where edges are defined by two d-simplices sharing a (d-1)-simplex. Our general



Figure 1: Illustrating a non-shortest facet-path.

strategy is to show $\Delta(d, n) \neq k$ by generating all nonisomorphic combinatorial facet-paths of length k on nvertices in dimension d and showing that none can be embedded on the boundary of a chirotope as a shortest path. This is established by showing for each candidate combinatorial facet-path π that there is no alternating sign map $\chi(\cdot)$ that

- P1 Satisfies the Grassman-Plücker constraints, i.e. is a chirotope,
- P2 Forces each *d*-simplex of the candidate facet-path to be a facet of the chirotope,
- P3 Does not induce a *shortcut*, i.e. a facet-path of length shorter than k between the end facets of π . See Figure 2 for an illustration of a shortcut on a 3-dimensional polytope.

There are $\binom{n}{d+3}$ Grassman-Plücker constraints in their natural encoding, and this further expands by a factor of 16 when converted to conjunctive normal form (CNF) suitable for a SAT solver.

Facet constraints actually remove variables from the problem, since they define sets of equations. Equations can in principle be removed as a preprocessing step, although most modern SAT solvers deal with equality constraints quite effectively, even when the constraints are transformed to conjunctive normal form.

Each potential shortcut can be eliminated with 2 constraints encoding the fact that some d-simplex of the potential shortcut is not a facet. In principle one can generate all conceivable shortcuts by considering all short paths in the graph of all possible pivots between dsimplices, but this approach is generally impractical. We therefore use an incremental approach where candidate chirotopes are generated and any shortcuts on the boundary of these candidate solutions are used to generate new constraints.

A notable omission from the list of constraints above is that we do not explicitly constrain the alternating map $\chi(\cdot)$ to be convex. We note that either every element is in some facet, and thus the chirotope is convex by definition, or there is some interior point not used by the long facet-path. A realization with interior points corresponds to a realization on a smaller number of elements. In the work here we are always have bounds for $\Delta(d, j)$ for j < n when working on a bound for $\Delta(d, n)$, so we effectively reduce non-convex cases to smaller convex ones.

Chirotopes can be viewed as a generalization of real polytopes in the sense that for every real polytope, we can obtain its chirotope directly. Therefore, showing the non-existence of chirotopes satisfying properties P1–P3 immediately precludes the existence of real polytopes satisfying the same properties.

The search for a chirotope with properties P1 and P2 is encoded as an instance of SAT [19, 20, 3], with P3 handled implicitly via adding constraints and resolving. Each SAT problem is solved with MiniSat [5]. MiniSat itself discovers many constraints during the solution process, and these are carried forward between successive subproblems.

The generation of all possible paths for particular d and n begins with case where the paths are *non-revisiting*, i.e. paths where no vertex is visited more than once. These can be generated via a simple recursive scheme, using a bijection with *restricted growth strings*, i.e. k-ary strings where the symbols first occur in order. Each symbol represents a choice of pivot, and the strings can be unpacked into combinatorial facet-paths.

Multiple revisit facet-paths are generated from facetpaths with one less revisit by identifying pairs of vertices. Such an identification is valid only if it results in another facet-path, i.e. does not introduce new ridges, and if the resulting facet-path is still end-disjoint.

If a vertex is not used in a facet-path we call this occurrence a *drop*. See Figure 2 for an illustration of a path of length 6 involving 1 revisit (vertex 2) and and 1 drop (vertex 8) with n = 9 and d = 3. We can then classify paths by dimension d, primal-facets/dual-vertices n, length k, the number of revisits m, and the number of drops l. For end-disjoint paths, a simple counting argument yields:

$$egin{array}{rcl} m-l&=&k+d-n\ m&\leq&k-d\ l&\leq&n-2d \end{array}$$

1

Table 2 provides the number of paths to consider for each possible combination of d, n, k, m, and l.

With the implementation of [3], we were able to reconfirm Goodey's results for $\Delta(4, 10)$ and $\Delta(5, 11)$ in a



Figure 2: Example of a facet-path.

d	n	k	$\mid m$	l	#
4	10	6	0	0	15
4	10	6	1	1	24
4	10	6	2	2	16
4	11	7	0	0	50
4	11	7	1	1	200
4	11	7	2	2	354
4	11	7	3	3	96
4	12	8	0	0	160
4	12	8	1	1	1258
4	12	8	2	2	5172
4	12	8	3	3	7398
4	12	8	4	4	1512
5	11	7	1	0	98
5	11	7	2	1	98
5	12	8	1	0	1079
5	12	8	2	1	3184
5	12	8	3	2	2904
6	12	7	1	0	11
6	13	8	1	0	293
6	13	8	2	1	452

Table 2: Number of paths to consider, SAT instances to solve.

matter of minutes. While the number of paths to consider increases with the number of the revisits, in our experiments these paths are much less computationally demanding than the ones with fewer revisits. For example, the 7,398 paths of length 8 on 4-polytopes with 12 facets and involving 3 revisits and 3 drops require only a tiny fraction of the computational effort to tackle the 160 paths without a drop or revisit.

In order to deal with the intractability of the problem as the dimension, number of facets, and path length increased, we proceeded by splitting our original facet embedding problem into subproblems by fixing chirotope



Figure 3: Using partial backtracking to generate subproblems

signs. We use the non-SAT based mpc backtracking software [2] to backtrack to a certain fixed level of the search tree; every leaf job was then processed in parallel on the Shared Hierarchical Academic Research Computing Network (SHARCNET). Figure 3 (a partial trace of the execution of mpc) illustrates the splitting process on a problem generated from the octahedron. Note that variable propagation (similar to the *unit propagation* used by SAT solvers) reduces the number of leaves of the tree.

Jobs requiring a long time to complete were further split and executed on the cluster until the entire search space was covered. Table 3 provides the number of paths which were computationally difficult enough to require splitting. For example, out of 160 paths of length 8 on 4-polytopes with 12 facets without drop or revisit, 2 required splitting.

d	n	k	$\mid m$	l	#
4	12	8	0	0	2
5	12	8	1	0	15
5	12	8	2	1	6
6	13	8	1	0	138
6	13	8	2	1	63

Table 3: Number of difficult paths.

3 Results

Summarizing the computational results, we have:

Proposition 1 There are no (4, 12)- or (5, 12)- polytopes with facet-disjoint vertices at distance 8.

Note that we actually prove something slightly stronger: for (d, n) = (4, 12) or (5, 12), no (d, n)-chirotope has has vertex-disjoint facets at distance 8, where distance is defined by the shortest facet-path.

While the non-existence of k-length paths implies the non-existence of (k+1)-length paths, it is not obvious if the non-existence of end-disjoint k-length paths implies the non-existence of (k+1)-length paths. To be able to rule out vertices — not necessarily facet-disjoint — at distance l > k, we introduce the following lemma.

Lemma 1 If $\Delta(d-1, n-1) < k$ and there is no (d, n)-polytope with two facet-disjoint vertices at distance k, then $\Delta(d, n) < k$.

Proof. Assume the contrary. Let u and v be vertices on a (d, n)-polytope at distance $l \ge k$. By considering a shortest path from u to v, there is a vertex w at distance k from u. u and w must share a common facet F to prevent a contradiction. F is a (d - 1, n - 1)-polytope with diameter at least k.

By Proposition 1 and because $\Delta(3,11) = 6$ and $\Delta(4,11) = 6$ (see [14, 3]) we can apply Lemma 1 to obtain the following new entry for $\Delta(d, n)$.

Corollary 1 $\Delta(4, 12) = \Delta(5, 12) = 7$

We recall the following result of Klee and Walkup [15]:

Property 1 $\Delta(d, 2d+k) \leq \Delta(d-1, 2d+k-1) + \lfloor k/2 \rfloor + 1$ for $0 \leq k \leq 3$

Applying Property 1 to $\Delta(5, 12) = 7$ yields a new upper bound $\Delta(6, 13) \leq 8$, from which we could obtain $\Delta(6, 13) = 7$ if the still underway computations for remaining 8-paths keep on showing unsatisfiability for (d, n) = (6, 13).

Property 1 along with the 2 new entries for $\Delta(d, n)$ and, assuming $\Delta(6, 13) = 7$, would imply the additional upper bounds: $\Delta(5, 13) \leq 9$, $\Delta(6, 14) \leq 11$, $\Delta(7, 14) \leq$ 8, $\Delta(7, 15) \leq 12$ and $\Delta(8, 16) \leq 13$; see Table 4.

			1	n-2d		
		0	1	2	3	4
	4	4	5	5	6	7
	5	5	6	7	7-9	8 +
d	6	6	7	8-11	9+	9+
	7	7-8	8-12	9+	10 +	11 +
	8	8-13	9+	10 +	11 +	12 +

Table 4: Summary of bounds on $\Delta(d, n)$ assuming $\Delta(6, 13) = 7$.

4 Conclusions

In this paper we have presented new bounds for the diameter of the 1-skeleton of convex polytopes in dimensions 4 and 5. It remains open to find the smallest n and d for which the Hirsch bound fails to hold: we are also interested if the current trend which shows $\Delta(4,n) = n-5$ continues beyond n = 12. The tools used here are mainly computational as in [3], although further analysis of the relationship between bounds on end-disjoint paths and bounds on more general paths was needed in order to establish new bounds without requiring a priori upper bounds. Furthermore, the scale of the computations forced us to solve individual cases in parallel. The simple strategy we used may be effective for other so called *tree search* problems. Finally, we observe experimentally that among our unrealizable simplicial complexes, the most difficult to show unsatisfiable are those with the simplest topology.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada and MI-TACS, and by the Canada Research Chair program, and made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (http://www.sharcnet.ca/).

References

- Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Günter M. Ziegler, Oriented Matroids, Cambridge University Press, second edition, 1999.
- [2] David Bremner, Jürgen Bokowski, and Gábor Gévay, Symmetric matroid polytopes and their generation, European Journal of Combinatorics, **30** (2009) no. 8, 1758–1777.
- [3] David Bremner and Lars Schewe, Edge-graph diameter bounds for convex polytopes with few facets, Experimental Mathematics, to appear (2011). arXiv:0809.0915.
- [4] George B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J. (1963).
- [5] Niklas Eén and Niklas Sörensson, MiniSat HP, http://minisat.se/
- [6] Friedrich Eisenbrand, Nicolai Hähnle, Alexander Razborov, and Thomas Rothvoß, *Diameter of polyhedra: limits of abstraction*, Mathematics of Operations Research, **35** (2010), no. 35, 786-794.

- [7] Oliver Friedmann, Thomas Hansen, and Uri Zwick, Subexponential lower bounds for randomized pivoting rules for the simplex algorithm, In Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC'11, San Jose, CA, USA, (2011).
- [8] Paul R. Goodey, Some upper bounds for the diameters of convex polytopes, Israel Journal of Mathematics 11 (1972), no. 4, 380-385.
- [9] Branko Grünbaum, Convex Polytopes, 2nd ed., Graduate Texts in Mathematics, vol. 221, Springer-Verlag, New York, 2003. Prepared and with a preface by Volker Kaibel, Victor Klee and Günter M. Ziegler, 341-355.
- [10] Fred Holt and Victor Klee, Many polytopes meeting the conjectured Hirsch bound, Discrete and Computational Geometry 20 (1998), 1-17.
- [11] Gil Kalai and Daniel J. Kleitman, A quasipolynomial bound for the diameter of graphs of polyhedra, Bulletin of the American Mathematical Society 26 (1992), no. 2, 315-316.
- [12] Edward D. Kim, Polyhedral graph abstractions and an approach to the Linear Hirsch Conjecture, available at arXiv:1103.3362.
- [13] Edward D. Kim and Francisco Santos, An update on the Hirsch conjecture, Jahresbericht der Deutschen Mathematiker-Vereinigung, **112** (2010), no. 2, 73-98.
- [14] Victor Klee, Diameters of polyhedral graphs, Canadian Journal of Mathematics 16 (1964), 602-614.
- [15] Victor Klee and David W. Walkup, The d-step conjecture for polyhedra of dimension d < 6, Acta Mathematica **117** (1967), no. 1, 53-78.
- [16] Jesús A. De Loera, Edward D. Kim, Shmuel Onn, and Francisco Santos, *Graphs of transportation polytopes*, Journal of Combinatorial Theory, Series A **116** (2009), no. 8, 1306-1325.
- [17] Benjamin Matschke, Francisco Santos, and Christophe Weibel, The width of 5-prismatoids and smaller non-Hirsch polytopes, http: //www.cs.dartmouth.edu/~weibel/hirsch.php (2011).
- [18] Francisco Santos, A counter-example to the Hirsch conjecture, available at arXiv:1006.2814.
- [19] Lars Schewe, *Satisfiability Problems in Discrete Geometry*, Dissertation, TU Darmstadt, 2007.

- [20] Lars Schewe, Non-realizable minimal vertex triangulations of surfaces: Showing non-realizability using oriented matroids and satisfiability solvers, Discrete and Computational Geometry 43 (2009) no. 2, 289-302.
- [21] Yinyu Ye, The simplex method is strongly polynomial for the Markov decision problem with a fixed discount rate, Available at http://www.stanford. edu/~yyye/simplexmdp1.pdf.

On the generation of topological (n_k) -configurations

Jürgen Bokowski *

Vincent Pilaud [‡]

Abstract

An (n_k) -configuration is a set of n points and n lines in the projective plane such that the point – line incidence graph is k-regular. The configuration is geometric, topological, or combinatorial depending on whether lines are considered to be straight lines, pseudolines or just combinatorial lines.

We provide an algorithm for generating all combinatorial (n_k) -configurations that admit a topological realization, for given n and k. This is done without enumerating first all combinatorial (n_k) -configurations.

Among other results, our algorithm enables us to confirm, in just one hour with a Java code of the second author, a satisfiability result of Lars Schewe in [11], obtained after several months of CPU-time.

1 Introduction

An (n_k) -configuration (P, L) is a set P of n points and a set L of n lines such that each point of P is contained in k lines of L and each line of L contains k points of P. Two lines of L are allowed to meet in at most one point of P and two points of P can lie in at most one common line of L. According to the underlying incidence structure, we distinguish three different levels of configurations, in increasing generality:

- Geometric configurations: Points and lines are ordinary points and lines in the real projective plane \mathbb{P} .
- Topological configurations: Points are ordinary points in \mathbb{P} , but lines are *pseudolines*, *i.e.* non-separating simple closed curves of \mathbb{P} .
- Combinatorial configurations: Points and lines are just required to form an abstract incidence structure (P, L) as described above.

The study of point-line configurations has a long history in discrete 2-dimensional geometry. We refer to Branko Grünbaum's recent monograph [8] for a detailed treatment of the topic. The current challenge is to determine for which values of n do geometric, topological, and combinatorial (n_k) -configurations exist for a given k, and to enumerate and classify them.

For k = 3, the existence of (n_3) -configurations is well understood: combinatorial (n_3) -configurations exist for every $n \geq 7$, but topological and geometric (n_3) -configurations exist only for every $n \ge 9$. For example, Fano's combinatorial (7_3) -configuration cannot be realized as a topological configuration. As further examples, Pappus' and Desargues' theorems form famous (9_3) - and (10_3) -configurations respectively. This description is still almost complete for k = 4: combinatorial (n_4) -configurations exist iff n > 13, topological (n_4) -configurations exist iff $n \ge 17$ [3] and geometric (n_4) -configurations exist iff $n \ge 18$ [7, 4], with the possible exceptions of 19, 22, 23, 26, 37 and 43. For general k, the situation is more involved, and the existence of combinatorial, topological and geometric (n_k) -configurations is not determined in general.

In this paper, we describe an algorithm for generating, for given n and k, all combinatorial (n_k) -configurations that admit a topological realization, without enumerating first all combinatorial (n_k) -configurations. The algorithm sweeps the projective plane to construct a topological (n_k) -configuration (P, L), but only considers as relevant the events corresponding to the sweep of points of P. This strategy enables us to identify along the way some topological configurations which realize the same combinatorial configuration, and thus to maintain a reasonable computation space and time.

We developed two different implementations of this algorithm. The first one was written in Haskell by the first author to develop the strategy of the enumeration process. Once the general idea of the algorithm was settled, the second author wrote another implementation in Java, focusing on the optimization of computation space and time of the process.

We underline three motivations for this algorithm. First, the algorithm is interesting in its own right. Before describing some special methods for constructing topological configurations, Branko Grünbaum writes in [8, p. 165] that "the examples of topological configurations presented so far have been ad hoc, obtained essentially through (lots of) trial and error". Our algorithm can reduce considerably the trial and error method. Second our algorithm enables us to check and confirm the previous results obtained in earlier papers, e.g. for k = 4 and $n \leq 18$ in [4, 11]. We can use a single method and reduce considerably the computation time (e.g. the computation of the (18₄)-configurations

^{*}Tech. Univ. Darmstadt, juergen.bokowski@googlemail.com [‡]Univ. Paris 7, vincent.pilaud@liafa.jussieu.fr, Research supported by Spanish MEC grant MTM2008-04699-C03-02.



Figure 1: Two rather different (18_4) -configuration which are combinatorially equivalent.

needed several months of CPU-time in [11], and only one hour with our Java implementation). Finally, our algorithm opens new opportunities of research based on enumeration to answer several open questions on configurations. Among others: Is there a symmetrical topological (19₄)-configuration [8, p. 169]? What is the smallest topological (n_5)-configuration? Is there a geometric (19₄)-configuration?

Topological configurations are pseudoline arrangements, or rank 3 oriented matroids. We assume the reader to have some basic knowledge on these topics see [2, 1, 9].

2 Preliminaries

What do we need? Our guideline and motivation in the study of configurations is the question of the existence of geometric (n_k) -configurations. In particular, it is challenging to determine, for a given k, which is the first n for which geometric (n_k) -configurations exist. For k = 3, Pappus' configuration is the first example (with two other combinatorially distinct (9₃)-configurations). For k = 4, it was known for a long time that no combinatorial (n_4) -configurations exist when $n \leq 12$. However, the smallest geometric configuration was unknown until the first author proved with Lars Schewe that no topological (n_4) -configurations exist when $n \leq 16$ [4], that the only combinatorial (17_4) -configuration which is topologically realizable is not geometrically realizable [5], and that there exists a geometric (18_4) -configuration [5].

The method presented in [5] makes it possible to decide whether a combinatorial configuration is geometrically realizable. The goal of our algorithm is to limit the research to combinatorial configurations which are already topologically realizable. In other words, for given n and k, we want to enumerate all topological (n_k) -configurations under combinatorial equivalence. **Three equivalence relations.** There are three distinct notions of equivalence on topological configurations.

The finest notion is the usual notion of equivalence between pseudoline arrangements in the projective plane: two configurations are *topologically equivalent* if there is an homeomorphism of their underlying projective planes that sends one arrangement onto the other.

The coarsest notion is combinatorial equivalence: two (n_k) -configurations are *combinatorially equivalent* if they realize the same combinatorial (n_k) -configuration.

The intermediate notion is based on the graph of admissible mutations. Remember that a *mutation* in a pseudoline arrangement is a local transformation of the arrangement where only one pseudoline ℓ moves, sweeping a single vertex v of the remaining arrangement. It only changes the position of the crossings of ℓ with the pseudolines incident to v. If those crossings are all 2-crossings, the mutation does not perturb the k-crossings of the arrangement, and thus produces another topological (n_k) -configuration. We say that such a mutation is *admissible*. Two configurations are *mutation equivalent* if they belong to the same connected component of the graph of admissible mutations.



Figure 2: An admissible mutation.

Obviously, topological equivalence implies mutation equivalence, which in turn implies combinatorial equivalence. The reciprocal implications are wrong.

Note that the topological equivalence between two (n_k) -configurations can be tested in $\Theta(n^3)$ time. Indeed, since the topological configurations are embedded

on the projective plane, the images of two pseudolines under an isomorphism of the projective plane determine the images of all the other pseudolines. Thus, the complexity to compute the topological equivalence classes among p topological (n_k) -configurations is in $\Theta(p^2n^3)$. Both combinatorial and mutation equivalences are however much harder to decide computationally.

In order to limit unnecessary computation, we can use topological (resp. mutation, resp. combinatorial) invariants associated to topological configurations. If two configurations have distinct invariants, they cannot be equivalent. Reciprocally, if they share the same invariant, it provides us information on the possible isomorphism between these two configurations. For example, the *face size vector* (the number of faces of each size) is a topological invariant, and the *distribution of the triangles* on the pseudolines is a combinatorial invariant (a triangle of a configuration (P, L) is a triple of points of P which are pairwise related by pseudolines of L).

As an illustration, the two (18_4) -configurations depicted in Figure 1 are combinatorially equivalent (the labels on the pseudolines provide a combinatorial isomorphism) but not topologically equivalent (the left one has 22 quadrangles and 2 pentagons, while the right one has 23 quadrangles). In fact, one can even check that they are not mutation equivalent.

What do we obtain? Our algorithm can enumerate all topological (n_k) -configurations up to either topological or combinatorial equivalence. In order to maintain a reasonable computation space and time, the main idea is to focus on the relative positions of the points of the configurations and to ignore at first the relative positions of the other crossings among the pseudolines. In other words, to work modulo mutation equivalence.

More precisely, we first enumerate at least one representative of each mutation equivalence class of topological (n_k) -configuration. From these representatives, we can derive:

- 1. all topological (n_k) -configurations up to topological equivalence: we explore each connected component of the mutation graph with our representatives as starting nodes.
- 2. all combinatorial (n_k) -configurations that are topologically realizable: we reduce the result modulo combinatorial equivalence.

3 Representation of arrangements

Simple configurations. A topological configuration (P, L) is *simple* if no three pseudolines of L meet at a common point except if it is a point of P. Since any topological (n_k) -configuration can be arbitrarily perturbed to become simple, we only consider simple topo-

logical (n_k) -configurations. Once we obtain all simple topological (n_k) -configurations, it is usual to obtain all (non-necessarily simple) topological (n_k) -configurations up to topological equivalence by exploring the mutation graph, and we do not report on this aspect.

In a simple (n_k) -configuration (P, L), there are two kinds of intersection points among pseudolines of L: the points of P, which we also call *k*-crossings, and the other points, which we call 2-crossings. Each pseudoline of Lcontains k k-crossings and n - 1 - k(k - 1) 2-crossings. In total, a simple (n_k) -configuration has n k-crossings and $\binom{n}{2} - n\binom{k}{2} - 1$ 2-crossings.

Segment length distributions. A segment of a topological configuration (P, L) is the portion of a pseudoline of L located between two consecutive points of P. If (P, L) is simple, a segment contains no k-crossing except its endpoints, but may contain some 2-crossings. The *length* of a segment is the number of 2-crossings it contains.

The lengths of the segments of a pseudoline of L form a k-partition of n - 1 - k(k - 1). We call a maximal representative of a k-tuple the lexicographic maximum of its orbit under the action of the dihedral group (*i.e.* rotations and reflections of the k-tuple). We denote by Π the list of all distinct maximal representatives of the k-partitions of n - 1 - k(k - 1), ordered lexicograhically. For example, when k = 4 and n = 17, $\Pi = [4, 0, 0, 0], [3, 1, 0, 0], [3, 0, 1, 0], [2, 2, 0, 0], [2, 0, 2, 0],$ [2, 1, 1, 0], [2, 1, 0, 1], [1, 1, 1, 1].

A suitable representation. We represent the projective plane as a disk where we identify antipodal boundary points. Given a simple topological (n_k) -configuration (P, L), we fix a representation of its underlying projective plane which satisfies the following properties (see Figure 3 left).

The leftmost point of the disk (which is identified with the rightmost point of the disk) is a point of P, which we call the *basepoint*. The k pseudolines of L passing through the basepoint are called the *frame pseudolines*, while the other n - k pseudolines of L are called *working pseudolines*. The frame pseudolines decompose the projective plane into k connected regions which we call *frame regions*. A crossing is a *frame* crossing if it involves a frame pseudoline and a *working* crossing if it involves only working pseudolines.

The boundary of the disk is a frame pseudoline, which we call the *baseline*. We furthermore assume that the segment length distribution Λ on the top half-circle appears in Π (*i.e.* is its own maximal representative), and that no maximal representative of the segment length distribution of a pseudoline of L appears before Λ in Π . In particular, the leftmost segment of the baseline is a longest segment of the configuration.



Figure 3: Suitable representation of a (17_4) -configuration, and the corresponding wiring diagram.

Wiring diagram and allowable sequence. Another interesting representation of our (n_k) -configuration is the wiring diagram [6] of its working pseudolines (see Figure 3 right). It is obtained by sending the basepoint to infinity in the horizontal direction. The frame pseudolines are k horizontal lines, and the n - k working pseudolines are vertical wires. The orders of the working pseudolines on a horizontal line sweeping the wiring diagram from top to bottom form the so-called allowable sequence of the working arrangement, as defined in [6].

4 Description of the algorithm

Main idea. Let us recall here the main idea of the algorithm: to enumerate (n_k) -configurations, we focus on the relative positions of the k-crossings and ignore at first the relative positions of the 2-crossings. More precisely, we first generate at least one (but as few as possible) representative of each mutation equivalent class of (n_k) -configurations.

Sweeping process. The algorithm sweeps the projective plane to construct a topological (n_k) -configuration. The *sweepline* sweeps the configuration from the baseline on the top of the disk to the baseline on the bottom of the disk. It always passes through the basepoint and always completes the configuration into an arrangement of n + 1 pseudolines. In other words, it sweeps the k frame regions from top to bottom, reaching the separating frame pseudoline when passing from one frame region to the next one, and discovers along the way the working pseudolines. Except those located on the frame pseudolines, we assume that the crossings of the configuration.

ration are reached one after the other by the sweepline. After the sweepline swept a crossing, we remember the order of its intersections with the working pseudolines. In other words, the sweeping process provides us with the allowable sequence of the working pseudolines of our configuration.

Since any admissible mutation is irrelevant for us, we only focus on the steps of the sweeping process where our sweepline sweeps a k-crossing. Thus, two different events can occur: when the sweepline sweeps a working k-crossing, and when the sweepline sweeps a frame pseudoline. In the later case, we sweep simultaneously k-1frame k-crossings (each involving the frame pseudoline and k-1 working pseudolines), and n-1-k(k-1)frame 2-crossings (each involving the frame pseudoline and a working pseudoline). Between two such events, the sweepline may sweep working 2-crossings which are only taken into account when we reach a new event. Let us repeat again that the precise positions of these working 2-crossings is irrelevant in our enumeration.

To obtain all possible solutions, we maintain a priority queue with all subconfigurations which have been constructed so far, remembering for each one (i) the order of the working pseudolines on the current sweepline, (ii) the number of frame and working k-crossings and 2-crossings which have already been swept on each working pseudoline, (iii) the length of the segment currently swept by the sweepline, and (iv) the history of the sweeps which have been performed to reach this subconfiguration. At each step, we remove the first subconfiguration from the priority queue, and insert all admissible subconfigurations which can arise after sweeping a new working k-crossing or a new frame pseudoline. We finally accept a configuration once we have swept k frame pseudolines and n - k(k - 1) - 1 working k-crossings.



Figure 4: Sweeping a working k-crossing (left) and a frame pseudoline (right).

Any subconfiguration considered during the algorithm is a potential (n_k) -configuration. Throughout the process, we make sure that any pair of working pseudolines cross at most once, that the number of frame pseudolines (resp. of working k-crossings) already swept never exceeds k (resp. n - 1 - k(k - 1)), and that the total number of working 2-crossings never exceeds (n - 2k)(n - 1 - k(k - 1))/2. Furthermore, on each pseudoline, the number of frame and working k-crossings (resp. 2-crossings) already swept never exceeds k (resp. n - 1 - k(k - 1)), the number of working 2- and k-crossings already swept never exceeds n - 1 - k(k - 1), and the segment currently swept is not longer than the leftmost segment of the baseline.

Initialization. We initialize our algorithm sweeping the baseline. We only have to choose the distribution of the lengths of the segments on the baseline. The possibilities are given by the list Π of maximal representatives of k-partitions of n - 1 - k(k - 1).

Sweep a working k-crossing. If we decide to sweep a working k-crossing, we have to choose the k working pseudolines which intersect at this k-crossing, and the direction of the other working pseudolines.

Since we are allowed to perform any admissible mutation, we can assume that all the pseudolines located to the left of the leftmost pseudoline of the working k-crossing, and all those located to the right of the rightmost pseudoline of the working k-crossing do not move.

We say that the pseudolines located between the leftmost and the rightmost pseudolines of the working k-crossing form the *kernel* of the working k-crossing. We have to choose the positions of the pseudolines of the kernel after the flip: each pseudoline of the kernel either belongs to the working k-crossing, or goes to its left, or goes to its right (see Figure 4 left).

A choice of directions for the kernel is admissible provided that (i) each pseudoline involved in the k-crossing can still accept a working k-crossing; (ii) each pseudoline of the kernel can still accept as many working 2-crossings as implied by the choice of directions for the kernel; (iii) no segment becomes longer than the leftmost segment of the baseline; and (iv) any two pseudolines which are forced to cross by the choice of directions for the kernel did not cross earlier (*i.e.* they still form an inversion on the sweepline before we sweep the working k-crossing).

Sweep a frame pseudoline. If we decide to sweep a frame pseudoline, we have to choose the $(k - 1)^2$ working pseudolines involved in one of the k - 1 frame k-crossings, and the direction of the other working pseudolines.

As before, we can assume that a pseudoline does not move if it is located to the left of the leftmost pseudoline involved in one of the k-1 frame k-crossings, or to the right of the rightmost pseudoline involved in one of the k-1 frame k-crossings. Otherwise, we can perform admissible mutations to ensure this situation.

The other pseudolines form again the *kernel* of the frame sweep, and we have to choose their positions after the flip. Each pseudoline of the kernel either belongs to one of the k-1 frame k-crossings, or can choose among k possible directions: before the first frame k-crossing, or between two consecutive frame k-crossings, or after the last frame k-crossing (see Figure 4 right).

As before, a choice of directions for the kernel is admissible if (i) each pseudoline involved (resp. not involved) in one of the k - 1 frame k-crossings can still accept a frame k-crossing (resp. a frame 2-crossing); (ii) each pseudoline of the kernel can still accept as many working 2-crossings as implied by the choice of directions for the kernel; (iii) no segment becomes longer than the leftmost segment of the baseline; and (iv) any two pseudolines which are forced to cross by the choice of directions for the kernel did not cross earlier (*i.e.* they still form an inversion on the sweepline before we sweep the frame pseudoline).

Sweep the last frame region. Our sweeping process finishes once we have swept n - 1 - k(k - 1) working k-crossings and k frame pseudolines. Each resulting subconfiguration should still be completed into a topological (n_k) -configuration with some necessary remaining 2-crossings. More precisely, we need to add on each working pseudoline as many working 2-crossings as its number of inversions in the permutation given by the working pseudolines on the final sweepline, without creating segments that are too long.

All the constructed configurations are guaranteed to be valid topological (n_k) -configurations. To make sure that we indeed obtain the representation presented in Section 3, we remove each configuration (P, L) in which the maximal representative of the segment length distribution of a pseudoline of L appears in the list Π before the segment length distribution of its baseline.

Parallelization. To close this description, we observe that our algorithm is easily parallelizable on different computers since it is a dynamic research in a tree. We did not use parallelization to obtain the current results, but it will certainly be an important advantage of the algorithm for exploring the question of finding the first integer n for which topological (n_5) -configurations exist.

5 Results

Check former results. As a first application, our algorithm enables us to check easily all former enumerations of topologically realizable combinatorial (n_k) -configurations. The Java implementation developed by the second author finds all (n_k) -configurations in less than a minute¹ when k = 3 and $n \leq 11$, or when k = 4 and $n \leq 17$. In particular, we checked that there is no topological (n_4) -configuration when $n \leq 16$ [4], and that there is a single combinatorial (17_4) -configuration which can be realized by (several) topological (17_4) -configurations, but which cannot be realized geometrically [5]. When k = 4 and k = 18, we reconstructed the 16 combinatorial equivalence classes of topological (18_4) -configurations depicted in [5, Figure 6]. To obtain this result, our implementation needed about one hour¹, compared to months of CPU-time used in [11]. The two (18_4) -configurations presented in Figure 1, which are combinatorially equivalent but not mutation equivalent, occured while we were reducing the list of (18_4) -configurations up to combinatorial equivalence, using as a first reduction a certain invariant of mutation equivalence.

Obtain new results. Our algorithm can furthermore be used to derive new enumerative results. To answer the question of the existence of geometric (19_4) -configurations, our first step is to compute the complete list of combinatorial (19_4) -configurations that admit a topological realization. We found 4028 combinatorially distinct topologically realizable (19_4) -configurations (222 of which are self-dual). This task has been accomplished by our algorithm within 16 days of CPU-time¹. We underline again that we did not use the extended list of all 269224652 combinatorial (19_4) -configurations computed in [10] to obtain this result. A detailed investigation and analysis of this result will be published in a subsequent paper.

Acknowledgements

The first author thanks Leah Berman from the University of Alaska Fairbanks for discussions about the subject. He also thanks three colleagues from the Universidad Nacional Autónoma de México, namely Rodolfo San Augustin Chi, Ricardo Strausz Santiago, and Octavio Paez Osuna, for many stimulating discussions about various different earlier versions of the presented algorithm during his one year sabbatical stay (2008/2009) in México City.

References

- A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. Oriented matroids, volume 46 of Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, second edition, 1999.
- [2] J. Bokowski. Computational oriented matroids. Cambridge University Press, Cambridge, 2006.
- [3] J. Bokowski, B. Grünbaum, and L. Schewe. Topological configurations (n₄) exist for all n ≥ 17. European J. Combin., 30(8):1778–1785, 2009.
- [4] J. Bokowski and L. Schewe. There are no realizable 15₄and 16₄-configurations. *Rev. Roumaine Math. Pures Appl.*, 50(5-6):483–493, 2005.
- [5] J. Bokowski and L. Schewe. On the finite set of missing geometric configurations (n₄). To appear in *Computational Geometry: Theory and Applications*, 2011.
- [6] J. E. Goodman and R. Pollack. Allowable sequences and order types in discrete and computational geomtry. In New trends in discrete and computational geometry, volume 10 of Algorithms Combin., pages 103–134. Springer, Berlin, 1993.
- [7] B. Grünbaum. Connected (n₄) configurations exist for almost all n—second update. Geombinatorics, 16(2):254–261, 2006.
- [8] B. Grünbaum. Configurations of points and lines, volume 103 of Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2009.
- [9] D. E. Knuth. Axioms and hulls, volume 606 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.
- [10] O. Páez Osuna and R. San Agustín Chi. The combinatorial (19₄) configurations. Preprint, 2011.
- [11] L. Schewe. Satisfiability Problems in Discrete Geometry. PhD thesis, Technische Universität Darmstadt, 2007.

¹Computation times on a double core processor on 2.4 GHz.

Sliding labels for dynamic point labeling

Andreas Gemsa^{*}

Martin Nöllenburg*

Ignaz Rutter*

Abstract

We study a dynamic labeling problem for points on a line that is closely related to labeling of zoomable maps. Typically, labels have a constant size on screen, which means that, as the scale of the map decreases during zooming, the labels grow relatively to the set of points, and conflicts may occur due to overlapping labels. Our algorithmic problem is a combined dynamic selection and placement problem in a sliding-label model: (i) select for each label ℓ a contiguous *active range* of map scales at which ℓ is displayed, and (ii) place each label at an appropriate position relative to its anchor point by sliding it along the point. The active range optimization (ARO) problem is to select active ranges and slider positions so that no two labels intersect at any scale and the sum of the lengths of active ranges is maximized. We present a dynamic programming algorithm to solve the discrete k-position ARO problem optimally and an FPTAS for the continuous sliding ARO problem.

1 Introduction

With the increasing practical importance of dynamic maps that allow continuous operations like zooming, panning, or rotations, dynamic labeling of map features becomes a critical aspect of the visual quality of a map. Examples of dynamic maps range from maps on smallscreen mobile devices to professional desktop GIS applications. The map dynamics add new dimensions to label placement, which result in challenging geometric optimization problems that are quite different from static labeling problems. Changes in dynamic maps due to continuous map movements need to be smoothly animated in order to preserve a coherent context and minimize the user's cognitive load for re-orientation [12]. This requirement is also known as "frame coherency" [2] or "temporal continuity" [5]. Hence we cannot simply solve the arising labeling problems independently for each intermediate map view during the animation; rather we need to solve the labeling problem globally such that the animations of all possible trajectories using the given set of navigation operations satisfy the quality constraints.

In order to avoid distraction and irritation of the user a dynamic map should—according to Been et al. [3]— adhere to the following quality constraints or *consistency* desiderata for dynamic map labeling: During monotone map movement labels should neither "jump" (noncontinuously change position or size) nor "pop" (vanish when zooming in or appear when zooming out); moreover, the labeling should be a function of the selected map viewport and not depend on the navigation history. In this paper we are only interested in dynamic labelings that are consistent in that sense. Of course each static map view in a dynamic map also needs to satisfy the quality standards for static maps [8], i.e., all labels—usually modeled as rectangles—are pairwise disjoint, each label is close to its anchor point, and, globally over all possible map views, the number of visible labels is maximum.

Been et al. [4] presented a first extensive study of algorithms for dynamic map labeling in several different models for one- and two-dimensional input point sets. However, they focused on the *dynamic label selection* problem, i.e., which set of labels to select at which scale, and assumed that for each label a single, fixed position relative to the anchor point is given in the input. They left dynamic label placement as an open problem, i.e., the problem where to place each label relative to its anchor point. One model for label placement is the k*position* or *fixed-position* model, where each label can be placed at a position from a set of k (usually 4 or 8) possible positions [1, 6, 17]. Another more general model is the *slider* model, where the finite-position assumption is dropped and each label can take any position such that the anchor point coincides with a point on the label boundary [15, 16]. In this paper we present labeling algorithms in a dynamic scenario that allows continuous zooming for the visualization of a one-dimensional input point set. To the best of our knowledge our algorithms are the first to combine the dynamic label selection problem with label placement in both the fixed-position and the slider model, thus answering (partially) an open question of Been et al. [4].

Related Work. Most previous algorithmic research on automated label placement deals with *static* fixedposition or slider models for point, line, or area features. The problem of maximizing the number of selected labels is NP-hard even for the simplest labeling models, whereas there are efficient algorithms for the decision problem that asks whether all points can be labeled in

^{*}Department of Computer Science, Karlsruhe Institute of Technology (KIT), Germany, {gemsa, noellenburg, rutter}@kit.edu

some of the simpler models (see, e.g., the discussion by Klau and Mutzel [9] or the comprehensive map labeling bibliography [19]). Approximation results [1, 16], heuristics [18], and exact approaches [9] are known for many variants of the static label number maximization problem.

More recently, *dynamic* map labeling has emerged as a new research topic that gives rise to many unsolved algorithmic problems. Petzold et al. [13] used a preprocessing step to generate a reactive conflict graph that represents possible label overlaps for maps of all scales. For any fixed scale and map region, their method computes a conflict-free labeling in the slider model using heuristics. Poon and Shin [14] described algorithms for labeling one- and two-dimensional point sets that precompute a hierarchical data structure storing solutions for a number of different scales; this allows them to answer adaptive zooming queries efficiently. Mote [10] presented another fast heuristic method for dynamic conflict resolution in label placement that does not require preprocessing and assumes a 4-position model. The consistency desiderata of Been et al. [3] for dynamic labeling, however, are not satisfied by any of these three methods. Been et al. [4] showed NP-hardness of the label number maximization problem in the consistent labeling model and presented several approximation algorithms for labeling two-dimensional point sets and an exact algorithm for one-dimensional point sets. They focused on dynamic label selection, i.e., assumed a 1-position model for label placement. Nöllenburg et al. [11] recently studied a dynamic version of the alternative boundary labeling model allowing continuous zooming and panning, where labels are placed at the sides of the map and connected to their points by leaders. Algorithms and complexity results for dynamic label selection in fixed-scale rotating maps that satisfy similar consistency desiderata were presented by Gemsa et al. [7].

Contribution. In this paper we present algorithms for labeling a set of points on a line with labels of arbitrary, non-uniform length in a dynamic scenario that supports continuous zooming of the points' visualization. Unlike previous efforts [4] we consider label placement in a k-position and slider model: we must select both an interval of scales at which each label is selected (dynamic selection problem) and an admissible label position for each label relative to its anchor point (dynamic placement problem). We require that the label position remains the same for all scales. In Section 2 we introduce a model for dynamic point labeling with sliding labels in the framework of Been et al. [3]. Section 3 presents a dynamic programming algorithm for dynamically labeling points in the k-position model. Our main contribution is the fully polynomial-time approximation scheme (FP-TAS) described in Section 4 for the more general sliding

206

model. We conclude in Section 5 with several remaining open questions that arise from our results.

2 Preliminaries

In this section we describe our model for dynamic labeling in the general framework of Been et al. [3, 4].

Model. Let $P = \{p_1, \ldots, p_n\}$ be a set of points on the x-axis (also called the *base line*) together with a set $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ of labels. The point p_i is called the anchor point of the label ℓ_i . Each label ℓ_i is a rectangle of (target) width w_i modeling the bounding box of the text describing the point p_i . Since we focus on labeling a one-dimensional point set, we can think of each label ℓ_i as actually being a line segment of width w_i . During zooming of the points' visualization we wish to keep the label size constant on screen, which means that if we scale the map by a factor of 1/s we need to increase the label size by a factor of s in order to maintain its width on screen constant. This is the label size invariance property of Been et al. [3]. So the width of ℓ_i on the base line required for a map of scale 1/s is given by the linear function $w_i(s) = w_i s$.

The label proximity constraint in map labeling says that each label must be close to its anchor point [8], i.e., we require for each label that the anchor point coincides with a point of the label. We consider sliding labels and define the shift position $t_i \in [0, 1]$ of a label ℓ_i as the fraction of ℓ_i that is to the right of p_i . For $t_i = 0$ the label is in its leftmost position, and for $t_i = 1$ it is in its rightmost position. In the fixed-position model only a finite subset of positions from [0, 1] is allowed. In this paper we consider *invariant point placements* [3], i.e., once a shift position t is selected for a label ℓ , ℓ maintains that position relative to its anchor point. This immediately prevents the labels from jumping. Figure 1 shows a set of five points with labels zoomed to four different scales. Note that as the scale decreases, the points move closer together and some labels must be removed to avoid conflicting labels.



Figure 1: Five (partially) labeled points on a line zoomed from smaller (top) to larger scales (bottom).

Following Been et al. [3, 4] we define an extended twodimensional coordinate system defined by the *x*-axis,



Figure 2: Triangular truncated extrusions (shaded blue) induced by the example of Figure 1.

which models the positions of the points P, and the y-axis, which models the inverse s of the scale 1/s. We denote s as the *scale factor* that is used to enlarge the labels before the whole base line (including the labels) is scaled down by the target scale 1/s to produce the actual visualization. We say a label is *active* at scale factor s if it is selected as being visible at s; otherwise it is *inactive*. The (static) *placement* of an active label ℓ with target width w and anchor point p at scale factor sis determined by a shift position $t \in [0, 1]$, i.e., the label is represented by the interval [p - (1 - t)ws, p + tws]. A dynamic placement of ℓ is a placement of ℓ for each scale factor s at which ℓ is active. Since we consider invariant point placements, the shift position is the same for all scales. If we extrude the growing label segment with its constant shift position t along the y-axis from y = 0 to some maximum scale factor $s_{\rm max}$ we obtain a triangle whose apex is placed at the point p and whose top side is parallel to the x-axis, see Figure 2. We call this triangle the extrusion E of ℓ . The shift position t determines the slant of E, but for a label ℓ of width w the width of E at any fixed scale factor s is ws independent of t. Let the trace $\operatorname{tr}_s(E)$ of E at scale factor s be the intersection of E with the horizontal line y = s. By definition $\operatorname{tr}_{s}(E)$ corresponds to the placement of ℓ at s if ℓ is active at s.

If the extrusions E and E' of two labels intersect at some scale factor s this means that the two labels ℓ and ℓ' overlap at scale 1/s. A standard requirement in point labeling, however, is that all labels must be pairwise disjoint [8]. Accordingly, at most one of ℓ or ℓ' can be active at scale factor s. Since one of the desiderata for consistent dynamic map labeling is that labels do not 'pop' during monotonous zooming in order to avoid flickering effects [3] we require that labels never vanish when zooming in and never appear when zooming out. This lets us define the *active range* of a label ℓ_i as an interval of scale factors $[0, a_i)$ for which ℓ_i is active. This active range implies that when zooming in the label ℓ_i appears exactly once at scale factor a_i and then remains active, or, conversely, when zooming out it disappears exactly once at scale factor a_i and remains inactive. The truncated extrusion T_i is the restriction of the extrusion E_i of ℓ_i to its active range $[0, a_i)$, see Figure 2 for an example. Now a consistent dynamic labeling for the points P corresponds to an assignment of a scale-independent shift position t_i and an active range $[0, a_i)$ for each label ℓ_i such that the truncated extrusions $\mathcal{T} = \{T_1, \ldots, T_n\}$ are pairwise disjoint. Hence we need to solve both a dynamic selection problem and a dynamic placement problem according to Been et al. [3]. Informally speaking, we can adjust the slant and the height of the truncated extrusions as long as they do not intersect each other.

Objective. A common objective in point labeling is to maximize the number of labeled points, and accordingly our goal is to maximize the *total active range length*, which is defined as the sum $H = \sum_{i=1}^{n} a_i$ of all active range lengths. Maximizing H corresponds to displaying a maximum number of labels integrated over all scale factors $s \in [0, s_{\max}]$. This problem is known as the active range optimization problem (ARO) [4]. We consider two one-dimensional variants of ARO: In the discrete *k*-position 1d ARO problem the set of admissible shift positions is restricted to a subset $S_i \subset [0, 1]$ of cardinality $|S_i| \leq k$. In the general sliding 1d ARO problem any shift position in [0, 1] is admissible.

3 A dynamic program for *k*-position 1d ARO

In this section we give a dynamic program for computing an optimal solution for the k-position version of the 1d ARO problem. For ease of notation we define two dummy points p_0 and p_{n+1} , where $p_0 = \min\{p_i - s_{\max}w_i \mid 1 \le i \le n\}$ and $p_{n+1} = \max\{p_i + s_{\max}w_i \mid 1 \le i \le n\}$. The only shift position of ℓ_0 is $\mathcal{S}_0 = \{0\}$ and the only shift position of ℓ_{n+1} is $\mathcal{S}_{n+1} = \{1\}$. Both labels have width 1. It is easy to see that in any optimal solution the height of T_0 and T_{n+1} must be s_{\max} since none of the extrusions T_i can intersect T_0 or T_{n+1} .

For a pair of points p_i and p_j with i < j and shift positions $k_i \in S_i$ and $k_j \in S_j$ we define the free space $\Delta(i, j, k_i, k_j)$ as the polygon bounded by the line s = 0, the supporting line of the right edge of T_i in shift position k_j , and, if the two supporting lines of T_i and T_j do not intersect below s_{max} , the line $s = s_{max}$. See Figure 3 for an example. Let $\mathcal{A}[i, j, k_i, k_j]$ be the maximum total active range height for the points p_{i+1}, \ldots, p_{j-1} , where all truncated extrusions T_{i+1}, \ldots, T_{j-1} are contained in $\Delta(i, j, k_i, k_j)$.

We observe that the tallest truncated extrusion T_l (i < l < j) in any optimal solution of the subinstance Iinduced by $\Delta(i, j, k_i, k_j)$ must touch the left, right, or top boundary of $\Delta(i, j, k_i, k_j)$, otherwise we could improve the total active range height. We use T_l in order to split I into two smaller independent subinstances I'and I'' induced by $\Delta(i, l, k_i, k_l)$ and $\Delta(l, j, k_l, k_j)$, see Figure 3. For each $l = i + 1, \ldots, j - 1$ let h_{l,k_l}^{i,j,k_i,k_j} denote



Figure 3: The subinstance induced by $\Delta(i, j, k_i, k_j)$ is split into two smaller independent subinstances by T_l .

the height at which T_l at shift position $k_l \in S_l$ first hits a non-bottom edge of $\Delta(i, j, k_i, k_j)$. We initialize $\mathcal{A}[i, i + 1, \cdot, \cdot] = 0$ for all $i = 0, \ldots, n$ and then recursively define $\mathcal{A}[i, j, k_i, k_j] = \max\{\mathcal{A}[i, l, k_i, k_l] + h_{l,k_l}^{i, j, k_i, k_j} + \mathcal{A}[l, j, k_l, k_j] \mid i < l < j$ and $k_l \in S_l\}$. By definition of \mathcal{A} the solution to the ARO problem is $\mathcal{A}[0, n + 1, 0, 1]$. We can compute the value $\mathcal{A}[0, n + 1, 0, 1]$ by dynamic programming in $O(n^3k^3)$ time: each of the $O(n^2k^2)$ values in \mathcal{A} is defined as the maximum of a set of O(nk) values, each of which can be computed by two table look-ups and two O(1)-time line intersection queries.

The correctness of the above dynamic program follows by induction on the number of points in a subinstance. Clearly for an empty subinstance $\Delta(i, i+1, k_i, k_{i+1})$ the maximum total active range height $\mathcal{A}[i, i+1, k_i, k_{i+1}]$ is 0. Let's consider a subinstance induced by $\Delta(i, j, k_i, k_j)$, where j - i = r and assume by induction that the values in \mathcal{A} are correct for all subinstances $\Delta(i', j', k_{i'}, k_{j'})$, where j' - i' < r. Let \mathcal{B} be an optimal active range assignment of the labels $\ell_{i+1}, \ldots, \ell_{j-1}$ within the free space $\Delta(i, j, k_i, k_j)$ and let $H(\mathcal{B})$ be its value. Let further T_l be a tallest truncated extrusion with shift position k_l in \mathcal{B} . Obviously T_l must have height h_{l,k_l}^{i,j,k_l,k_j} if \mathcal{B} is optimal. Since our algorithm explicitly considers all labels and all shift positions as candidates for the tallest truncated extrusion, it also considers T_l and its shift position k_l , which splits the given instance into two independent subinstances with free spaces $\Delta(i, l, k_i, k_l)$ and
$$\begin{split} &\Delta(l, j, k_l, k_j). \text{ Since } l-i < r \text{ and } j-l < r \text{ we know that} \\ &\mathcal{A}[i, j, k_i, k_j] \geq \mathcal{A}[i, l, k_i, k_l] + h_{l, k_l}^{i, j, k_i, k_j} + \mathcal{A}[l, j, k_l, k_j] = \end{split}$$
 $H(\mathcal{B}).$

Since the free space $\Delta(0, n + 1, 0, 1)$ is chosen such that none of the truncated extrusions T_1, \ldots, T_n can touch T_0 or T_{n+1} , $\mathcal{A}(0, n + 1, 0, 1)$ indeed contains the value of an optimal solution to the k-position ARO problem. We can easily augment the algorithm to keep track of the pair (l, k_l) that achieved the maximum value in order to reconstruct the solution by backtracking from $\mathcal{A}(0, n + 1, 0, 1)$. We summarize this result in the following theorem.

Theorem 1 Given n points $P = \{p_1, \ldots, p_n\}$ on the x-axis, a label ℓ_i of base width w_i for each point p_i , and

a set $S_i \subset [0,1]$ of at most k shift positions for each label ℓ_i , we can compute an optimal solution to the k-position 1d ARO problem in $O(n^3k^3)$ time and $O(n^2k^2)$ space.

We note that this algorithm generalizes the $O(n^3)$ -time algorithm of Been et al. [4] for the 1-position 1d ARO problem, where each label has only a single available shift position.

4 An FPTAS for general 1d sliding ARO

In this section we present an FPTAS for approximating the optimal solution of the sliding 1d ARO problem within a factor of $(1 - \varepsilon)$. The idea of the FPTAS is based on uniformly discretizing the interval [0, 1] of shift positions. Let k > 0 be an integer and define the set of shift positions $\mathcal{S}^k = \{i/k \mid i \in \mathbb{Z}, 0 \le i \le k\}$. For an instance I of the sliding 1d ARO problem consisting of a point set $P = \{p_1, \ldots, p_n\}$ and corresponding label set \mathcal{L} , we consider instead the (k+1)-position 1d ARO problem for the instance I' consisting of P, \mathcal{L} , and the shift position sets $S_i = S^k$ for $1 \leq i \leq n$. By Theorem 1 this instance I' can be solved in $O(n^3k^3)$ time and $O(n^2k^2)$ space using the dynamic programming algorithm of Section 3. In the following theorem we show that this approach gives an FPTAS for the original sliding 1d ARO problem.

Theorem 2 Given n points $P = \{p_1, \ldots, p_n\}$ on the x-axis and a corresponding label set $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$, where label ℓ_i has base width w_i , we can compute a $(1-\varepsilon)$ -approximate solution to the sliding 1d ARO problem in $O(n^3(1/\varepsilon)^3)$ time and $O(n^2(1/\varepsilon)^2)$ space.

Proof. We need to show that for a suitably chosen parameter $k = k(\varepsilon)$ the optimal solution for the (k + 1)position 1d ARO instance I' as defined above is actually a $(1 - \varepsilon)$ approximate solution for the sliding 1d ARO instance I. Let us assume that we know an optimal solution A^* for I, i.e., a shift position $t_i \in [0, 1]$ and an active range $[0, a_i) \subseteq [0, s_{\max}]$ for each label ℓ_i . Since A^* is optimal, each truncated extrusion T_i has either height $a_i = s_{\max}$ or touches the left or right supporting line of another, taller, truncated extrusion T_j . In the latter case a_i is the smallest scale factor, where the extrusions E_i and E_j intersect.

For proving the approximation factor we derive a discretized solution A' from A^* , where each shift position $t'_i \in S^k$ and the active ranges are shortened to $[0, a'_i) \subseteq [0, a_i)$ as to satisfy the label disjointness property. For every shift position t_i in A^* we define the new shift position t'_i in A' as follows

$$t'_{i} = \begin{cases} \lfloor kt_{i} \rfloor / k & \text{if } t_{i} < 1/2\\ \lceil kt_{i} \rceil / k & \text{if } t_{i} \ge 1/2 \end{cases}$$

In other words, we tilt T_i towards its "heavier" side until it reaches a shift position in the set S^k . Due to the tilting the truncated extrusions are no longer necessarily disjoint and we need to shorten the active ranges for some labels. Figure 4 shows an example.



Figure 4: Discretizing the shift positions of two labels for k = 4.

Let T'_i and T'_j be two tilted truncated extrusions that intersect in their interior. Without loss of generality let T'_i be the smaller one such that, before the tilting, its top right corner was touching the left edge of T'_j as in Figure 4a. We first consider the case that T'_i and T'_j are tilted towards each other. Then the right edge of T'_i , the left edge of T'_j , and the horizontal line $y = a_i$ define a triangle D as in Figures 4b and 5. We decrease the active range of label ℓ_i to $[0, a'_i)$, where $a'_i = a_i - h$ for the height h of D. Obviously the truncated extrusions T'_i and T'_j no longer intersect in their interior for the new active range $[0, a'_i)$.



Figure 5: Intersection triangle D.

Next, we bound the height h of D. Let α be the angle between the right edge of T'_i and the x-axis. The same angle α is found at the top right corner of D. From Figure 4b we obtain that $\tan(\alpha) = a_i/(w_it'_ia_i) = 1/(w_it'_i)$ and from Figure 5 that $\tan(\alpha) = h/x$, where x is distance between the base point of the height h on the top side c and the top right corner. Since T'_i is tilted to the right and T'_j to the left we have $t'_i \geq 1/2$ and $(1 - t'_j) \geq 1/2$. By definition of the new shift positions t'_i and t'_j we know that the length of the top side c of D is at most $(w_ia_i + w_ja_i)/k$. This is because at scale factor s the tilt moves each truncated extrusion with base width w horizontally by at most a 1/k fraction of

its width ws at s. With $x \leq c$ this yields

$$h = \frac{x}{w_i t'_i} \le \frac{c}{w_i/2} \le \frac{2a_i}{k} \frac{w_i + w_j}{w_i}$$

Similar reasoning for the angle β in Figures 4b and 5 yields $\tan(\beta) = a_i/(w_j(1-t'_j)a_i) = h/(c-x)$ and subsequently $h \leq 2a_i/k \cdot (w_i + w_j)/w_j$. Again without loss of generality we assume that $w_i \geq w_j$ and obtain $\min\{(w_i + w_j)/w_i, (w_i + w_j)/w_j\} = (w_i + w_j)/w_i \leq 2$. So we can finally bound the height of D by $h \leq 4a_i/k$.

We still need to consider the case that both truncated extrusions are tilted in the same direction, say to the left (the case that both are tilted to the right is symmetric). A conflict can still occur if T'_j is tilted further to the left than T'_i . The triangle D is defined as before, but now we know that the length of the side c is at most $w_j a_i/k$. Since T'_j is tilted to the left we still have $(1 - t'_j) \ge 1/2$. Now we argue about the angle β using the same identities as before and obtain $h \le c/(w_j(1 - t'_j)) \le 2a_i/k$.

In the case that T'_i and T'_j are tilted away from each other obviously no conflict can occur. Furthermore, the analysis still holds for conflicts involving the top left corner of T'_i instead of the top right corner.

So each truncated extrusion T_i of height a_i is shortened by at most $4a_i/k$ due to the discretization of the shift positions, or, equivalently, $a'_i \ge (1 - 4/k)a_i$. If we set $k = 4/\varepsilon$ we arrive at $\sum_{i=1}^n a'_i \ge (1 - \varepsilon)\sum_{i=1}^n a_i$. \Box

5 Discussion

In this paper we studied an extension of the initial ARO problem, introduced by Been et al. [4], where we additionally allow to slide the labels. Our dynamic programming approach for discrete k-position 1d ARO is a generalization of their approach to solve simple 1d ARO with proportional dilation. It shows that k-position 1d ARO can be solved in polynomial time.

Based on the dynamic program for k-position 1d ARO, we further derived an FPTAS for sliding 1d ARO by suitably discretizing the set of allowed shifts for the labels. While this shows that we can approximate the optimal value arbitrarily closely in polynomial time, the complexity of sliding 1d ARO is still open. The main difficulty in devising an NP-hardness proof is that the problem becomes efficiently solvable when every label has only a polynomial number of relevant sliding positions. It thus seems difficult to encode binary decisions as label positions.

Note that in our model the shift position of each label, once selected, remains fixed for all scales. For the *k*position model this is actually required in order to avoid jumping labels, whereas in a more general sliding model we leave as an open problem to determine a continuous function that defines the label position for every scale. Here we might require that this function is monotone or that its slope is bounded. In practice it is common that some points are more important than others and hence the active ranges of their labels should be more influential in the objective function. More precisely, let $\gamma_i > 0$ be a weight for each point p_i . We can then optimize the weighted total active range length $H_{\gamma} = \sum_{i=1}^{n} \gamma_i a_i$ instead of H. It is easy to see that both the dynamic programming algorithm and the FPTAS remain valid for optimizing H_{γ} .

Another problem variant is to use a non-linear objective function, motivated by the observation that H favors active labels at large values of s, i.e., in maps with small scales 1/s. It might be reasonable in practice to choose a logarithmic function over a linear function for measuring the active ranges. Using the objective function $H_{\log} = \sum_{i=1}^{n} \log a_i$ instead of H has the effect that doubling the scale range at which a label is active has a fixed impact on the objective function regardless of the actual scale. The dynamic programming algorithm can immediately deal with H_{\log} . Even the approximation scheme of Section 4 remains an FPTAS under the mild additional assumptions that the minimum scale factor is 1 (instead of 0), that each $a_i \geq 2$, and that $\varepsilon \leq 1/2$.

Ultimately, the challenge in 2d dynamic map labeling is to consistently support multiple modes of interaction (zooming, panning, rotations) using a slider model for the labels. In this sense, our results are a first step towards consistent dynamic labeling of 2d zoomable maps with sliding labels. Unfortunately, our algorithms do not easily generalize to 2d point sets. In fact, it can be easily seen that k-position 2d ARO and sliding 2d ARO are both NP-hard. The result of Been et al. [4] essentially shows that 1-position 2d ARO, and thus also k-position 2d ARO is NP-hard. For the sliding 2d ARO problem deciding whether all labels may be active at all scales amounts to deciding whether all labels can be placed at scale s_{max} . A slight modification of the NP-hardness proof for map labeling in the four-slider model [16] shows that this problem is NP-hard, even if all labels are unit squares.

Acknowledgments

We thank an anonymous reviewer for helpful suggestions. A. Gemsa and M. Nöllenburg are supported by the Concept for the Future of KIT under project YIG 10-209 within the framework of the German Excellence Initiative and by a Google Research Award.

References

- P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11:209–218, 1998.
- [2] K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3D illustrations. *Journal of the WSCG*, 13(1):1–8, 2005.

- [3] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006.
- [4] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory Appl.*, 43(3):312–328, 2010.
- [5] B. Bell, S. Feiner, and T. Höllerer. View management for virtual and augmented reality. In ACM Sympos. on User Interface Software and Technology (UIST'01), pages 101–110, 2001.
- [6] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In Proc. 7th Annual ACM Sympos. on Computational Geometry (SoCG'91), pages 281–288, 1991.
- [7] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In Proc. 12th Algorithms and Data Structures Symposium (WADS'11). To appear, 2011.
- [8] E. Imhof. Positioning names on maps. The American Cartographer, 2(2):128–144, 1975.
- [9] G. W. Klau and P. Mutzel. Optimal labeling of point features in rectangular labeling models. *Mathematical Programming (Series B)*, pages 435–458, 2003.
- [10] K. D. Mote. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 6(4):249–260, 2007.
- [11] M. Nöllenburg, V. Polishchuk, and M. Sysikaski. Dynamic one-sided boundary labeling. In Proc. 18th ACM SIGSPATIAL Int'l Conf. Advances in Geographic Information Systems, pages 310–319. ACM Press, 2010.
- [12] K. Ooms, W. Kellens, and V. Fack. Dynamic map labeling for users. In Proc. 24th Int'l Cartographic Conference (ICC'09), Santiago, Chile, 2009.
- [13] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling—data-structures and algorithms. In Proc. 23rd Internat. Cartographic Conf. (ICC'03), pages 288–298, Durban, South Africa, 2003.
- [14] S.-H. Poon and C.-S. Shin. Adaptive zooming in point set labeling. In Proc. 15th Internat. Sympos. Fundam. Comput. Theory (FCT'05), volume 3623 of Lecture Notes Comput. Sci., pages 233–244. Springer-Verlag, 2005.
- [15] T. Strijk and M. van Kreveld. Practical extensions of point labeling in the slider model. *GeoInformatica*, 6(2):181–197, 2002.
- [16] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21– 47, 1999.
- [17] F. Wagner and A. Wolff. A practical map labeling algorithm. *Comput. Geom. Theory Appl.*, 7:387–404, 1997.
- [18] F. Wagner, A. Wolff, V. Kapoor, and T. Strijk. Three rules suffice for good label placement. *Algorithmica*, 30(2):334–349, 2001.
- [19] A. Wolff and T. Strijk. The Map-Labeling Bibliography, 1996.

A Discrete and Dynamic Version of Klee's Measure Problem

Hakan Yıldız *

John Hershberger[†]

Subhash Suri^{*}

Abstract

Given a set of axis-aligned boxes $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ and a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ in *d*-space, let the *discrete measure* of \mathcal{B} with respect to \mathcal{P} be defined as $meas(\mathcal{B}, \mathcal{P}) = |\mathcal{P} \cap \{\bigcup_{i=1}^n B_i\}|$, namely, the number of points of \mathcal{P} contained in the union of boxes of \mathcal{B} . This is a discrete and dynamic version of Klee's measure problem, which asks for the *Euclidean* volume of a union of boxes. Our result is a data structure for maintaining $meas(\mathcal{B}, \mathcal{P})$ under dynamic updates to both \mathcal{P} and \mathcal{B} , with $O(\log^d n + m^{1-\frac{1}{d}})$ time for each insert or delete operation in \mathcal{B} , $O(\log^d n + \log m)$ time for each insert and $O(\log m)$ time for each delete operation in \mathcal{P} , and O(1) time for the measure query. Our bound is slightly better than what can be achieved by applying a more general technique of Chan [3], but the primary appeal is that the method is simpler and more direct.

1 Introduction

A classical problem in computational geometry, known as Klee's Measure Problem, asks for an efficient algorithm to compute the volume of the union of n axisaligned boxes in d dimensions. While optimal $O(n \log n)$ time algorithms are known for dimensions one and two [1, 7], the best bound in higher dimensions is roughly $O(n^{d/2})$ [4]. Indeed, despite more than twenty years of effort, the barrier of $O(n^{3/2})$ remains unbroken even in three dimensions. It is known, however, that as the dimension becomes large, the problem is NP– hard [2].

In this paper, we consider a *discrete* and *dynamic* version of Klee's problem, in which the volume of a box is defined as the cardinality of its intersection with a finite point set \mathcal{P} , and both the boxes and the points are subject to insertion and deletion. In particular, we have a set of axis-aligned boxes $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ in *d*-space, and we wish to maintain the *discrete measure* of \mathcal{B} with respect to \mathcal{P} , namely, $meas(\mathcal{B}, \mathcal{P}) = |\mathcal{P} \cap \{\bigcup_{i=1}^n B_i\}|$, under insertion and deletion of both points and boxes.

The problem is fundamental, and arises naturally in several applications dealing with multi-attribute data.

In databases, for instances, data records with d independent attributes are viewed as d-dimensional points, and selection rules are given as ranges over these attributes. A conjunction of ranges over d attributes is then equivalent to a d-dimensional box. Given a set of selection rules, the problem of *counting* all the data records that satisfy the union (namely, the disjunction) of all the rules is our discrete measure problem. Similarly, one may ask for the set of records that fail to satisfy *any of the rules*, and thus form the set of points "not covered" by the union of boxes.

Similarly, the management of firewall rules for network access can also be formulated as a discrete measure problem. The data packets in the Internet are classified by a small number of fields, such as IP address of the source and destination, the network port number, etc. The managers of a local area network (LAN) use a number of "firewall rules" based on these attributes to block some external services (such as ftp) from their network. The discrete measure problem in this setting keeps track of the number of services blocked by all the firewall rules; conversely, one can keep track of the number of services that become "exposed" by the deletion of a box.

Problem Formulation and Our Results

We begin with a formal definition of the problem. A d-dimensional box B is the Cartesian product of d onedimensional ranges, namely $B = \prod_{i=1}^{d} [a_i, b_i]$, where a_i and b_i are reals. The *discrete measure* of a single box Bwith respect to a finite set of points \mathcal{P} is the cardinality of the intersection $\mathcal{P} \cap B$. The discrete measure of the set of boxes \mathcal{B} with respect to \mathcal{P} , denoted $meas(\mathcal{B}, \mathcal{P})$, is the cardinality of $\mathcal{P} \cap \{\bigcup_{B \in \mathcal{B}} B\}$. (Because a point may lie in multiple boxes, the discrete measure of \mathcal{B} is not the sum of the measures of the individual boxes.) In this paper, we consider the problem of maintaining the discrete measure under insertion and deletion of both points and boxes. Specifically, we propose a data structure that supports modifying \mathcal{P} through insertion or deletion of a point, modifying \mathcal{B} through insertion or deletion of a box, and querying for the current discrete measure $meas(\mathcal{B}, \mathcal{P})$.

Despite its natural formulation, the problem appears not to have been studied in this form. This may be partially attributed to the fact that the *static* version of the problem is easy to solve using standard

^{*}Department of Computer Science, University of California, Santa Barbara, {hakan,suri}@cs.ucsb.edu

[†]Mentor Graphics Corp., john_hershberger@mentor.com

data structures of computational geometry: build a *d*dimensional version of a segment tree for the set of boxes, and then query separately for each point to determine whether any box contains it, for a total of $O(n \log^d n + m \log^{d-1} n)$ time. This approach, however, is inefficient when the set of boxes is dynamic, because each insertion or deletion can affect a large number of points, requiring $\Omega(m)$ recomputation time per update.

During the writing of this paper, we discovered that a technique of Chan [3] can be used to solve this problem. In [3], he describes a data structure for maintaining a set of points and a set of hyperplanes in *d*-space to answer queries of the form "does any of the points lie below the lower envelope of the hyperplanes." One can use this data structure in combination with standard range searching structures and a dynamization technique by Overmars and van Leeuwen [9] to solve our discrete measure problem so that point insertions and box updates require $O(\log^2 m + \log^d n)$ and $O(m^{1-\frac{1}{d}} \log m + \log^d n)$ time respectively.¹

Compared to this bound, the time complexity of our data structure is better by a factor of log m. However, a more important contribution may be the simplicity of our method and the fact that it solves the problem in a more *direct* way, making it more appealing for implementation. Specifically, our result gives a dynamic data structure for the discrete measure problem with the following performance: a box can be inserted or deleted in time $O(m^{1-\frac{1}{d}} + \log^d n)$; a point can be inserted in time $O(\log m + \log^d n)$ and deleted in time $O(\log m)$. The data structure always updates its measure, so a query takes O(1) time.

The data structure also solves the *reporting* problem in output-sensitive time. Specifically, if k is the number of points in the union of the boxes, then they can be found in $O(k + k \log \frac{m}{k})$ worst-case time. The same bound also holds if one wants to report the points *not* contained in the union. Finally, we extend our results to a *stochastic* version of the problem, in which each point and each box is associated with an independent probability of being present. In this case, one can naturally define an *expected discrete measure*, which is the expected number of points present that are covered by the union of the boxes present. Our bounds for the stochastic case are the same as the deterministic one.

2 Maintaining the Discrete Measure

In the following discussion we assume that all the boxes in \mathcal{B} and points in \mathcal{P} have distinct coordinates.² Before we describe our dynamic data structure, it is helpful to consider a solution for the static problem. Let \mathcal{B} be a set of n boxes and \mathcal{P} a set of m points in d-space. For each point $p \in \mathcal{P}$, we define its *stab-bing count*, denoted stab(p), as the number of boxes in \mathcal{B} that contain p. The measure of a single point p, $meas(\mathcal{B}, \{p\})$, is 1 if stab(p) > 0 and 0 otherwise. One can easily see that the overall discrete measure can be written as the sum of point measures; that is, $meas(\mathcal{B}, \mathcal{P}) = \sum_{p \in \mathcal{P}} meas(\mathcal{B}, \{p\})$. The stabbing count of a point can be efficiently obtained using a multi-level segment tree [10], which achieves the following performance bounds.

Lemma 1 ([10]) The multi-level segment tree represents a set of n boxes in d-space. The structure can report the stabbing count of any query point in $O(\log^{d-1} n)$ time. It requires $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ preprocessing time for construction.

By building a multi-level segment tree and then querying it for the stabbing count of each point in \mathcal{P} , we can calculate the measure $meas(\mathcal{B}, \mathcal{P})$ for the static problem in $O(n \log^d n + m \log^{d-1} n)$ time using $O(n \log^{d-1} n)$ space.

2.1 Invariants for Stabbing and Measure

The static solution described above loses its appeal in the dynamic setting because each box insertion or deletion can invalidate the stabbing count of $\Omega(m)$ points. We circumvent this problem by storing the stabbing counts *indirectly*, using an idea from *anonymous segment trees* [12], so that only a small number of these indirect values need to be modified after a box update. We describe the technique in general first, deferring its specialization for the efficient maintenance of the discrete measure until later.

Consider a balanced tree (not necessarily binary) whose leaves are in one-to-one correspondence with the points of \mathcal{P} . The point corresponding to a leaf v is denoted p_v . In order to represent the stabbing counts of the points, we store a *non-negative* integer field $\sigma(w)$ at each node w of the tree subject to the following sum invariant: for each leaf v, the sum of $\sigma(a)$ over all ancestors a of v (including v itself) equals $stab(p_v)$. By assigning $\sigma(v) = stab(p_v)$ to each leaf v and $\sigma(w) = 0$ to all internal nodes w, we may obtain a trivial assignment with the sum invariant. But, as we will see, the flexibility afforded by these σ values allows us to update the stabbing counts of many points by modifying only a few σ values. As an example, if a box covering all the points of \mathcal{P} were inserted, then incrementing the single value $\sigma(root)$ by 1 suffices, where root denotes the root of the tree.

We will maintain the discrete measure, $meas(\mathcal{B}, \mathcal{P})$, through the σ values. In particular, at each node v, we

 $^{^{1}}$ Reducing box update time is possible at the expense of increasing the cost of point insertions and vice versa.

 $^{^{2}}$ This assumption merely simplifies the presentation; one can use symbolic perturbation to break ties between identical coordinates without affecting the result.



Figure 1: The push-up operation on a node with two children.

store a quantity $\bar{\mu}(v)$ representing the number of points that have a stabbing count of 0, considering only the information stored in the subtree rooted at v. (The notation $\bar{\mu}$ is meant to suggest that it represents the complement of the measure.) The quantity $\bar{\mu}(v)$ is defined recursively using the σ values as follows:

$$\bar{\mu}(v) = \begin{cases} 0 & \text{if } \sigma(v) > 0\\ 1 & \text{if } \sigma(v) = 0 \ \land \ v \text{ is a leaf}\\ \sum_{w \in child(v)} \bar{\mu}(w) & \text{if } \sigma(v) = 0 \ \land \\ & v \text{ is an internal node} \end{cases}$$

where child(v) represents the children of a non-leaf node v. It is easy to show that $\bar{\mu}(root)$ is the number of points in \mathcal{P} whose stabbing counts are 0. Consequently, $meas(\mathcal{B}, \mathcal{P}) = m - \bar{\mu}(root)$, and one can report $meas(\mathcal{B}, \mathcal{P})$ in O(1) time.

We add one final constraint on σ values to achieve uniqueness, which also contributes to the efficiency of our specialized structure. In particular, we push the σ values as high up the tree as possible to enforce the following *push-up invariant*: at least one child of every non-leaf node v has a σ value of 0. This specifies σ uniquely, as shown by the following lemma.

Lemma 2 Let T be a tree representing a set of points \mathcal{P} and their stabbing counts as described above. Then there exists a unique configuration of σ values satisfying the sum and the push-up invariants in T.

Proof. We prove only the existence of the desired configuration due to space limitation; the proof of uniqueness can be found in the full version of the paper. Consider an arbitrary configuration of σ values satisfying the sum invariant. (For instance, $\sigma(v) = stab(p_v)$ for each leaf and $\sigma(v) = 0$ for each non-leaf.) We then apply the following *push-up* operation at each non-leaf node v to revise its value: increment $\sigma(v)$ by Δ and decrement $\sigma(w)$ by Δ for each child w of v, where $\Delta = \min_{w \in child(v)} \sigma(w)$. (See Figure 1). This achieves the push-up invariant at v while preserving the sum invariant in the tree. Repeated applications of the push-up operation from the leaves to the root produce a configuration of σ values satisfying both invariants.



Figure 2: A measure tree of 9 points on the plane.

2.2 The Measure Tree and Dynamic Updates

In order to allow efficient insertion and deletion of boxes, and the corresponding updates of the points' stabbing counts, we organize \mathcal{P} in a balanced tree that supports efficient range queries. A k-d tree, where points are stored at the leaves, allows efficient range queries, but is inefficient for insertion and deletion of *points.*³ The structure we propose, which we call a *measure tree*, is a variant of divided k-d trees [11], and allows both efficient range queries and updates on the set of points. We note that the tree described in this section has slightly slower amortized bounds but these can be easily improved to achieve our main result as explained in Section 2.5.

We describe the measure tree in two dimensions for simplicity; the extension to d dimensions is conceptually straightforward, but we defer those details for later. Given a dynamic set of points \mathcal{P} in the plane, we represent \mathcal{P} as a two-level tree. The first level consists of an upper tree that partitions the points of \mathcal{P} into at most $2\sqrt{m}$ subsets along the x-axis, each containing at most $2\sqrt{m}$ points, where m is the current size of \mathcal{P} . Each leaf of the upper tree acts as a root for a lower level tree that further partitions the corresponding subset of points using their *y*-coordinates. These lower trees form the second level of our tree. Figure 2 shows an example. Both levels of the tree are organized using 2-3 trees in which each data element is stored in a single leaf. Consequently, each leaf of the measure tree corresponds to a single point of \mathcal{P} and we can use our measure maintenance scheme to store σ and $\bar{\mu}$ values on the nodes. We now discuss how to perform updates on the measure tree while preserving the invariants.

Insertion or Deletion of a box B. Let us consider insertion first. We find a set C of subtrees whose leaves correspond to the points covered by B. This is a range query, where we first perform a one-dimensional range search on the upper tree to locate the subsets of points that are completely or partially covered by the x-range of B. Observe that at most two subsets are partially covered. We then search the lower level trees corresponding to the partially covered subsets to find the points contained in B. The leaves corresponding to these points are included in C. For each subset that is completely covered by the x-range of B, we perform

 $^{^{3}}$ There is also no easy way to implement our scheme using range trees because they contain multiple copies of the points.

a one-dimensional range search on the corresponding lower tree to find a set of maximal subtrees containing the points that lie in B. These maximal subtrees are also included in C. It is straightforward to show that the subtrees in C span the set of points covered by Band the total cost of the range query is $O(\sqrt{m} \log m)$.

The insertion of B causes the stabbing count of each point contained in B to increase by 1. We effect this by incrementing the σ value of the root of each subtree in \mathcal{C} by 1. This corrects the sum invariant in the tree, but may invalidate the push-up invariant. We therefore apply push-ups on the nodes whose σ values are updated. Since each push-up may introduce a violation of the push-up invariant at the parent, we continue applying push-ups until all violations are resolved. Finally, we recompute $\bar{\mu}$ for all ancestors of nodes whose σ values changed. This recomputation is also done bottom-up, since the $\bar{\mu}$ value of a node depends on the $\bar{\mu}$ values of its descendants. We note that both the push-ups and the recomputations of $\bar{\mu}$ values can be done as part of the tree traversal of the range query. It follows that the total cost of the box insertion is $O(\sqrt{m}\log m)$ time.

The handling of deletion is similar to insertion, except that we decrement the σ value of the root of each subtree found by the range query. The time complexity is $O(\sqrt{m} \log m)$, as for insertion. Decrementing the σ 's may cause some values to drop below zero, but the push-up operations eliminate these negative values. In particular, observe that a push-up at a node v restores not only the push-up invariant but also the non-negativity of v's children. To see that the final value of $\sigma(root)$ is non-negative, imagine a root-to-leaf path (as in the proof of uniqueness for Lemma 2 found in the appendix) such that σ is zero for all nodes on the path except *root*. The path ends at a leaf v such that $stab(p_v)$ equals $\sigma(root)$, and so it follows that $\sigma(root)$ is non-negative.

Insertion or Deletion of a point *p***.** When inserting a point p, we search the upper tree with the x-coordinate of p to find the lower tree in which p should be inserted, and then insert p using the standard 2-3 tree insertion algorithm. This creates a new leaf v with $p_v = p$. We need to know the stabbing count of p in order to initialize $\sigma(v)$ correctly. For the moment, let us assume that we know stab(p)—see Lemma 3—and focus on the update of the tree. In order to preserve the sum invariant, we set $\sigma(v)$ to $stab(p) - \Sigma$, where Σ is the sum of $\sigma(a)$ over all strict ancestors a of v. If $\sigma(v)$ is less than 0, we apply push-ups to v and all of its ancestors to push the negativity to the root, where it is canceled out. The 2-3 tree insertion may split one or more ancestors of v, and during those splits, the σ values of the resulting nodes are set to the original node's σ value, thereby preserving the sum invariant. After the split, we apply push-ups on the resulting nodes to re-establish the push-up in-



Figure 3: Push-down in a merge.

variant. Altogether, $O(\log m)$ splits and push-ups are performed, and so the cost of the insertion is $O(\log m)$. The insertion might cause the size of the lower tree to exceed $2\sqrt{m}$, but this is discussed in Section 2.3.

When a point p is deleted, we locate the lower tree containing it and simply delete the leaf corresponding to p, and restructure the tree to reestablish the balance of the 2-3 tree. The sum invariant is unaffected by the deletion, but we may need to apply push-ups to the ancestors of v to restore the push-up invariant. The deletion may also cause 2-3 tree merge or redistribution operations, and to preserve the sum invariant during these operations, we push the σ values of the participating nodes down to their children (see Figure 3). After the operation, push-ups are applied on these nodes to restore the push-up invariant. If the lower tree becomes empty as a result of the deletion, we simply delete it and apply the same deletion algorithm on the upper tree. Due to the decrease in the value of m, the sizes of some upper or lower trees may exceed $2\sqrt{m}$; we deal with this in Section 2.3.

2.3 Complexity Analysis

We use two types of operations to ensure that the upper and the lower trees do not exceed their size thresholds. First, when a lower tree's size exceeds $2\sqrt{m}$, we split it into two new lower trees of equal size, destroying the original tree and constructing the new trees from scratch. During this process, we traverse the original tree to obtain the stabbing counts of the points and use those to construct the new trees. This split operation takes $O(\sqrt{m})$ time since the y-order of the points is known. Second, we avoid violating the upper tree's threshold by periodically rebuilding the entire measure tree. This reconstruction creates a lower tree for each set of $\lceil \sqrt{m} \rceil$ points along the x-axis (except perhaps the last one in the sequence, which may be smaller). Consequently, the size of the upper tree is at most \sqrt{m} . The reconstruction takes $O(m \log m)$ time. (It can be done in O(m) time if we maintain the x- and y-orders of the points separately.) We determine when to do the reconstruction as follows. Assume that the most recent reconstruction of the tree was done when $m = m_0$. We reconstruct the tree after $\frac{1}{5}m_0$ point insertions or deletions. This ensures that the upper tree does not exceed its threshold. The proof is straightforward and left to the reader.

Next, we discuss how to initialize the stabbing count of a point when it is first inserted. We enable this by maintaining a separate *dynamic multi-level segment tree* [5], which provides the following functions dynamically.

Lemma 3 ([5]) The dynamic multi-level segment tree represents a dynamic set of n boxes in d-space. The structure uses $O(n \log^{d-1} n)$ space and can report the stabbing count of any query point in $O(\log^d n)$ time. It supports insertion or deletion of boxes in $O(\log^d n)$ time apiece.

Putting together these pieces, we obtain our main result in two dimensions.

Theorem 4 We can maintain the discrete measure in two dimensions using $O(n \log n + m)$ space, with constant time measure queries, $O(\log^2 n + \sqrt{m} \log m)$ time for insertion or deletion of a box, $O(\log^2 n + \log m)$ time for a point insertion, and $O(\log m)$ time for a point deletion time. (The log m term in the bounds is amortized.)

Proof. We use the measure tree along with a twodimensional dynamic segment tree. The bound on the space complexity follows because the measure tree requires linear space and the multi-level segment tree requires $O(n \log n)$ space by Lemma 3. The query complexity is obviously constant. The insertion or deletion of a box takes $O(\sqrt{m}\log m)$ time for the measure tree and $O(\log^2 n)$ time for the segment tree. The cost of inserting or deleting a point is $O(\log m)$ for the measure tree if there is no reconstruction of a lower tree or the whole measure tree. Reconstruction of the measure tree costs $O(m \log m)$. We charge the cost of this construction to the $\Omega(m)$ point updates that must precede it, which gives us an amortized cost of $O(\log m)$ per update. The reconstruction of a lower tree costs $O(\sqrt{m})$. One can easily show that $\Omega(\sqrt{m})$ point insertions precede the construction, which gives us an amortized cost of O(1). Finally, we do a stabbing count query costing $O(\log^2 n)$ time when we insert a point. This completes the proof.

2.4 Extension to Higher Dimensions

The measure tree naturally extends to higher dimensions, as a *d*-level tree, with each level partitioning the points along one of the coordinate axes. The tree at the top level partitions the set of points into at most $2m^{1/d}$ subsets, each of which is partitioned into at most $2m^{1/d}$ subsets by a second level tree. This partitioning continues through *d* levels. The measure is maintained using the σ and $\bar{\mu}$ values, as in two dimensions.

All the update procedures are natural extensions of their two-dimensional counterparts. The initial tree size is at most $\lceil m^{1/d} \rceil$ at all levels; a tree is split when its size becomes larger than $2m^{1/d}$. Moreover, one can show that there is a positive constant C such that reconstructing the tree after each Cm_0 point insertions or deletions guarantees that the size of the upper tree is bounded by $2m^{1/d}$. The following theorem summarizes the bounds of the *d*-dimensional structure; its proof is similar to that of Theorem 4.

Theorem 5 We can maintain the discrete measure in d dimensions, for $d \ge 2$, using $O(n \log^{d-1} n + m)$ space, with constant time measure queries, $O(\log^d n + m^{1-\frac{1}{d}} \log m)$ time for insertion or deletion of a box, $O(\log^d n + \log m)$ time for insertion of a point, and $O(\log m)$ time for the deletion of a point. (The log m term in the bounds is amortized.)

2.5 Further Improvements

The amortized bounds of our structure can be converted to worst case bounds, using a technique called *global rebuilding* [8]. The idea, in brief, is to spread out the process of subtree reconstruction over time, operating on a shadow copy of the main data structure and then swapping in the result when the reconstruction is finished.

Finally, the term $m^{1-\frac{1}{d}} \log m$ in box update bounds can be improved to $m^{1-\frac{1}{d}}$ by using an optimized version of the measure tree. For instance, in two dimensions, the partitioning parameter can be tuned to achieve $O(\sqrt{m} + \log^2 n)$ time for inserting or deleting a box, by mimicking the construction of [6].

3 Extensions

3.1 Reporting Queries

In some applications, it is useful to report explicitly the points covered (or uncovered) by the union of boxes. Our structure can be used to answer such queries in output-sensitive time as in the following theorem.

Theorem 6 A reporting query can be answered in $O(k + k \log \frac{m}{k})$ time, where k is the size of the output.

Proof. The proof will appear in the full version of the paper. $\hfill \Box$

3.2 Stochastic Discrete Measure

The recent proliferation of data mining applications has created an urgent need to deal with *data uncertainty*, which may arise because the mining algorithms output probability distributions over an output space, or because attributes whose values are not explicitly known are modeled with a discrete set of probabilistic values. This motivates a natural *stochastic* extension of our discrete measure problem, in which both the underlying set of points \mathcal{P} and the set of boxes \mathcal{B} are associated with independent probabilities. Specifically, each point p in \mathcal{P} occurs with probability π_p and each box B in \mathcal{B} occurs with probability π_B . The probabilities are independent, but otherwise can take any real values. A natural problem in this setting is to compute the *expected size* of the discrete measure—that is, how large is $meas(\mathcal{B}, \mathcal{P})$ for a random sample of boxes and points drawn from the given probability distribution?

Our structure can be easily adapted to this stochastic problem with the same complexity bounds. The details will appear in a journal version of the paper.

Theorem 7 The d-dimensional stochastic measure problem can be solved with a data structure that requires $O(n \log^{d-1} n + m)$ space, O(1) query time, $O(\log^d n + m^{1-\frac{1}{d}})$ time for insertion or deletion of a box, $O(\log^d n + \log m)$ time for a point insertion and $O(\log m)$ time for a point deletion.

4 Closing Remarks

We introduced a discrete measure problem, and presented a data structure that supports dynamic updates to both the set of points and the set of boxes. The queries for the current measure take constant time, the updates to the set of points take polylogarithmic time, while updates to the set of boxes take time polylogarithmic in the number of boxes and sub-linear in the number of points. The data structure permits outputsensitive enumeration of the points covered by the union of the boxes, and also lends itself to a stochastic setting in which points and boxes are present with independent, but arbitrary, probabilities.

Our work leads to a number of research problems. First, can the update bounds be improved? Second, is there a trade-off between the update time for boxes and the update time for points? In particular, can one achieve polylogarithmic complexity in both n and m?

References

- J. L. Bentley. Solutions to Klee's rectangle problems. Unpublished manuscript, Dept. of Comp. Sci., CMU, Pittsburgh PA, 1977.
- [2] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. CGTA, 43:601–610, 2010.
- [3] T. Chan. Semi-online maintenance of geometric optima and measures. In Proc. 13th annual ACM-SIAM Symposium on Discrete algorithms, pages 474–483, 2002.
- [4] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. CGTA, 43(3):243–250, 2010.

- [5] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. *Report F59, Institut f\u00fcr In*form., TU Graz, 1980.
- [6] K. Kanth and A. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proc. ICDT*, page 257. Springer, 1998.
- [7] V. Klee. Can the measure of $\cup [a_i, b_i]$ be computed in less than $O(n \lg n)$ steps? American Mathematical Monthly, pages 284–285, 1977.
- [8] M. Overmars. The design of dynamic data structures. Springer, 1983.
- [9] M. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981.
- [10] V. K. Vaishnavi. Computing point enclosures. *IEEE Trans. Comput.*, 31:22–29, January 1982.
- [11] M. van Kreveld and M. Overmars. Divided k-d trees. Algorithmica, 6(1):840–858, 1991.
- [12] H. Yıldız, L. Foschini, J. Hershberger, and S. Suri. The union of probabilistic boxes: Maintaining the volume. In Proc. 19th Annual European Symposium on Algorithms (ESA), 2011.
Kinetically-aware Conformational Distances in Molecular Dynamics

Chen Gu^*

Xiaoye Jiang[†]

 $iang^{\dagger}$

Leonidas Guibas[‡]

Abstract

In this paper, we present a novel approach for discovering kinetically metastable states of biomolecular conformations. Several kinetically-aware metrics which encode both geometric and kinetic information about biomolecules are proposed. We embed the new metrics into k-center clustering and r-cover clustering algorithms to estimate the metastable states. Those clustering algorithms using kinetically-aware metrics are tested on a large scale biomolecule conformation dataset. It turns out that our algorithms are able to identify the kinetic meaningful clusters.

1 Introduction

Conformational changes are of fundamental importance in a wide range of biological processes including protein folding [4], RNA folding [1] and the operation of key cellular machinery [7]. Extensive genetic, biochemical, biophysical and structural experiments can help to understand these conformational changes. However, probing the mechanisms of conformational changes at atomic resolution is very difficult experimentally and without these details it is impossible to understand the crucial chemistry they perform. On the other hand, computer simulations may complement such experiments by providing dynamic information at an atomic level. With powerful individual processors, or large distributed clusters of processors, one can routinely generate large quantities of simulation data for a given phenomenon of interest. As a result, a growing challenge is how to mine such massive data sets so as to gain insight into the interesting biochemical phenomena under study.

To meet such a challenge, a lot of recent effort has been devoted to constructing stochastic kinetic models, often in the form of discrete-state Markov models, from relatively short molecular dynamics simulations [2]. In order to construct useful mathematical models that can faithfully represent the molecular dynamics at the timescales of interest, it is often necessary to decompose the conformational space into a set of kinetically *metastable states*, or clusters.

In this paper, we present a new method for the discovery of kinetically metastable states that are gen-



Figure 1: Two conformations which are geometrically close but kinetically far away. Red dots and blue lines denote atoms and bonds respectively.

erally applicable to solvated macromolecules. Given molecular dynamics trajectories consisting of thousands of molecular conformations, our algorithm can identify the long lived, kinetically metastable states by clustering with respect to the *kinetically-aware conformational distances*. Such distance functions encode both the geometry and kinetic information about molecular conformations, which allow robust partitioning of the conformational space into kinetically related regions.

2 Conformational distance measures

2.1 cRMS distance

In bioinformatics, a commonly used metric for estimating the distance between two molecules is the *coordinate root mean squared (cRMS) distance*. Such a distance can be evaluated as the root mean squared deviation (RMSD) distance¹ of the Cartesian coordinates of heavy atoms in the molecules, optimally aligned by a rigid body translation and rotation minimizing the RMSD [6]. The cRMS distance is a popular choice for biological computation because it possesses all the qualities of a proper distance metric [9], which takes account of both local similarities between molecule conformations and global ones. Moreover, the complexity of estimating the cRMS distance is proportional to the number of atoms, which makes it possible to compute distances between large molecules quickly.

However, a key disadvantage of the cRMS distance is that it ignores the kinetic deformation change from one conformation to another. As illustrated in Figure 1, each of the two conformations has two folded arms, yet the orders that the arms overlap are different. Thus, the two conformations are close geometrically, while they indeed differ greatly kinetically because the deformation change from one to the other has to follow a long trajectory without self-collision in the conformational space. Therefore, it would be more appropriate if we can incorporate such kinetic information from trajectories into the distance functions.

The RMSD distance between two vectors $x = (x_1, \dots, x_N)^T$, $y = (y_1, \dots, y_N)^T$ is $\sqrt{\sum_{i=1}^N (x_i - y_i)^2/N} = ||x - y||_2/\sqrt{N}$.

^{*}Institute for Computational and Mathematical Engineering, Stanford University, guc@stanford.edu

[†]Institute for Computational and Mathematical Engineering, Stanford University, xiaoyej@stanford.edu

 $^{^{\}ddagger} \text{Department}$ of Computer Science, Stanford University, <code>guibas@cs.stanford.edu</code>



Figure 2: An illustration of the delayed coordinates distance.

2.2 Kinetically-aware conformational distances

In this section, we propose two different kineticallyaware conformational distance functions. The first distance function is defined using the delayed coordinates of conformations which incorporate information of conformational changes at nearby timesteps. The second distance function is given by the shortest path graph distance, while such a graph is constructed based on trajectory dynamics.

2.2.1 Delayed coordinates distance

To define the *delayed coordinates distance* between two conformations, we examine their path context – a set of conformations surrounding them in the trajectory where they come from. Here we assume that the sampling is at the same rate along all trajectories, and each conformation belongs to a unique trajectory in simulation. When we compare two conformations A_i and B_j , we take 2h+1 samples around each conformation on their paths: $\{A_{i-h}, \ldots, A_i, \ldots, A_{i+h}\}$ and $\{B_{j-h}, \ldots, B_j, \ldots, B_{j+h}\}$ (*h* is a pre-given sample window size), and define the distance between A_i and B_j as a weighted average of the cRMS distances between the corresponding samples:

$$D(A_i, B_j) = \sum_{\ell=-h}^{h} w_\ell d(A_{i+\ell}, B_{j+\ell})$$
(1)

In (1), $d(A_{i+\ell}, B_{j+\ell})$ is the cRMS distance between $A_{i+\ell}$ and $B_{j+\ell}$, and all weights w_{ℓ} 's are non-negative. It is easy to verify that the distances defined in (1) satisfy the triangle inequality:

$$D(A_{i}, B_{j}) + D(B_{j}, C_{k})$$

$$= \sum_{\ell=-h}^{h} w_{\ell} d(A_{i+\ell}, B_{j+\ell}) + \sum_{\ell=-h}^{h} w_{\ell} d(B_{j+\ell}, C_{k+\ell})$$

$$= \sum_{\ell=-h}^{h} w_{\ell} \Big(d(A_{i+\ell}, B_{j+\ell}) + d(B_{j+\ell}, C_{k+\ell}) \Big)$$

$$\geq \sum_{\ell=-h}^{h} w_{\ell} d(A_{i+\ell}, C_{k+\ell}) = D(A_{i}, C_{k}).$$

Therefore, the above defined delayed coordinates distance is a valid metric.

As depicted in Figure 2, A_i and B_j are geometrically very close in the conformational space, but they occur on paths that pass through in very different ways. By considering their path neighbors, we can better characterize their distance because the nearby conformations can help address the kinetic difference between them.

Notice that in (1), we need to compute the best alignment for each conformation pair in the sample win-

dow. Alternatively, we can optimize one alignment jointly for all conformation pairs. Without loss of generality, we assume all conformations are centered at the origin (after optimal translation). We map A_i to $A'_i = [w_{-h}A_{i-h}, \ldots, w_0A_i, \ldots, w_hA_{i+h}]^T$ and B_j to $B'_j = [w_{-h}B_{j-h}, \ldots, w_0B_j, \ldots, w_hB_{j+h}]^T$, and define $D(A_i, B_j)$ as the cRMS distance between A'_i and B'_j , which is also a valid metric. The optimal alignment (rotation) f between A'_i and B'_j will minimize the following objective function:

$$D^{2}(A_{i}, B_{j}) = d^{2}(A'_{i}, B'_{j}) = \|f(A'_{i}) - B'_{j}\|_{2}^{2}$$

$$= \sum_{\ell=-h}^{h} \|f(w_{\ell}A_{i+\ell}) - w_{\ell}B_{j+\ell}\|_{2}^{2}$$

$$= \sum_{\ell=-h}^{h} \|w_{\ell}f(A_{i+\ell}) - w_{\ell}B_{j+\ell}\|_{2}^{2}$$

$$= \sum_{\ell=-h}^{h} w_{\ell}^{2} \|f(A_{i+\ell}) - B_{j+\ell}\|_{2}^{2}$$
(2)

(ignoring the constant sacling factor 1/N in RMSD definition). So, f gives the best alignment jointly for all conformation pairs in the sample window.

2.2.2 Shortest path graph distance

Given a large number of relatively short conformational trajectories, we can adapt the cRMS distance to reflect the fact that successive conformations along a trajectory should in some sense be closer to each other than their cRMS distance represents, capturing the affinity between the conformations implied by the physical process generating the trajectory.

Since ultimately we deal with a discrete set of conformations, we can consider a large graph of all the conformations along the generated trajectories as nodes and add edges between all pairs with weights given by their corresponding cRMS. To incorporate kinetic information into our distance function, for conformation pairs that are neighbors along a trajectory, we reduce their cRMS distance by multiplying a certain factor 0 < c < 1, so that conformations along a trajectory are closer to each other than their static distances.

However, after discounting the cRMS distance for certain edges that correspond to conformations along the same trajectory, these new weights may violate the triangle inequality. To retain the metric property, we define the new distances as the lengths of shortest paths in this graph connecting the two conformations in question, which clearly define a metric.

The factor c controls the tradeoff between the static cRMS distances and the kinetic information from trajectories. When c = 1, the distance function is purely static; On the other hand, when $c \to 0$, all conformations along a trajectory are arbitrarily close to each



Figure 3: Relation between two distance functions.

other. As a result, each trajectory becomes a cluster itself. Thus, by varying the factor c, we can control the relative amount of geometry and kinetic information that are used in the new distance function.

2.2.3 Relation between two distance functions

Intuitively, these two distance functions represent two different ways to incorporate kinetic information from trajectoties: either penalize conformation pairs from different trajectories, or maintain conformation pairs along a same trajectory close to one another. In fact, both of them can be viewed as graph distances (see Figure 3). In the delayed coordinates distance, we consider a set of 2h+1 paths from A_i to B_j : $\{A_i \to \ldots \to A_{i+\ell} \to B_{j+\ell} \to \ldots \to B_j\}|_{\ell=-h}^h$. Notice that $d(A_{i+\ell}, B_{j+\ell})$ can be seen as the length of the path $\{A_i \to \ldots \to A_{i+\ell} \to B_{j+\ell} \to \ldots \to B_j\}$ with a discount factor c = 0, so the delayed coordinates distance $D_{w,h}^{(1)}(A_i, B_j)$ is equal to the weighted average of these 2h+1 path lengths. In contrast, the shortest path graph distance $D_c^{(2)}(A_i, B_j)$ is defined as the minimum path length among all possible paths from A_i to B_j in the complete graph. There-

fore,
$$D_{w,h}^{(1)}(A_i, B_j) \ge \sum_{\ell=-h}^{h} w_\ell \cdot D_{c=0}^{(2)}(A_i, B_j)$$
.

3 Clustering massive data sets

3.1 k-center clustering

Due to the heterogeneous nature of many biological processes at the molecular scale, we usually need a large quantity of simulation data to mine in order to gain insight into the fundamental biochemical phenomena under study. To reach an understanding into the data scientifically, one often needs to shrink the data sets by applying a clustering algorithm to yield a family of clusters (metastable states) of much smaller size than the original data set. Since it is common for simulations conducted on supercompters to generate data sets that contain 10^5-10^7 conformations in up to 10^4 trajectories, we would prefer a clustering algorithm with computational complexity linear in the number of conformations. In non-Euclidean space, a good candidate for clustering such massive data sets is the *k*-center clustering.

The k-center problem originates from the facility location problem, whose goal is to open k facilities centers among n points such that every point is near some facility center. The problem is formulated as follows:

k-center problem: Given n demand points \mathcal{D} in a metric space, find k supply points $\mathcal{S} \subseteq \mathcal{D}$, such that the maximum distance between a demand point $p \in \mathcal{D}$ and its nearest supply point $q \in \mathcal{S}$ is minimized.

In the k-center problem, the goal is to find the optimal value $r = \min_{\mathcal{S}} \max_{p \in \mathcal{D}} \min_{q \in \mathcal{S}} |p - q|$, and to specify which points should be chosen as centers to satisfy the constraints with that value of r. Notice that if we draw kballs centered at these supply points with radius r, they will cover all n demand points (see Figure 4). Therefore,



Figure 4: *k*-center problem (a) and its equivalent formulation (b).

the k-center problem can be equivalently formulated as follows: Given n points \mathcal{D} in a metric space, find k balls of smallest radius centered at $\mathcal{S} \subseteq \mathcal{D}$ which altogether cover every point in \mathcal{D} .

A basic fact about the k-center problem is that it is NP-hard. Thus there is no efficient algorithm that always returns the optimal solution. However, there is a simple greedy algorithm called *farthest-first traversal* [3] that works fairly well in practice. The algorithm iteratively picks a new center farthest from the ones chosen so far, and it returns a 2-approximation solution for the k-center problem. In fact, it is not possible to achieve a better approximation ratio for arbitrary metric spaces: even getting a factor $2 - \epsilon$ for any $\epsilon > 0$ is NP-hard [8].

Assuming we can fetch the distance between two points in $\mathcal{O}(1)$ time, farthest-first traversal takes $\mathcal{O}(kn)$ running time and $\mathcal{O}(n)$ working space. So, this algorithm is good for clustering using delayed coordinates distance. However, in the case of shortest path graph distance, we cannot get the pairwise distance from the graph in constant time. As a result, the running time grows to $\mathcal{O}(kn^2)$ since we need to run Dijkstra's algorithm to update the distances from every point to its nearest center in each iteration. In scenarios when k is also large (e.g., clustering all conformations into hundreds of microstates), farthest-first traversal becomes too slow. In the next section, we propose a new clustering algorithm using shortest path distances by considering a related variant problem of the k-center problem, namely, the problem of computing covering numbers.

3.2 r-cover clustering

When we use the k-center clustering, a natural question is how many clusters should we choose (especially for the case when k is large)? As we have seen before, when we cluster data, we implicitly compute the radius r. If we choose a large number for k, then r should be small. In contrast, when k is small, the returned number r should be large. Therefore, it is equivalent to ask how large is the radius we want for clustering? From this observation, we transfer the original k-center problem into a variant problem of computing covering numbers, by swapping the input k and the output r.

r-covering number: Given *n* demand points \mathcal{D} in a metric space, an *r*-cover of \mathcal{D} is a set of supply points $\mathcal{S} \subseteq \mathcal{D}$ such that every demand point $p \in \mathcal{D}$ is at most distance *r* away from its nearest supply point $q \in \mathcal{S}$. The *r*-covering number of \mathcal{D} is the size of its smallest

r-cover, i.e., $\mathcal{N}(\mathcal{D}, r) = \min_{\mathcal{S}} \{ |\mathcal{S}| : \max_{p \in \mathcal{D}} \min_{q \in \mathcal{S}} |p - q| \le r \}.$

In the farthest-first traversal algorithm, we repeatedly choose a new center that is farthest from all previous centers, which costs $\mathcal{O}(n^2)$ per iteration for shortest path distances. The main problem here is that we spend a lot of time to compute the real shortest path distances between nodes that are very far from each other. However, by transforming the k-center problem into the rcover model, it is possible for us to combine Dijkstra's shortest path algorithm and clustering together (see Algorithm 1).

In this r-cover clustering algorithm, we randomly choose an uncovered node as a new center, and run Dijkstra's algorithm to cover all nodes that are at most raway from this new center. Recall that Dijkstra's algorithm finds the real shortest path distances for all nodes in an increasing order. Once we find a node whose real shortest path distance is greater than r from the source, we can stop Dijkstra's algorithm, because all the remaining nodes are outside this cluster and we do not care about their real shortest path distances. Finally, if a node is covered by multiple clusters, it will be assigned to its nearest center at the end of this algorithm.

Let S be the *r*-cover returned by Algorithm 1. Then, $\mathcal{N}(\mathcal{D}, r) \leq |S| \leq \mathcal{N}(\mathcal{D}, r/2)$ because all centers in S are more than *r* away from each other. Theoretically, this may not be a good approximation for $\mathcal{N}(\mathcal{D}, r)$, and the design of a better approximation algorithm for covering numbers is still an open problem [3]. However, our goal here is not to compute covering numbers but use the *r*-cover model for clustering. Moreover, we can adjust the returned size |S| by varying the input radius *r* to approximate the number of clusters we want. We will discuss the running time of Algorithm 1 in Section 4.3.

4 Experiments

4.1 Test model - alanine dipeptide

We test our clustering algorithms using kineticallyaware conformational distances on a simple model system, terminally blocked alanine peptide (sequence Ace-Ala-Nme) in explicit solvent. This data set covers both thermodynamic simulations and kinetic simulations useful for testing algorithms analyzing the biomolecular systems, and has already been used in several research papers before [2].

The trajectories were obtained from the 400K replica of a 20ns/replica parallel tempering simulation, and consisted of an equilibrium pool of 1,000 constantenergy, constant-volume trajectory segments 20ps in length with conformations stored every 0.1ps. A small population of the trajectories contained an ω peptide bond in the *cis* state, rather than the typical *trans* state, were removed from the set of trajectories used for analysis, leaving 975 trajectories with a total of 195,000 conformations. The minimum cRMS distance between conformation pairs is 3.54×10^{-2} , and the maximum cRMS distance between conformation pairs is 1.87.

220

Algorithm 1 r-cover clustering

- **Input:** A complete graph $G = \langle V, E \rangle$ and a radius r. **Output:** An r-cover of V, according to shortest path distances. **Procedure:**
- 1) Initialize *r*-cover $S = \phi$.
- 2) Assign to every node a label $\ell(v) = \infty$ (distance to its nearest center).
- 3) Randomly pick an uncovered node s as a new center, $S = S \cup \{s\}$. 4) Assign to every node a distance label: d(s) = 0 and $d(v) = \infty$ for all other nodes.
- 5) Mark all nodes as unvisited.
- 6) Extract node u with smallest d(u) among all unvisited nodes (if
- all nodes are visited, go to step 12).
- 7) If d(u) > r, go to step 12.
- 8) If d(u) ≥ ℓ(u), go to step 11.
 9) Update ℓ(u) = d(u) (assign node u into this new cluster).
- 10) Update $d(v) = min\{d(v), d(u) + w(u, v)\}$ for all unvisited node v.
- 11) Mark node u as visited, go to step 6.
- 12) If all nodes are covered, return S; otherwise, go to step 3.



Figure 5: The terminally blocked alanine dipeptide with ϕ, ψ, ω backbone torsions are labeled on the left. Potential of mean force and state decompositions for alanine dipeptide are labeled manually on the right. This picture is taken from [2].

In the protein backbone geometry, although there are many degrees of freedom, many of these are not important and what really matters are only a few local angles: the torsion angles ϕ and ψ (see Figure 5) are the primary degrees of freedom on the backbone. Since the slow degrees of freedom (ϕ and ψ) are known a priori, it is relatively straightforward to manually identify metastable states from examination of the potential of mean force, making it a popular choice for the study of biomolecular dynamics. Previously, a master equation model constructed using six manually identified states was shown to reproduce dynamics over long times. We therefore determine whether our algorithms can recover a model of equivalent utility to this manually constructed six-state decomposition for this system. Because the algorithm uses the solute Cartesian coordinates, rather than the (ϕ, ψ) torsions, this is a good test of whether good approximations to the true metastable states can be discovered without prior knowledge of the slow degrees of freedom.

For ease of visualization, we still project the state assignments onto the (ϕ, ψ) torsion map for comparison with the manually constructed states. As depicted in Figure 5, a two-dimensional potential of mean force at 400K in the (ϕ, ψ) backbone torsions was estimated from the parallel tempering simulation using the weighted histogram analysis method by discretizing



Figure 6: Good manual state decompositions and automatic state decompositions with their implied timescales plots. This picture is taken from [2].

each degree of freedom into 10° bins. The six such states identified in the previous study can be seen adequately separate the free energy basins observed at 400K. In [2], the authors designed an automatic state decomposition algorithm using the method of splitting and lumping to get a good clustering result (see Figure 6). We will take these decompositions as our references of groundtruth decomposition and compare the results from our algorithms with them.

4.2 Clustering results

Among the six states in the manual state decompositions (see Figure 5), states 1 and 2 are the two densest clusters. It is usually difficult to distinguish these two states using the original cRMS distance, because they are more kinetically distinct rather than structurally distinct. States 3 and 4 are two large clusters that are also difficult to be distinguished, but the internal kinetic barrier separating them is smaller than the barrier separating states 1 and 2. The remaining two states 5 and 6 have much smaller sizes than states 1-4.

The clustering result using the cRMS distance is shown in Figure 7-(1). It turns out that directly applying the cRMS metric will cluster states 1 and 2 together. Thus, such a clustering result is a poor decomposition because its states include internal kinetic barriers.

We first test the clustering quality of kinetically-aware conformational distances using delayed coordinates. We set the weights $w_{\ell} = \exp(-\lambda|\ell|)$ which decay exponentially around the center. When the window size h = 0, the metric is simply cRMS. Figure 7-(2-4) shows the clustering results with decay rate $\lambda = 1$ and window sizes h = 2, 5, 10 respectively. We can see that as we increase the window size, the conformations in states 1 and 2 become separated. For h = 10, the returned six clusters are almost in the same locations as the groundtruth (Figure 7-(4)). If we further increase the window size h, the clustering result will not change too much, because conformations that are far from the center have small weights w_{ℓ} in the distance function (1).

Figure 7-(5,6) shows two more clustering results that are close to the groundtruth decomposition with different decay rates λ . In Figure 7-(5), the decay rate $\lambda = 0$ and sample window size h = 12, so all conformations in the sample window are equally weighted. As a result, the boundaries of the clustering result become ambiguous as there are many outliers in the (ϕ, ψ) torsion map. In Figure 7-(6), the decay rate $\lambda = 0.5$ and the sample window size h = 12. By letting $\lambda > 0$, we can reduce the number of outliers significantly.

We have also implemented the alternative approach where we find only one transformation that jointly align two series of conformations in the sample window. As depicted in Figure 7-(7,8), we use decay rates and window sizes ($\lambda = 0.5, h = 5$) and ($\lambda = 1, h = 5$) respectively. The clustering quality is also very close to the groundtruth. Notice that the clustering results converge faster in this case because the weights w_{ℓ} are squared in the objective function (2).

We finally test the kinetically-aware conformational distances using shortest paths (see Figure 7-(9-12)), which use discount factors and radii (c = 0.9, r = 1.1), (c = 0.8, r = 1.0), (c = 0.7, r = 1.0) and (c = 0.5, r = 0.9) respectively. We can see that as we decrease the discount factor c, more kinetic information is incorporated into the distance function, and thus the conformations in states 1 and 2 become separated. For c = 0.7, the clustering quality is closest to the groundtruth (Figure 7-(11)). When the discount factor c is too small, conformations from the same trajectory are more likely to be clustered together, while in this case we will observe that the conformations in states 3 and 4 are merged into a single cluster (Figure 7-(12)).

For validation, we examine the implied timescales as a function of lag time (τ) , as computed from the eigenvalues of the transition matrix [5]. Theoretically, if the model is Markovian, then the implied timescales will be independent of the lag time for large τ . Figure 8 shows the estimated implied timescales (in picoseconds) as a function of lag time for good decompotitions in Figure 7-(4), (6), (7), (8) and (11) respectively, indicating that they can reproduce dynamics over long times.

4.3 Running time analysis

In this section, we investigate the running time of our clustering algorithms. For k-center clustering, the farthest-first traversal algorithm takes $\mathcal{O}(kn)$ time, which is fairly efficient. For r-cover clustering, we set the discount factor c = 0.8 and generate clusters of different sizes by varying the input radius r. The empirical runtime of Algorithm 1 is shown in Table 1. Such an experiment is performed on a computer cluster with AMD Opteron(tm) Processor 250 and 16GB Memory. When $r \to \infty$, the r-cover contains only one node, so the run-

Radius r	0	0.10	0.15	0.20	0.25
Size of r-cover	195,000	42,479	4,826	1,042	377
Runtime (hour)	24.9	63.3	142.3	81.7	109.4
Radius r	0.30	0.40	0.50	1.00	2.00
Size of r-cover	180	67	33	6	1
Runtime (hour)	88.1	65.6	67.9	53.3	24.5

Table 1: Running time of r-cover clustering algorithm.



Figure 7: Clustering results with kinetically-aware conformational distances.



Figure 8: Implied timescales as a function of lag time. The metastability Q is the sum of the self-transition probabilities of the Markov transition matrix.

ning time is $\Theta(n^2)$ by running Dijkstra's algorithm once in a complete graph. As we decrease the radius r, the size of r-cover increases, but we can save more running time from Dijkstra's algorithm because we will never compute the real shortest path distances between nodes that are greater than r. Finally, when r = 0, the rcover contains all nodes in the graph, so we only relax one node (the source) in each Dijkstra's computation, and the total running time is also $\Theta(n^2)$.

For $0 < r < \infty$, the running time is $\Omega(n^2)$ since all nodes in the graph are covered, and it would be larger than those two extreme cases because there exists overlap between different clusters. However, the experimental results in Table 1 show that Algorithm 1 usually runs in $\mathcal{O}(n^2)$ time in practice, which is significantly faster than farthest-first traversal for large k.

5 Conclusions and future work

In this paper, we designed and tested algorithms that use kinetically-aware distances to cluster biomolecular conformations. The proposed approach outperforms the existing methods that only use geometric information within biomolecules to build distance functions. The shortest path graph distance is of particular interest for constructing metric spaces on a discrete point set: Once we have a distance function, we need to check whether it satisfies the triangle inequality. If not, we can always form a new metric by using shortest path graph distances. Therefore, it would be interesting to derive a theoretical upper bound on the expected running time of Algorithm 1, or develop other efficient algorithms for clustering using shortest path distances. This would be a topic for our future research.

Acknowledgments

This research was supported by NSF grant IIS 0914833, NSF/NIH grant 0900700, as well as ARO grant W911NF-10-1-0037. The authors wish to thank Xuhui Huang and Lutz Maibaum for their helpful comments and suggestions.

References

- [1] P. Brion and E. Westhof. Hierarchy and dynamics of RNA folding. Annual review of biophysics and biomolecular structure, 26, 1997.
- J. Chodera, N. Singhal, V. Pande, K. Dill, and W. [2]Swope. Automatic discovery of metastable states for the construction of Markov models of macromolecular conformational dynamics. Journal of Chemical Physics, 126(15), 2007.
- [3] S. Dasgupta. Lecture notes on unsupervised learning. http://cseweb.ucsd.edu/~dasgupta/291/, 2008.
- C. Dobson. Protein folding and misfolding. Nature, 426(6968), 2003.
- [5] X. Huang, Y. Yao, G. Bowman, J. Sun, L. Guibas, G. Carlsson, and V. Pande. Constructing multi-resolution Markov state models to elucidate RNA hairpin folding mechanisms. Pacific Symposium on Biocomputing, 2010.
- [6] L. Kavraki. Molecular distance measures. Connexions, http://cnx.org/content/m11608/1.23/, 2007.
- [7] R. Marshall, C. Aitken, M. Dorywalska, and J. Puglisi. Translation at the single-molecule level. Annual Review of Biochemistry, 77, 2008.
- [8] S. Plotkin. Lecture notes on advanced algorithms. http://cs361b.stanford.edu/, 2010.
- [9] B. Steipe. A revised proof of the metric properties of optimally superimposed vector sets. Acta Crystallographica Section A, 58(5), 2002.

Collinearities in Kinetic Point Sets

Ben D. Lund*

George B. Purdy[†]

Justin W. Smith[‡]

Csaba D. Tóth[§]

Abstract

Let P be a set of n points in the plane, each point moving along a given trajectory. A k-collinearity is a pair (L, t) of a line L and a time t such that L contains at least k points at time t, L is spanned by the points at time t (*i.e.*, the points along L are not all coincident), and not all of the points are collinear at all times. We show that, if the points move with constant velocity, then the number of 3-collinearities is at most $2\binom{n}{3}$, and this bound is tight. There are n points having $\Omega(n^3/k^4 + n^2/k^2)$ distinct k-collinearities. Thus, the number of kcollinearities among n points, for constant k, is $O(n^3)$, and this bound is asymptotically tight. In addition, there are n points, moving in pairwise distinct directions with different speeds, such that no three points are ever collinear.

1 Introduction

Geometric computation of moving objects is often supported by kinetic data structures (KDS), introduced by Basch, Guibas and Hershberger [1, 5]. The combinatorial structure of a configuration is described by a set of certificates, each of which is an algebraic relation over a constant number of points. The data structure is updated only if a certificates fails. A key parameter of a KDS is the maximum total number of certificate failures over all possible simple motions of n objects. For typical tessellations (*e.g.*, triangulations [8] or pseudotriangulation [10]) or moving points in the plane, a basic certificate is the orientation of a triplet of points, which changes only if the three points are collinear.

We are interested in the maximum and minimum number of collinearities among n kinetic points in the plane, each of which moves with constant velocity. Velocity is speed and direction combined. A *k*-collinearity is a pair (L, t) of a line L and a time t such that L contains at least k points at time t, the points along L do not all coincide, and not all of them are collinear at all times. The last two conditions help to discard a continuum of trivial collinearities: we are not interested in kpoints that coincide, or are always collinear (*e.g.* if they move with the same velocity).

Results. The maximum number of 3-collinearities among n kinetic points in the plane, each moving with constant velocity, is $2\binom{n}{3}$. In particular, if three points are not always collinear, then they become collinear at most twice. Moreover, the maximum is attained for a kinetic point set where no three points are always collinear. We also show that, for constant k, the number of k-collinearities is $O(n^3)$, and this bound is asymptotically tight. In the lower bound construction, the number of k-collinearities is $\Omega(n^3/k^4 + n^2/k^2)$ such that at each k-collinearity at most $\lceil k/2 \rceil$ of the points are always collinear.

The minimum number of collinearities among n kinetic points in the plane is obviously 0. Consider, for example, n points in general position that have the same velocity. We construct n kinetic points that move with pairwise distinct speeds in different directions, and yet they admit no 3-collinearities.

Preliminaries. We assume an infinite time frame $(-\infty, \infty)$. The motion of a point p in \mathbb{R}^d can be represented by its trajectory in \mathbb{R}^{d+1} , where the last ("vertical") dimension is time. If a point p moves with constant velocity in \mathbb{R}^d , its trajectory is a nonhorizontal line $L_p \subset \mathbb{R}^{d+1}$. Every algebraic condition on kinetic points in \mathbb{R}^d has an equivalent formulation in terms of their trajectories in \mathbb{R}^{d+1} . We use both representations throughout this paper.

Related previous results. Previous research primarily focused on collisions. Two kinetic points $p, q \in \mathbb{R}^d$ collide if and only if their trajectories $L_p, L_q \subset \mathbb{R}^{d+1}$ intersect. A *k*-collision is a pair (P,t) of a point $P \in \mathbb{R}^d$ and a time *t* such that at least *k* kinetic points meet at *P* at time *t*, but not all these points are always coincident. It is easy to see that for *n* points in \mathbb{R}^1 , each moving with constant velocity, the number of 2-collisions is at most $\binom{n}{2}$, and this bound is tight. The number of *k*-collisions in \mathbb{R}^1 is $O(n^2/k^3 + n/k)$, and this bound is also the best possible, due to the Szemerédi-Trotter theorem [12].

Without additional constraints, the bounds for the number of collisions remains the same in \mathbb{R}^d for every $d \geq 1$, since the points may be collinear at all times. Sharir and Feldman [11, 4] considered the number of

^{*}lund.ben@gmail.com. Department of Computer Science, University of Cincinnati, Cincinnati, OH 45221, USA.

[†]george.purdy@uc.edu. Department of Computer Science, University of Cincinnati, Cincinnati, OH 45221, USA.

[‡]smith5jw@mail.uc.edu. Department of Computer Science, University of Cincinnati, Cincinnati, OH 45221, USA.

[§]cdtoth@ucalgary.ca. Department of Mathematics and Statistics, University of Calgary, Calgary, AB, Canada.

3-collisions in the plane among points that are not always collinear. The trajectories of such a 3-collision form a so-called "joint" in 3-space. Formally, in an arrangement of n lines in \mathbb{R}^{d+1} , a *joint* is a point incident to at least d + 1 lines, not all of which lie in a hyperplane. Recently, Guth and Katz [6] proved that n lines in \mathbb{R}^3 determine $O(n^{3/2})$ joints. Their proof was later generalized and simplified [3, 9]: n lines in \mathbb{R}^{d+1} determine $O(n^{(d+1)/d})$ joints. These bounds are the best possible, since $\Theta(n^{(d+1)/d})$ joints can be realized by naxis-parallel lines arranged in a grid-like fashion in \mathbb{R}^d . However no nontrivial bound is known for the number of joints under the additional constraint that no d lines lie in a hyperplane.

A k-collinearity is the natural generalization of a kcollision in dimensions $d \ge 2$. It is easy to give a $\Theta(n^3)$ bound on the maximum number of 3-collinearities in the plane, since three random points, with random velocities, form $\Theta(1)$ collinearities in expectation. However, a 4-collinearity assumes an algebraic constraint on the trajectories of the 4 kinetic points. Here we present initial results about a new concept, including tight bounds on the number of 3-collinearities in the plane, and asymptotically tight bounds on the number of k-collinearities in the plane, for constant k.

Organization. We present our results for the maximum number of 3- and k-collinearities in Section 2. We construct a kinetic point set with no collinearities in Section 3 and conclude with open problems in Section 4.

2 Upper bound for 3-collinearities

Given any two kinetic points a and b in the plane, denote by $S_{a,b}$ the set of point-time pairs in \mathbb{R}^3 that form a 3collinearity with a and b. This will be the set of all horizontal lines that intersect both L_a and L_b . We can find the times at which a third point, c, is collinear with a and b by characterizing the set $L_c \cap S_{a,b}$. In particular, the cardinality of $L_c \cap S_{a,b}$ is the number of 3-collinearities formed by these three points.

The first issue is to characterize the set $S_{a,b}$. For this purpose, we will use a classical geometric result.

Lemma 1 (14.4.6 from [2]) Let L_a and L_b be disjoint lines in a three-dimensional Euclidean affine space, and let a and b be points moving along L_a and L_b with constant speed. The affine line through a and b describes a hyperbolic paraboloid as t ranges from $-\infty$ to ∞ .

This is a special case of a construction that produces a hyperboloid of one sheet or a hyperbolic paraboloid from three skew lines [7, p. 15]. Given three skew lines, the union of all lines that intersect all three given lines is a doubly ruled surface. If the three given lines are all parallel to some plane, the surface will be a hyperbolic paraboloid; otherwise, the surface will be a hyperboloid of a single sheet.

Given two kinetic points a and b moving at constant velocity, we can arbitrarily choose three horizontal lines that intersect L_a and L_b to use with the above construction. Since horizontal lines are parallel to a horizontal plane, the resulting surface will be a hyperbolic paraboloid.

This characterizes $S_{a,b}$ in the case that L_a and L_b are skew. It remains to extend the characterization to the cases that a and b collide or have the same speed and direction.

Lemma 2 Given two kinetic points, a and b, each moving with constant velocity, there are three possibilities for $S_{a,b}$.

- 1. If a and b have the same direction and speed, then $S_{a,b}$ is a non-horizontal plane.
- 2. If a and b collide, then $S_{a,b}$ is the union of a horizontal and a non-horizontal plane.
- 3. Otherwise, $S_{a,b}$ is a hyperbolic paraboloid.

Proof. If L_a and L_b intersect or are parallel, then there is a unique plane Π that contains both L_a and L_b . Since neither L_a nor L_b is horizontal, Π is not horizontal. Every point in Π belongs to the union of all horizontal lines containing a point from each of L_a and L_b .

Since two non-coincident points span a unique line and the intersection of Π with a horizontal plane is a line, if L_a and L_b are parallel, then $S_{a,b} = \Pi$. This covers the case that a and b have the same direction and speed.

If L_a and L_b intersect, then every point in the horizontal plane Π' containing the intersection point $L_a \cap L_b$ is on a horizontal line containing a point from each of L_a and L_b . In this case, $S_{a,b} = \Pi \cup \Pi'$. This covers the case that a and b collide.

If L_a and L_b are skew, Lemma 1 implies that $S_{a,b}$ is a hyperbolic paraboloid. This covers the generic case. \Box

Lemma 3 Three points in the plane, each moving with constant velocity, will either be always collinear or collinear at no more than two distinct times.

Proof. Label the points a, b, and c. By lemma 2, $S_{a,b}$ is a plane, the union of two planes, or a hyperbolic paraboloid. Every time L_c intersects $S_{a,b}$, the points a, b, and c are collinear. Since a plane is a surface of degree 1 and a hyperbolic paraboloid is a surface of degree 2, L_c cannot intersect $S_{a,b}$ more than twice without being contained in $S_{a,b}$.

Theorem 4 A set of n points in the plane, each moving with constant speed and direction, determines no more than $2\binom{n}{3}$ 3-collinearities.

Proof. There are $\binom{n}{3}$ subsets of 3 points, each of which forms at most two 3-collinearities.

Clearly, this bound applies directly to k-collinearities, for any $k \ge 3$. If no three points are always collinear, this bound can easily be improved for k > 3.

Theorem 5 A set of n points in the plane, each moving with constant speed and direction, and no three of which are always collinear, determines no more than $2\binom{n}{3}/\binom{k}{3}$ k-collinearities.

Proof. By Theorem 4, there are at most $2\binom{n}{3}$ sets of 3 instantaneously collinear points. A *k*-collinearity accounts for at least $\binom{k}{3}$ distinct sets of 3 instantaneously collinear points.

2.1 The $2\binom{n}{3}$ bound is tight for 3-collinearities

Theorem 6 Theorem 4 is tight for the case k = 3.

Proof. We construct a set of *n* kinetic points, no three always collinear, such that they admit exactly $2\binom{n}{3}$ 3-collinearities.

Let the points be $\{p_1, p_2, ..., p_n\}$. Each point moves with speed 1. The direction of motion of point p_i forms an angle of $\theta_i = 3\pi/2 + \pi/(4i)$ with the positive x direction. At time t = 0, each point is on a circle of radius 1 centered at (-1, 1), and positioned so that its direction of travel will cause it to cross the origin at some later time. Since two locations on the circle might satisfy this property, we choose the one closer to the origin (Fig. 1).



Figure 1: A set of kinetic points forming $2\binom{n}{3}$ three point lines over the time interval $(-\infty, \infty)$, at time 0.

At time t = 0, no three points are collinear, so no triplet of points is always collinear. Choose any three elements from $\{p_1, p_2, ..., p_n\}$, say p_j , p_k , and p_l , such that j > k > l, so $\theta_j < \theta_k < \theta_l$. We will show that these points are collinear at two distinct times.

Let H_L and H_R denote the left and right halfplanes, respectively, determined by the directed line $p_j p_l$. Let C be a closed curve passing through p_j , p_k , and p_l such that it crosses line $p_j p_l$ at p_j and p_l only. We can determine which half-plane contains p_k from the cyclic order of the three points on C. If the clockwise order is (p_j, p_k, p_l) , then $p_k \in H_L$; if the clockwise order is (p_j, p_l, p_k) , then $p_k \in H_R$.

At time 0, the points are distributed on the circle of radius 1 with center (-1, 1), and the clockwise order of the chosen points on this circle is (p_j, p_k, p_l) . Thus, $p_k \in H_L$.

Let c_i be the distance between p_i and the origin at time 0. Since all points are initially moving toward the origin at a speed of 1, the distance between p_i and the origin is $|c_i - t|$ at time t.

We now establish that p_k is in H_R for $|t| \gg 1$. If $t \gg 1$, all of the points $\{p_1, p_2, ..., p_n\}$ will lie approximately on a circle of radius t centered at the origin. The clockwise order of the points on a convex curve approximating this circle will be (p_j, p_l, p_k) , and $p_k \in H_R$. Likewise, when $t \ll -1$ the points will be approximately on a circle of radius |t| (but at points antipodal to those when $t \gg 1$), and the order will be (p_j, p_l, p_k) with $p_k \in H_R$. Figure 2 depicts the configuration for $t \gg 1$.



Figure 2: A set of kinetic points forming $2\binom{n}{3}$ three point lines over the time interval $(-\infty, \infty)$, at time $\gg 1$.

Since p_k alternates from H_R to H_L and back to H_R as t goes from negative to positive infinity, there must exist times t' and t'' at which the three points are collinear.

The above construction is degenerate in the sense that the paths of the points are all concurrent through the origin. Note that our argument is not sensitive to a small perturbation in the location or the direction of the points. The direction of motion of each point may be perturbed so that the trajectories are in general position.

Additionally, the construction may be altered so that the points travel at different speeds. If the speeds of $\{p_1, p_2, ..., p_n\}$ are not all the same, then the points will not approach a circle as |t| approaches ∞ . However, as long as no three points are always collinear and the points approach some closed convex curve as |t| approaches infinity, the arguments used will remain valid. For example, if the speed of point p_i is $1/(1-\cos(\theta_i)/2)$, then for $|t| \gg 1$, the points will be approximately distributed on an ellipse enclosing the origin. This ensures that any three points will be collinear at two distinct times, so the set of n points will have $2\binom{n}{3}$ 3collinearities.

2.2 The $O(n^3)$ bound is tight for fixed k

By Theorem 4, n kinetic points moving with constant velocities determine $O(n^3)$ k-collinearities. Here for all integers $n \ge k \ge 3$, we construct a set of n kinetic points that determines $\Omega(n^3/k^4 + n^2/k^2)$ k-collinearities.

First assume that $n \geq k^2$. We construct *n* kinetic points with $\Omega(n^3/k^4)$ *k*-collinearities. The points will move on two parallel lines $L_1: x = 0$ and $L_2: x = 1$ in varying speeds. A simultaneous $\lfloor k/2 \rfloor$ -collision on L_1 and a $\lfloor k/2 \rfloor$ -collision on L_2 will define a *k*-collinearity.

Without loss of generality we may assume that n is a multiple of k. Let $\{A_1, A_2, ..., A_{\lfloor k/2 \rfloor}\}$ and $\{B_1, B_2, ..., B_{\lceil k/2 \rceil}\}$ be sets of n/k points each. At time 0, let

$$A_i = \{a_{i,j} = (0,j) : j = 1, ..., n/k\} \text{ for } 1 \le i \le \lfloor k/2 \rfloor, \\ B_i = \{b_{i,j} = (1,j) : j = 1, ..., n/k\} \text{ for } 1 \le j \le \lceil k/2 \rceil.$$

All points move in direction (0, 1). The points in $A = \bigcup_{i=1}^{\lfloor k/2 \rfloor} A_i$ are always in line x = 0, and the point in $B = \bigcup_{i=1}^{\lceil k/2 \rceil} B_i$ are always in line x = 1. Let the speed of every point in A_i or B_i be i - 1. For example, each point in set A_1 has speed 0.

At each time $t \in \{0, 1, ..., n/(k\lfloor k/2 \rfloor)\}$, there are $(n/k - (k - 1)t) \lfloor k/2 \rfloor$ -way collisions among points in A and $(n/k - (k - 1)t) \lceil k/2 \rceil$ -way collisions among points in B. Each line connecting a $\lfloor k/2 \rfloor$ -collision among points in A and a $\lceil k/2 \rceil$ -collision among points in A and a $\lceil k/2 \rceil$ -collision among points in A and a $\lceil k/2 \rceil$ -collision among points in B is a k-collinearity. Thus, at each time $t \in \{0, 1, ..., n/(k\lfloor k/2 \rfloor)\}$, there are $(n/k - (k - 1)t)^2$ distinct k-collinearities. Taking the sum, the number of k-collinearities over $t = [0, \infty)$ is

$$\sum_{t=0}^{n/(k\lfloor k/2 \rfloor)} (n/k - (k-1)t)^2 \geq \sum_{t=0}^{n/(k\lfloor k/2 \rfloor)} (k-1)^2 t^2$$
$$\geq (k-1)^2 \sum_{t=0}^{n/(k\lfloor k/2 \rfloor)} t^2$$
$$= \Omega(n^3/k^4).$$

Now assume that $k \leq n < k^2$. We construct *n* kinetic points with $\Omega(n^2/k^2)$ *k*-collinearities. The *n* points are partitioned into subsets, $A_1, A_2, \ldots, A_{\lfloor n/k \rfloor}$, each of size at least $\lceil k/2 \rceil$. The points in each subset have a single $\lceil k/2 \rceil$ -collision at time 0, at points in general position in the plane. Any line between two $\lceil k/2 \rceil$ -collisions is a *k*collinearity. Hence there are *k*-collinearities is $\Omega(n^2/k^2)$.

3 Kinetic point sets with no collinearities

It is clearly possible to have no 3-collinearities among n kinetic points if the points move with the same direction and speed—this is simply a set of relatively static points, no three of which are collinear. Similarly, if we are only interested in collinearities in the time interval $(0, \infty)$, it is clearly possible to have no collinearities—any set of kinetic points will have a final 3-collinearity.

Less obviously, we can construct n kinetic points, any two of which have different direction and speed, that admit no 3-collinearities over the time interval $(-\infty, \infty)$.

Theorem 7 For every integer $n \ge 1$, there is a set of n points in the plane, each moving with constant speed and direction, no two of the points having the same speed or direction, such that no three points are collinear over the time interval $(-\infty, \infty)$.

Proof. We will start by constructing a set of kinetic points with no 3-collinearities, having different directions but the same speed. Then, we will modify the construction so that the points move with different speeds.

For $1 \leq i \leq n$, let $\theta_i = \pi/2 + \pi/2i$. At time 0, place point p_i at a distance of 1 from the origin at an angle of θ_i from the positive x direction. Each point moves with speed 1 in the direction $\theta_i - \pi/2$, tangent to the circle containing the points (see Fig. 3).



Figure 3: A set of points, each moving at speed 1, of which no three are ever collinear.

By this construction, the lines $L(p_i)$ will be from one ruling of a hyperboloid of a single sheet S [7]. The intersection of any horizontal plane with S will be a circle. Since no line intersects a circle in more than two points, there will never be three points on any line.

In order to modify this construction so that no two points have the same speed, we will stretch it in the x-direction.

For $1 \leq i \leq n$, if p_i is at location (x_i, y_i) at time 0, then place point p'_i at location $(2x_i, y_i)$. If the velocity vector of p_i is $(v_{(x,i)}, v_{(y,i)})$, then the velocity vector of p'_i is $(2v_{(x,i)}, v_{(y,i)})$ (see Fig. 4).



Figure 4: A set of points, no two moving at the same speed, of which no three are ever collinear.

Since no two points $p_i, p_j \in \{p_1, p_2, ..., p_n\}$ have the same x component to the vector describing their motion, no two points $p'_i, p'_j \in \{p_1, p_2, ..., p_n\}$ have the same speed.

The lines $L_{p'_i}$ are from one ruling of a hyperboloid of a single sheet S'. The main difference between S and S' is that S' is stretched in the *x*-direction, so the intersection of any horizontal plane with S' is an ellipse rather than a circle. No line intersects an ellipse in more than two points, so again there will never be three points on any line.

4 Conclusion

We derived tight bounds on the minimum and maximum number of 3- and k-collinearities among n kinetic points, each moving with constant velocity in the plane. Our initial study poses more questions than it answers.

Open Problem 1 What is the maximum number of kcollinearities among n kinetic points in the plane? Is our lower bound $\Omega(n^3/k^4 + n^2/k^2)$ tight?

Open Problem 2 What is the maximum number of kcollinearities among n kinetic points in the plane if no three points are always collinear and no two points collide?

Open Problem 3 What is the maximum number of 3collinearities among n kinetic points in the plane if the trajectory of each point is an algebraic curve of degree bounded by a constant b?

Open Problem 4 A d-collinearity in \mathbb{R}^d is called fulldimensional if not all points involved in the collinearity are in a hyperplane at all times. What is the maximum number of full-dimensional d-collinearities among n kinetic points in \mathbb{R}^d ?

The trajectories of n kinetic points in \mathbb{R}^d is an arrangement of n nonhorizontal lines in \mathbb{R}^{d+1} . Recall that a k-collinearity corresponds to a *horizontal* line that intersects k trajectories. If we drop the restriction to horizontal lines, we are led to the following problem.

Open Problem 5 For an arrangement \mathcal{A} of n lines in \mathbb{R}^3 , what is the maximum number of lines L such that L intersects at least 3 lines in \mathcal{A} , which are not all concurrent and not all from a single ruling of a doubly ruled surface?

References

- J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–28, 1999.
- [2] M. Berger. Geometry. II. Universitext. Springer-Verlag, Berlin, 1987. Translated from the French by M. Cole and S. Levy.
- [3] G. Elekes, H. Kaplan, and M. Sharir. On lines, joints, and incidences in three dimensions. J. Comb. Theory, Ser. A, 118(3):962–977, 2011.
- [4] S. Feldman and M. Sharir. An improved bound for joints in arrangements of lines in space. Discrete & Computational Geometry, 33(2):307–320, 2005.
- [5] L. Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23–1–23–18. Chapman and Hall/CRC, 2004.
- [6] L. Guth and N. H. Katz. Algebraic methods in discrete analogues of the kakeya problem. Adv. in Math., 225:2828–2839, 2010.
- [7] D. Hilbert and S. Cohn-Vossen. Geometry and the imagination. Chelsea Publishing Company, New York, NY, 1952. Translated by P. Neményi.
- [8] H. Kaplan, N. Rubin, and M. Sharir. A kinetic triangulation scheme for moving points in the plane. *Comput. Geom.*, 44(4):191–205, 2011.
- [9] H. Kaplan, M. Sharir, and E. Shustin. On lines and joints. Discrete & Computational Geometry, 44(4):838– 843, 2010.
- [10] D. G. Kirkpatrick and B. Speckmann. Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. In *Sympos. on Comput. Geom.*, pages 179–188. ACM Press, 2002.
- [11] M. Sharir. On joints in arrangements of lines in space and related problems. J. Comb. Theory, Ser. A, 67(1):89–99, 1994.
- [12] E. Szemerédi and W. T. Trotter, Jr. Extremal problems in discrete geometry. *Combinatorica*, 3(3-4):381–392, 1983.

Convexifying Polygons Without Losing Visibilities

Oswin Aichholzer* Ferran Hurtado[¶] Greg Aloupis[†]

Erik D. Demaine[‡]

Martin L. Demaine[‡]

Vida Dujmović[§]

Anna Lubiw[∥]

Günter Rote** André Schulz^{††} Diane L. Souvaine^{‡‡}

Andrew Winslow^{‡‡}

Abstract

We show that any simple *n*-vertex polygon can be made convex, without losing internal visibilities between vertices, using n moves. Each move translates a vertex of the current polygon along an edge to a neighbouring vertex. In general, a vertex of the current polygon represents a set of vertices of the original polygon that have become co-incident.

We also show how to modify the method so that vertices become very close but not co-incident, in which case we need $O(n^2)$ moves, where each move translates a single vertex.

The proof involves a new visibility property of polygons, namely that every simple polygon has a visibility*increasing edge* where, as a point travels from one endpoint of the edge to the other, the visibility region of the point increases.

1 Introduction

There are many interesting problems about reconfiguring geometric structures while maintaining some properties. Examples include: flips in triangulations [5], pushing and sliding block puzzles [17], morphing of polygons and planar graphs [18, 21], and linkage reconfiguration [7, 23]. Reconfiguration has also been studied outside the geometric domain [19].

This paper is about *convexifying* a simple polygon, i.e., continuously transforming the polygon to a convex polygon while maintaining simplicity. If no other structure must be maintained, this can be done in a trivial way, moving only one vertex at a time. When edge lengths must be maintained, this is a major result, namely the Carpenter's Rule Theorem [7, 23], and the reconfiguration process involves moving all vertices simultaneously.

In the Open Problem session at CCCG 2008 [11], Satyan Devadoss asked whether a polygon can be convexified without losing internal visibility between any pair of vertices, and in particular, whether this can be done by moving only one vertex at a time [12]. We give a positive answer. We first consider a version of the problem where vertices may become co-incident during the transformation, so one vertex of the polygon in general represents a set of vertices of the original polygon. We show that any polygon can be convexified by a sequence of n moves, where each move strictly increases the set of pairs of vertices that are internally visible, and each move translates one vertex along an edge of the polygon to a neighbour. In terms of the original polygon, each move translates a set of vertices along a straight line to join another set of vertices.

In Section 3 we modify our method so that vertices become very close but not coincident. In this case, we need $O(n^2)$ moves, each moving one vertex. Internal vertex visibilities are never lost, but a single move does not necessarily add any internal vertex visibilities.

Our main tool, which may be of independent interest, is to show that every polygon has a visibility-increasing edge where, as a point travels from one endpoint of the edge to the other, the visibility region of the point increases.

Previous Work

In the original model where coincident vertices are not allowed, Aichholzer et al. [2] showed that any monotone polygon can be convexified without losing vertex visibilities. Their transformation moves one vertex at a time, but the number of vertex moves is not polynomially bounded. If all vertices may move simultaneously,

^{*}Institute for Software Technology, University of Technology Graz, Austria, oaich@ist.tugraz.at. Partially supported by the FWF under grant S9205-N12 NRN Industrial Geoemtry.

[†]Département d'Informatique, Université Libre de Bruxelles, aloupis.greg@gmail.com

[‡]MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine, mdemaine}@mit.edu

[§]School of Computer Science, Carleton University, vida@scs.carleton.ca

[¶]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, ferran.hurtado@upc.edu. Partially supported by projects MICINN MTM2009-07242 and Gen. Cat. DGR 2009SGR1040

School of Computer Science, University of Waterloo, alubiw@uwaterloo.ca

^{*}Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany. rote@inf.fu-berlin.de

^{††}Institut für Mathematische Logik und Grundlagenforschung, Universität Münster, Germany, andre.schulz@uni-muenster.de

^{‡‡}Department of Computer Science, Tufts University, {dls, awinslow}@cs.tufts.edu. Research supported in part by NSF grants CCF-0830734 and CBET-0941538.

they observe that a monotone polygon can be convexified in one move. They also show that, even for monotone polygons, it is not always possible to move just one vertex and strictly increase the set of vertex visibilities. Note that such an example depends crucially on prohibiting coincident vertices! If vertices are allowed to be coincident, our result shows that for any simple polygon, it is possible to move one vertex until it gains a new neighbour in the visibility graph.

The issue of allowing/disallowing coincident vertices has arisen before in problems of transforming (or "morphing") polygons and straight-line graph drawings. Cairns [6] showed how to transform between any two straight-line planar triangulations that are combinatorially the same, using a sequence of moves each of which translates one vertex onto another (or the reverse). He then comments that it is possible to avoid coincident vertices by keeping them a small distance apart. A somewhat similar issue comes up in the result of Guibas and Hershberger [16] who show that for any two simple polygons on vertices $1, 2, \ldots, n$ such that edge (i, i+1)has the same direction vector in both polygons, there is a morph between the polygons that preserves simplicity and the direction vectors of edges. Their method moves vertices infinitesimally close together and operates on the infinitesimal structures.

The question of moving only one vertex at a time has recently been settled in independent work by Ábrego et al. [1], who show that if a there is a transformation that convexifies a polygon without losing vertex visibilities, then the transformation can be accomplished by moving only one vertex at a time.

Although not directly relevant to this paper, we note that there is a considerable body of work on making polygons convex by means of "pivot" operations, such as *flips* [13, 8, 15, 25] and *flipturns* [3, 4].

For background on visibility graphs of polygons, see the books by Ghosh [14] and O'Rourke [22].

Definitions

Two points inside a polygon P are visible if the line segment between them is contained in the closed polygon. Given this definition, we will now use "visibility" rather than "internal visibility". We will assume that the input polygon does not have three or more collinear vertices. It is possible to perturb a polygon to achieve this without losing internal vertex visibilities. Note the consequence that if two vertices are visible, then the line segment between them does not go through another vertex. For point p in P, the visibility region of p, denoted V(p), is the set of points in P visible from p.

Let a be a reflex vertex with neighbours b and b' on the polygon boundary. Extend a line segment from b to a and beyond, until it first hits the polygon boundary at p. Define Pocket(b, a) to be the region bounded by the chain along the polygon boundary from a to p going through b', together with the line segment pa. We consider points along the line segment pa to be outside the pocket (i.e., the pocket is open along its "mouth"). In particular, a is outside Pocket(b, a). See the shaded region in Figure 1(a).

2 Convexifying polygons

Theorem 1 An n-vertex polygon can be convexified in n moves, where each move strictly increases the set of pairs of visible vertices, and each move translates one vertex of the current polygon along an incident edge to a neighbour on the polygon boundary.

The main tool in proving the theorem is the following. We prove that if a polygon is not convex then it has an edge along which visibility increases. More precisely, define an edge (u, v) to be a *visibility-increasing edge* if for every point p along the edge (u, v) we have $V(u) \subseteq V(p) \subseteq V(v)$, and there is a vertex in V(v) - V(u).

We will use a stronger induction hypothesis to prove that every non-convex polygon has a visibilityincreasing edge (u, v) where v is a reflex vertex. Note that the fact that v is reflex implies that there is a vertex in V(v) - V(u).

Lemma 2 Let P be a simple polygon with reflex vertex a and edge (b, a). Then there is a visibility-increasing edge (u, v) with v reflex and u, v exterior to Pocket(b, a) such that u does not see into Pocket(b, a).

Proof. We prove the result by induction on the number of reflex vertices of the polygon exterior to the pocket. If (b, a) is a visibility-increasing edge, then it satisfies the lemma, since b does not see into Pocket(b, a). See Figure 1(a). This takes care of the base case where every vertex $v \neq a$ exterior to the pocket is convex.



Figure 1: Visibility-increasing edges: (a) the edge (b, a) is a visibility-increasing edge; (b) vertex b is reflex, so we apply induction on (c, b).

If b is a reflex vertex then let c be the other neighbour of b (i.e., the neighbour not equal to a). See Figure 1(b). Then $\text{Pocket}(c, b) \supseteq \text{Pocket}(b, a)$. Also, note that the reflex vertex a is exterior to Pocket(b, a) and

not exterior to Pocket(c, b). Therefore we can apply induction to conclude that there is a visibility-increasing edge (u, v) exterior to Pocket(c, b) such that v is reflex and u does not see into Pocket(c, b). Then u cannot see into Pocket(b, a), so (u, v) satisfies the lemma.



Figure 2: Visibility-increasing edges in the general case, where we apply induction on (y, x).

We are left with the case where b is a convex vertex but (b, a) is not a visibility-increasing edge. Note that because a is a reflex vertex, V(a) contains a vertex not in V(b). Therefore, the only way that (b, a) can fail to be visibility-increasing is that there is a point p on (b, a)and a point t on the boundary of P such that t sees p, but t does not see a. See Figure 2. Now we rotate the line through t and p about t until it hits the polygon boundary. More precisely, consider the first point q along the line segment pa such that the line segment qt does not lie in the interior of P. Then some vertex xlies on the line segment qt. Note that x must be a reflex vertex. There are two paths on the polygon boundary from x to t. Take the path that does not contain a, and let y be the neighbour of x on this path. (It may happen that y = t.) We will apply induction on the edge (y, x). Observe that $Pocket(y, x) \supseteq Pocket(b, a)$. Also, note that the reflex vertex a is exterior to Pocket(b, a) and not exterior to Pocket(y, x). Therefore we can apply induction to conclude that there is a visibility-increasing edge (u, v) exterior to Pocket(y, x) such that v is reflex and u does not see into Pocket(y, x). Then u cannot see into Pocket(b, a), so (u, v) satisfies the lemma.

Proof. [of Theorem 1] The proof is by induction on the number of vertices. If the polygon has three vertices then it is already convex. For the general case, if the polygon is convex then there is nothing to prove, so suppose there is a reflex vertex. Then by Lemma 2, there is a visibility-increasing edge (u, v). The plan is to move vertex u to vertex v, resulting in a simple polygon with fewer vertices on which we apply induction. See Figure 5. Let w be the other neighbour of u on the polygon boundary. We have $V(u) \subseteq V(v)$ and $w \in V(u)$, so w must be visible to v. In particular, u is a convex vertex and the line segment wv does not intersect the polygon boundary except at its endpoints. Therefore moving u to v results in a simple polygon. Observe that no vertex visibilities are affected by the move, except that u gains visibilities once it reaches v (if not before). Note that u may become collinear with two other vertices of the polygon at an intermediate point of the move, but this causes no problems. \Box

3 Avoiding coincident vertices

In the previous section we showed how to convexify any polygon without losing internal visibilities, provided that vertices are allowed to become coincident. In this section we show how to avoid coincident vertices. Each set of coincident vertices from the previous method is replaced by a cluster of vertices that are close together but not coincident. One move of the previous method becomes O(n) moves, each moving a single vertex. The total number of moves is therefore $O(n^2)$. Vertex visibilities are never lost, but a single move might not increase vertex visibilities.



Figure 3: Cluster vertices along a single edge (top); a reflex cluster (left); and a convex cluster (right). Shaded areas indicate the interior of the polygon.

The basic idea is to replace an edge uv by a slightly outward-bent convex chain, with some points on a shallow convex curve close to u, and other points on a shallow convex curve close to v, see Figure 3 (top). In general, a *cluster* will consist of a *representative vertex* v, together with the vertices that have been moved to join v, and now lie on two convex curves incident to v. The representative vertex v will be at the same point in the plane as it was in the original polygon. If C is a cluster with representative vertex v, we will say that C is the cluster of v. Figure 3 depicts a *reflex* and a *convex* cluster. In a convex cluster all vertices see each other; in a reflex cluster only vertices in the same arc see each other, and the representative vertex sees the whole cluster.

All vertices of a cluster lie in the ε -neighbourhood of

the representative vertex for some sufficiently small ε . In addition, all vertices of a cluster lie within some angle α of the original edge. See Figure 4(a).

We define values for ϵ and α that will work throughout the algorithm. As the convexification proceeds, edges between representative vertices of the intermediate polygons are always chords of the original polygon. We will take all the chords into account when we define ε and α . We choose ε small enough that visibility between two points in the ε -neighbourhoods of two vertices x and y behaves like visibility between x and y. Thus ε should be smaller than the distance between any vertex and a (non-coincident) chord or edge extension—see Figure 4(b). We choose α small enough that a representative vertex x does not block visibilities of vertices in its cluster, and that a convex vertex remains convexsee Figure 4(c). Apart from the constraints imposed by ϵ and α we are free to place the cluster vertices on any convex chain, and we will have occasion to alter the chain.



Figure 4: (a) Cluster vertices are located in the shaded region determined by ε and α ; (b) Constraints on ε , which must be small enough that visibility from a point within an ϵ -neighbourhood of a vertex acts like visibility from the vertex; (c) Constraints on α , which must be small enough that a vertex x does not block visibility to its cluster.

We now consider the move operation from the previous section as it operates on clusters. The move operation always moves a convex vertex u to join a reflex vertex v. See Figure 5. The only other vertex affected by the move is w, the other neighbour of u, which forms a triangle with u and v. Suppose without loss of generality that v, u, w are in clockwise order around the polygon. When vertices are replaced by clusters, the vertices affected by the move are: all of u's cluster; the left part of v's cluster; and the right part of w's cluster. See Figure 6. Note that, although the original move always increased the set of vertices visible from u, the modified move will not necessarily increase visibility from u or any of its cluster, since we do not move any vertex all the way to v.



Figure 5: Moving vertex u along the visibility-increasing edge (u, v) affects vertices u, v, and w, which form a triangle. Vertex v may remain reflex (left) or become convex (right).



Figure 6: The operation from Figure 5 in the presence of cluster vertices: (a) the initial configuration, the final configuration shown with dashed lines, and the vertex correspondence shown with thin lines; (b) the intermediate configuration after moving u and its cluster close to v.

We show that the transformation of clusters as shown in Figure 6 can be accomplished by moving one vertex at a time. The first phase is to move u and its cluster close to v, in a configuration congruent to their final configuration. Move the vertices one by one starting with the vertex closest to v along the chain. Note that u loses its status as a representative vertex. The result of the first phase is shown in Figure 6(b). Note that the union of the initial and final positions of all the vertices that are moved in the first phase is in convex position. Therefore, convexity of the cluster and visibility within the cluster are maintained. Globally, as each cluster vertex moves from the neighbourhood of u's initial position to v's neighbourhood, its visibility changes exactly as u's visibility changed in the original non-cluster move (stopping just before reaching v).

In the second phase (from Figure 6(b) to the final

configuration) the transformation we wish to realize is a counterclockwise rotation of w's right cluster and a clockwise rotation of v's left cluster to their final positions. We describe how to do this for v's left cluster. In the first step, move the vertices of v's left cluster (one by one) close enough to v that their new positions and their final positions are in convex position, as shown in Figure 7. In the second step, move the vertices one by one to their final positions, starting with the vertex farthest from v along the chain. Convexity of the cluster (and hence visibility within the cluster) is maintained during the second step because the union of the initial and final positions of all moved vertices is in convex position. Global visibilities may be gained but are never lost.



Figure 7: Adjusting the position of v's left cluster vertices. All movement takes place within the ϵ -neighbourhood of v. The first vertex move is shown with a thin directed line. Note that this figure is not to scale since the angle α should be much smaller.

From the above ideas, we obtain the following result.

Theorem 3 An n-vertex polygon can be convexified in $O(n^2)$ moves, so that visibilities between vertices are never lost, and vertices never become coincident. Each move is a translation of a single vertex.

4 Discussion and Open Problems

We have shown that any simple *n*-vertex polygon can be convexified in $O(n^2)$ single-vertex moves without ever decreasing the visibility graph, answering a question posed by Devadoss et al. [12]. If coincident vertices are allowed, then *n* moves suffice, and each move strictly increases the visibility graph.

In the same paper, Devados et al. ask about transforming a polygon to decrease the visibility graph: can any simple polygon be transformed to a polygon whose visibility graph is a triangulation without ever increasing the visibility graph? This question remains open. For orthogonal polygons, it would be desirable to maintain orthogonality. We conjecture than every simple orthogonal polygon can be convexified (i.e., transformed to a rectangle) without losing visibilities, while maintaining orthogonality. A minimal motion that maintains orthogonality is to move one edge orthogonal to itself (i.e., a horizontal edge moves vertically, and vice versa). However, Figure 8 shows an example where no edge can be moved orthogonally to gain visibilities.

It is possible that the current result can be generalized to straight line drawings of planar graphs: Given a planar graph embedded in the plane as a straight-line drawing, is it possible to transform the drawing so that every internal face becomes convex, while remaining straightline planar, and without losing internal visibilities? Our result is the special case where the drawing has only one internal face. The fact that such a transformation is possible, ignoring visibility constraints, is not at all obvious, but follows from the result by Thomassen [24], who showed (based on a result of Cairns [6]) that there is a transformation between any two straight-line planar drawings of the same embedded graph that preserves straight-line planarity. Vertices become coincident during this transformation, although that can be avoided by keeping them close but distinct. The number of vertex movements is not polynomially bounded. For further discussion on morphing of graph drawings, see [20, 21].

Finally, we make two remarks about our result on the existence of a visibility-increasing edge in any simple polygon. Since good things (like ears of polygons) come in pairs, it is natural to ask whether every simple polygon has *two* visibility-increasing edges.

Visibility-increasing edges may have other uses in the study of visibility graphs. A major open question is whether visibility graphs of polygons can be recognized in polynomial time (with or without the information about which edges form the polygon boundary). This is Problem 17 in the Open Problems Project [9].



Figure 8: An orthogonal polygon where no single edge can be moved orthogonally to gain visibilities.

5 Acknowledgments

This work was begun at the 26th Bellairs Workshop on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint. We thank the other participants of the workshop for stimulating discussions.

References

- B. M. Ábrego, M. Cetina, J. Leaños, and G. Salazar. Visibility-preserving convexifications using singlevertex moves. http://arxiv.org/pdf/1105.3435v1.
- [2] O. Aichholzer, M. Cetina, R. Fabila-Monroy, J. Leanos, G. Salazar, and J. Urrutia. Convexifying monotone polygons maintaining internal visibility. Extended abstract, XIV Spanish Meeting on Computational Geometry, Alcalá de Henares, Spain, pages 35–38, 2011.
- [3] O. Aichholzer, C. Cortés, E. D. Demaine, V. Dujmovic, J. Erickson, H. Meijer, M. Overmars, B. Palop, S. Ramaswami, and G. Toussaint. Flipturning polygons. *Discrete & Computational Geometry*, 28:231–253, 2002.
- [4] T. Biedl. Polygons Needing Many Flipturns. Discrete & Computational Geometry, 35:131-141, 2006.
- [5] P. Bose and F. Hurtado. Flips in planar graphs. Computational Geometry, 42:60–80, 2009.
- [6] S. S. Cairns. Deformations of plane rectilinear complexes. Amer. Math. Monthly, 51(5):247–252, 1944.
- [7] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30:205–239, 2003.
- [8] B. de Sz.-Nagy. Solution of problem 3763. Amer. Math. Monthly, 49:176–177, 1939.
- [9] E. D. Demaine, J. Mitchell, and J. O'Rourke. The open problems project. http://maven.smith.edu/~/TOPP/, May 2010.
- [10] E. D. Demaine and J. O'Rourke. Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, 2007.
- [11] E. D. Demaine and J. O'Rourke. Open problems from CCCG 2008. In Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG), pages 75–78, 2009.
- [12] S. L. Devadoss, R. Shah, X. Shao, and E. Winston. Visibility graphs and deformations of associahedra. http: //arxiv.org/abs/0903.2848.
- [13] P. Erdős. Problem 3763. Amer. Math. Monthly, 42:42, 1935.
- [14] S. Ghosh. Visibility Algorithms in the Plane. Cambridge University Press, 2007.
- [15] B. Grünbaum. How to convexify a polygon. Geocombinatorics, 5:24–30, 1995.
- [16] L. Guibas and J. Hershberger. Morphing simple polygons. In Proceedings of the 10th Annual Symposium on Computational Geometry, SCG '94, pages 267–276, New York, NY, USA, 1994. ACM.

- [17] R. A. Hearn and E. D. Demaine Games, puzzles, and computation. A. K. Peters, Ltd., 2009.
- [18] H. N. Iben, J. F. O'Brien and E. D. Demaine. Refolding Planar Polygons. In Proceedings of the 22nd Annual Symposium on Computational Geometry, pages 71–79, 2006. ACM.
- [19] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara and Y. Uno On the Complexity of Reconfiguration Problems. *Theoretical Computer Science*, in press.
- [20] A. Lubiw and M. Petrick. Morphing planar graph drawings with bent edges. *Electronic Notes in Discrete Mathematics*, 31:45 – 48, 2008. The International Conference on Topological and Geometric Graph Theory.
- [21] A. Lubiw, M. Petrick, and M. Spriggs. Morphing orthogonal planar graph drawings. In *Proceedings of the* 17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '06, pages 222–230, 2006. ACM.
- [22] J. O'Rourke. Art gallery theorems and algorithms. Oxford University Press, Inc., 1987.
- [23] I. Streinu. Pseudo-triangulations, rigidity and motion planning. Discrete & Computational Geometry, 34:587– 635, 2005.
- [24] C. Thomassen. Deformations of plane graphs. Journal of Combinatorial Theory, Series B, 34(3):244 – 257, 1983.
- [25] G. Toussaint. The Erdös-Nagy theorem and its ramifications. In Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG), pages 219– 236, 1999.

Expansive Motions for *d*-Dimensional Open Chains

Erik D. Demaine*

Sarah Eisenstat*

Abstract

We consider the problem of straightening chains in $d \geq 3$ dimensions, possibly embedded into higher dimensions, using expansive motions. For any $d \geq 3$, we show that there is an open chain in d dimensions that is not straight and not self-touching yet has no expansive motion. Furthermore, for any $\Delta > 0$ and $d \geq 3$, we show that there is an open chain in d dimensions that cannot be straightened using expansive motions when embedded into $\mathbb{R}^d \times [-\Delta, \Delta]$ (a bounded extra dimension). On the positive side, we prove that any open chain in $d \geq 2$ dimensions can be straightened using an expansive motion when embedded into \mathbb{R}^{d+1} (a full extra dimension).

1 Introduction

Expansive motions have proved to be a powerful technique for reconfiguring planar linkages. The purpose of this paper is to determine how useful they can be in higher dimensions.

Expansive motions first proved useful by providing the key to solving the Carpenter's Rule Problem [7,10], whether every planar chain linkage (forming a path or a cycle) could be universally reconfigured by motions that preserve edge lengths and avoid self-crossings. A positive answer was established by proving that every planar open chain that is not straight, and every planar closed chain that is not convex, has an *expansive* motion in the sense that no two vertices ever get closer together. While it is difficult to avoid self-crossings directly, expansiveness implies such avoidance, and expansive motions are easier to argue about because of their relation to tensegrities (explained below).

Expansive motions have since proved useful in establishing universal reconfigurability of other types of planar linkages. Streinu and Whiteley [11] extended the result to sufficiently short chains on a sphere, which has applications to rigid folding of single-vertex origami crease patterns. Connelly et al. [5] proved that chains of "slender adornments" can be universally configured, using expansive motions of an underlying chain linkage. This result was in turn useful for avoiding selfintersection in universal hinged dissections [1]. Expansive motions also led to the extensive study of "pointed pseudotriangulations", because pointed pseudotriangulations are the extreme rays in the cone of expansive motions [8, 10].

Beyond two dimensions, linkages have been studied, but not in the context of expansive motions. Chain linkages cannot be universally reconfigured in three dimensions [2], implying that some nonstraight open chain has no expansive motion. One might hope that the subset of 3D open chains that can be straightened can do so via expansive motions. In 4D and higher, the situation is more promising: chain linkages can be universally reconfigured [3], via fairly simple algorithms. A natural question, therefore, is whether all 4D and higherdimensional chains have expansive motions.

Our results. Alas, we prove the existence of open chains in d dimensions, for all $d \geq 3$, that are not straight yet have no expansive motion. Furthermore, we can (for d = 3) guarantee that the chain can be straightened (by nonexpansive motions), shooting down the hope that such chains have expansive motions. We start by constructing a self-touching chain with this property, and then prove that there exist sufficiently small perturbations of the chain that still have the property and are non-self-touching.

Next we consider how many extra dimensions we have to add to *d*-dimensional space to guarantee expansive motions of a chain that lives in a *d*-dimensional (sub)space. On the one hand, we show that adding a bounded dimension $[-\Delta, \Delta]$ does not suffice: for any $\Delta > 0$, there is a *d*-dimensional open chain that has no expansive motion in $\mathbb{R}^d \times [-\Delta, \Delta]$. On the other hand, we show that a full extra dimension suffices: any *d*-dimensional open chain has an expansive motion all the way to straight in d + 1 dimensions.

2 Preliminaries

We begin by defining tensegrity frameworks, using a modified version of the notation in [4,9].

An abstract tensegrity is an undirected graph G, with vertices V(G) and edges $E(G) = B(G) \cup C(G) \cup S(G)$, where B(G), C(G), and S(G) are pairwise disjoint. The edges $e \in B(G)$ are bars whose lengths are fixed. The edges $e \in C(G)$ are cables whose lengths cannot increase. The edges $e \in S(G)$ are struts whose lengths

^{*}MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA. {edemaine, seisenst}@mit.edu. Research supported in part by NSF CA-REER award CCF-0347776.

cannot decrease. A tensegrity framework G(p) in d dimensions consists of an abstract tensegrity G and a mapping $p: V(G) \to \mathbb{R}^d$ assigning a location to each vertex in the abstract tensegrity. Equivalently, we may consider p to be a point in \mathbb{R}^{nd} , where n = |V(G)|. For convenience, we say that for each $k \in \{1, 2, \ldots, d\}$, the function p_k gives the kth coordinate of p, so that $p(v_i) = (p_1(v_i), p_2(v_i), \ldots, p_d(v_i))$ for each v_i .

A linkage framework G(p) is a tensegrity framework such that $C(G) = S(G) = \emptyset$. An open chain of length n is a linkage framework G(p)such that $V(G) = \{v_1, v_2, \ldots, v_n\}$ and B(G) = $\{(v_1, v_2); (v_2, v_3); \ldots; (v_{n-1}, v_n)\}$. In this paper, we use the terms chain and open chain interchangeably.

We say that G(q) is another embedding of a tense grity framework G(p) if $q(v_1) = p(v_1)$ and all of the following conditions hold:

$$\begin{aligned} \forall (v_i, v_j) \in B(G) : \|q(v_i) - q(v_j)\| &= \|p(v_i) - p(v_j)\|, \\ \forall (v_i, v_j) \in C(G) : \|q(v_i) - q(v_j)\| \leq \|p(v_i) - p(v_j)\|, \\ \forall (v_i, v_j) \in S(G) : \|q(v_i) - q(v_j)\| \geq \|p(v_i) - p(v_j)\|. \end{aligned}$$

Note that this relation is transitive, but not generally symmetric. Note also the non-standard requirement that $q(v_1) = p(v_1)$. This is used to ensure that any framework G(p) where $(V(G), B(G) \cup C(G))$ is a connected graph has a bounded configuration space.

A tensegrity framework G(p) is *self-touching* if there exist four distinct vertices v_i, v_j, v_k , and v_ℓ such that the edges $(v_i, v_j), (v_k, v_\ell) \in E(G)$, and the segment between $p(v_i)$ and $p(v_j)$ intersects with the segment between $p(v_k)$ and $p(v_\ell)$. If a tensegrity framework G(p) with $B(G) \cup S(G) = V(G) \times V(G)$ is not self-touching, then any other embedding of G(p) is also not self-touching.

A motion of a framework G(p) is a continuous mapping from a time $t \in [0, 1]$ to a configuration p^t such that $p^0 = p$ and each $G(p^t)$ is another embedding of G(p). A rigid transformation T is a distance-preserving transformation of \mathbb{R}^d . We use the notation T(p) to denote the result of applying T to every vertex location $p(v_i)$. A motion is rigid if at every time t, there exists a rigid transformation T such that $p^t = T(p)$. A framework G(p) is rigid if all motions of G(p) are rigid motions.

An expansive motion is a motion where the distance between any pair of vertices is always non-decreasing. More formally, an *expansive motion* is a motion p^t such that for all pairs of vertices v_i, v_j , and for all times t < t', $\|p^t(v_i) - p^t(v_j)\| \leq \|p^{t'}(v_i) - p^{t'}(v_j)\|$. We say that a tensegrity framework G(p) is *rigid under expansive motion* if any expansive motion p^t is rigid.

An alternate embedding G(q) is reachable from G(p)if there is some motion p^t of G(p) such that $p^0 = p$ and $p^1 = q$. A framework G(p) is *locked* if there exists an alternate embedding of G(p) that is not reachable from G(p). A chain G(p) is *straight* if it is not selftouching and all vertices lie along a line. Note that all straight chains are rigid under expansive motions.

An *infinitesimal motion* of a framework G(p) assigns a velocity vector $u(v_i)$ to every vertex v_i which satisfies the following constraints:

$$\begin{aligned} \forall (v_i, v_j) \in B(G) : (u(v_i) - u(v_j)) \cdot (p(v_i) - p(v_j)) &= 0, \\ \forall (v_i, v_j) \in C(G) : (u(v_i) - u(v_j)) \cdot (p(v_i) - p(v_j)) &\leq 0, \\ \forall (v_i, v_j) \in S(G) : (u(v_i) - u(v_j)) \cdot (p(v_i) - p(v_j)) &\geq 0. \end{aligned}$$

These equations can be derived by taking the first derivative of the equations used to test whether some p^t is a motion for the framework G(p), and setting t = 0. An infinitesimal motion of G(p) is a rigid infinitesimal motion if it is equal to the derivative of some rigid motion at time t = 0. We say that G(p) is infinitesimally rigid if all infinitesimal motions of G(p) are rigid.

For a fixed choice of p, each of the infinitesimal motion equations is a linear inequality over u. Hence, if G(p) is a linkage framework, then the constraints for infinitesimal motions can be written as a matrix with |E(G)|rows and dn columns. The rank of this matrix can be used to calculate the number of degrees of freedom for infinitesimal motions. Hence, a linkage framework is infinitesimally rigid if and only if the rank of the matrix is equal to dn - d(d+1)/2. This gives a simple way to test for infinitesimal rigidity, but only for linkages.

One way to test whether a tense grity framework is infinitesimally rigid involves a concept called stress. A stress on a tense grity framework G(p) assigns a scalar value $s(v_i, v_j)$ to each edge $(v_i, v_j) \in E(G)$. Intuitively, the stress on some edge (v_i, v_j) applies a force proportional to $s(v_i, v_j)$ to both v_i and v_j . A stress $s(v_i, v_j)$ is known as an equilibrium stress if it satisfies the following constraint:

$$\forall v_i \in V(G), k \in \{1, \dots, d\} : \\ \sum_{v_j : (v_i, v_j) \in E(G)} s(v_i, v_j) \cdot (p_k(v_i) - p_k(v_j)) = 0.$$

Roth and Whiteley [9] showed the following:

Theorem 1 [9] Let G(p) be a tensegrity framework. Let \overline{G} be an abstract tensegrity framework such that $V(\overline{G}) = V(G), B(\overline{G}) = E(G), \text{ and } C(\overline{G}) = S(\overline{G}) = \emptyset$. Then G(p) is infinitesimally rigid if and only if $\overline{G}(p)$ is infinitesimally rigid and there exists an equilibrium stress on G(p) such that $s(v_i, v_j) > 0$ for all cables (v_i, v_j) and $s(v_i, v_j) < 0$ for all struts (v_i, v_j) .

This theorem makes it easier to show the infinitesimal rigidity of a tensegrity.

One key theorem that we use several times in this paper is a result by Connelly [4] which has come to be known as sloppy rigidity [6]. Intuitively, if a framework G(p) is rigid, then even if the edge lengths vary by



Figure 1: The three-dimensional version of the self-touching chain used in Lemma 3.

some small δ , any motion can only perturb the vertices of G(p) by some small amount. Hence, G(p) remains locked even with weaker constraints on the edge lengths. The theorem uses the idea of a rigidity neighborhood. A *rigidity neighborhood* U_p of some rigid framework G(p)is any set that contains p and all of its rigid transformations, but does not contain any other q such that G(q)is an alternate embedding of G(p).

Theorem 2 [4] Let G(p) be rigid in \mathbb{R}^n , and let U_p be a rigidity neighborhood of p for G(p). Let $\varepsilon > 0$ be given. Then there is some $\delta > 0$ such that, if $q \in U_p$ and the following conditions hold

$$\begin{aligned} \forall (v_i, v_j) \in C(G) \cup B(G) : \\ \|q(v_i) - q(v_j)\|^2 < \|p(v_i) - p(v_j)\|^2 + \delta, \\ \forall (v_i, v_j) \in S(G) \cup B(G) : \\ \|q(v_i) - q(v_j)\|^2 > \|p(v_i) - p(v_j)\|^2 - \delta, \end{aligned}$$

then there is a rigid transformation T of \mathbb{R}^d such that $||T(q) - p|| < \varepsilon$.

3 Chains Rigid Under Expansive Motions

In this section, we show that for any $d \ge 3$, there exists a non-straight chain in d dimensions that is rigid under expansive motions. To do so, we begin by giving a selftouching chain that is rigid under expansive motions, which we later modify to make non-self-touching.

3.1 Self-Touching

Lemma 3 For any $d \ge 3$, there exists a self-touching configuration of a chain in d dimensions that is rigid under expansive motions but is not straight.

Proof. Consider the chain G(p) of length 2d with

$$p_k(v_i) = \begin{cases} 1 & \text{if } i = 2k - 1 \\ -1 & \text{if } i = 2k, \\ 0 & \text{otherwise,} \end{cases}$$

where $1 \leq k \leq d$ and $1 \leq i \leq 2d$. Intuitively, this chain is the result of connecting *d* bars, each of length 2, and each of which lies along one of the axes of \mathbb{R}^d and is centered on the origin. The three-dimensional version of this chain is depicted in Figure 1.

Consider the tensegrity framework G'(p) obtained by adding a strut between every pair of unconnected vertices. Proving that G'(p) is rigid is equivalent to proving that G(p) is rigid under expansive motions. To show this, we must first provide an equilibrium stress for the tensegrity that is negative on every strut. We define the stress $s(v_i, v_j)$ between two vertices $v_i \neq v_j$ as follows:

$$s(v_i, v_j) = \begin{cases} d-1 & \text{if } \exists k \text{ s.t. } \{i, j\} = \{2k-1, 2k\}, \\ -1 & \text{otherwise.} \end{cases}$$

It is easy to verify that this is an equilibrium stress of G'(p), and that it is negative on all of the struts of G'(p).

To show that G'(p) is infinitesimally rigid, we must also show that replacing every strut in the tensegrity with a bar results in a linkage that is infinitesimally rigid. Consider the linkage framework $\overline{G}(p)$ that results from this process. There is a bar between every pair of vertices, so the linkage is clearly rigid. To see that it is infinitesimally rigid, we use an inductive argument on the number of dimensions for the chain.

We claim that, for any $d \geq 3$, there exists a set of d(d+1)/2 linear equations that, when combined with the constraints on the velocities for an infinitesimal motion, yield only the zero solution. This claim shows that the framework has d(d+1)/2 degrees of freedom, and is therefore infinitesimally rigid.

We show this claim by induction on d. The claim can be verified for d = 3. Assume by induction that there exist d(d+1)/2 linear equations that, when combined with the $2d^2 - d$ constraints for the *d*-dimensional version of this chain, restrict the space of solutions so that no infinitesimal motions are allowed. Now consider the (d+1)-dimensional version of this chain. Any infinitesimal motion must satisfy $2(d+1)^2 - (d+1)$ linear constraints, one for each edge. The first 2d vertices of the chain have the same coordinates as the vertices of the d-dimensional version of the chain. Hence, the $2d^2 - d$ constraints corresponding to each edge among those vertices are the same for the two chains. By adding the d(d+1)/2 linear equations guaranteed to exist by the inductive assumption, we ensure that, for any $k \neq d+1$ and any $i \notin \{2d+1, 2d+2\}, u_k(v_i) = 0.$

We now add (d + 1) more linear equations setting $u_k(v_{2d+1}) = 0$ for all $k \in \{1, \ldots, d+1\}$, for a total of (d+1)(d+2)/2 extra equations, which is precisely what we wanted. Consider the possible values for $u_{d+1}(v_i)$, for some $i \notin \{2d+1, 2d+2\}$. The edge between v_{2k-1} and v_{2d+1} results in the following equation:

$$(u(v_{2k-1}) - u(v_{2d+1})) \cdot (p(v_{2k-1}) - p(v_{2d+1})) = 0.$$

The vector $(p(v_{2k-1}) - p(v_{2d+1}))$ is non-zero in two coordinates: k and d+1. In addition, the d+1 equations we added ensure that $u(v_{2d+1}) = (0, 0, ..., 0)$. Hence, the above equation becomes

$$1 \cdot u_k(v_{2k-1}) + -1 \cdot u_{d+1}(v_{2k-1}) = 0.$$

The d(d+1)/2 equations from the inductive step ensure that $u_k(v_{2k-1}) = 0$. Hence, $u_{d+1}(v_{2k-1}) = 0$. A similar argument shows that $u_{d+1}(v_{2k}) = 0$ for any $k \neq d+1$. This means that the equations we have selected ensure that the velocity of every vertex $v_i \neq v_{2d+2}$ is zero.

Now consider the velocity of v_{2d+2} . The edge between v_{2d+1} and v_{2d+2} gives us the equation:

$$2 \cdot u_{d+1}(v_{2d+1}) - 2 \cdot u_{d+1}(v_{2d+2}) = 0.$$

Hence, we know that $u_{d+1}(v_{2d+2}) = 0$. Now consider the edge between v_{2d+2} and v_{2k-1} , which results in the following equation:

$$(u(v_{2k-1}) - u(v_{2d+2})) \cdot (p(v_{2k-1}) - p(v_{2d+2})) = 0.$$

The velocity $u(v_{2k-1})$ is zero in all coordinates, and $(p(v_{2k-1}) - p(v_{2d+2}))$ is nonzero only in coordinates k and d+1. Therefore, the equation becomes:

$$1 \cdot -u_k(v_{2d+2}) + 1 \cdot -u_{d+1}(v_{2d+2}) = 0.$$

We have shown that $u_{d+1}(v_{2d+2}) = 0$, and so it must be that $u_k(v_{2d+2}) = 0$.

3.2 Non-Self-Touching

Now that we have shown the existence of a self-touching chain that is rigid under expansive motions, we use the sloppy rigidity results from Theorem 2 to show that there is also a non-self-touching chain that is rigid under expansive motions.

Theorem 4 For any $d \ge 3$, there exists a non-selftouching configuration of a chain in d dimensions that is rigid under expansive motions but is not straight.

Proof. Consider the chain G(p) specified in Lemma 3. Add a strut between every pair of vertices that is not already connected by a bar to obtain a rigid tensegrity framework G'(p). Because G'(p) is rigid, there must exist some c such that any other embedding G'(q) that is not a rigid transformation of p has ||p - q|| > c.

Now let U_p be a neighborhood of p such that $\forall r \in U_p$, there exists a rigid transformation T such that ||T(r) - p|| < c. By definition, U_p is a rigidity neighborhood of p. Let $\varepsilon = c/2$, be the value we use for Theorem 2, and let $\delta > 0$ be the resulting sloppiness.

If $\delta > \varepsilon/n$, set $\delta = \varepsilon/n$. Randomly perturb each vertex in p by a distance less than $\delta/2$ to get a framework $G'(p^*)$. Then any alternate embedding of $G'(p^*)$

has the property that its edge lengths violate the length constraints for an alternate embedding of G'(p) by a distance of at most δ . By Theorem 2, if there is any $q \in U_p$ such that G'(q) is another embedding of $G'(p^*)$, then there must exist a rigid transformation T such that $||T(q) - p|| < \varepsilon$, and therefore

$$||T(q) - p^*|| < ||T(q) - p|| + ||p - p^*|| < \varepsilon + n\delta/2 \le \frac{3c}{4}.$$

Hence, any motion of $G'(p^*)$ cannot result in any framework G'(q) such that $q \notin U_p$. As a result, $G'(p^*)$ is locked, and all reachable alternative embeddings of $G'(p^*)$ are not straight.

Define the function f(q) to be the sum of all pairwise distances between the points in G'(q). The set of alternate embeddings reachable from $G'(p^*)$ is both closed and bounded, so we can pick an embedding $G'(q^*)$ from that set that maximizes $f(\cdot)$.

Consider any alternate embedding G(r) reachable from $G(q^*)$. Because G'(r) is an alternate embedding of $G'(q^*)$, we know that for each edge $(v_i, v_j) \in E(G')$, $||r(v_i) - r(v_j)|| \ge ||q^*(v_i) - q^*(v_j)||$. Hence, we have

$$\sum_{i,j} \|r(v_i) - r(v_j)\| \ge \sum_{i,j} \|q^*(v_i) - q^*(v_j)\|,$$
$$f(r) \ge f(q^*).$$

By transitivity, G'(r) is also a reachable alternate embedding of $G'(p^*)$. By definition of q^* , this means that $f(r) \leq f(q^*)$, and therefore that $f(r) = f(q^*)$. As a result, it must be that $||r(v_i) - r(v_j)|| = ||q^*(v_i) - q^*(v_j)||$ for all edges (v_i, v_j) . Then r is a rigid transformation of q^* . So $G(q^*)$ is rigid under expansive motions.

Because of the random perturbations used to construct p^* , and because $d \ge 3$, no four vertices can lie in the same plane. Hence, $G'(p^*)$ must be non-selftouching. Because any alternate embedding of $G'(p^*)$ cannot decrease the distance between any two vertices, all alternate embeddings are also non-self-intersecting. This means that $G(q^*)$ is also not self-touching. \Box

In Theorem 4, we give a non-constructive proof of the existence of a non-self-touching chain that is rigid under expansive motions in $d \ge 3$ dimensions. The following conjecture would provide a way to construct such a chain, but has only been verified for $3 \le d \le 8$.

Conjecture 1 In $d \ge 3$ dimensions, the non-selftouching chain given by the coordinates

$$p_k(v_i) = \begin{cases} 1 & if \ i = 2k - 1, \\ -1 & if \ i = 2k, \\ 0.01 & if \ \lceil i/2 \rceil + 1 \equiv k \pmod{d}, \\ 0 & otherwise, \end{cases}$$

where $1 \leq k \leq d$ and $1 \leq n \leq 2d$, is rigid under expansive motions.

For d = 3, we can additionally prove that the constructed chain can be straightened: either end link can be folded by itself to extend the next link, thus effectively reducing the number of links to 4, which implies that the chain can be straightened [2].

3.3 Bounded Extra Dimension

We have shown that for any $d \geq 3$, there exists a *d*-dimensional chain that is rigid under expansive motions, but not straight. This naturally raises the question of how much space is required to expansively straighten a *d*-dimensional chain. In this section, we show that for any Δ , there exists a *d*-dimensional chain that cannot be straightened using an expansive motion in $\mathbb{R}^d \times [-\Delta, \Delta]$. We begin by proving the following lemma.

Lemma 5 For any non-straight chain G(p) in d dimensions that is rigid under expansive motions, there exists a constant $\Delta > 0$ such that G(p) cannot be straightened using an expansive motion in $\mathbb{R}^d \times [-\Delta, \Delta]$.

Proof. Let G(p) be a non-straight chain in d dimensions that is rigid under expansive motions. Add a strut between every pair of vertices that are not connected by a bar, and call the resulting framework G'(p). Then our goal is to show that G'(p) is locked in $\mathbb{R}^d \times [-\Delta, \Delta]$.

Let G'(q) be any alternate embedding of G'(p) in $\mathbb{R}^d \times [-\Delta, \Delta]$. Consider projecting G'(q) into d dimensions by omitting the last coordinate to get a framework G'(r). For any v_i and v_j ,

$$\|q(v_i) - q(v_j)\|^2 = \|r(v_i) - r(v_j)\|^2 + (q_{d+1}(v_i) - q_{d+1}(v_j))^2.$$

Because G'(q) is embedded in $\mathbb{R}^d \times [-\Delta, \Delta]$, we know that $0 \leq (q_{d+1}(v_i) - q_{d+1}(v_j))^2 \leq 4\Delta^2$. Hence we have

$$||r(v_i) - r(v_j)||^2 \ge ||q(v_i) - q(v_j)||^2 - 4\Delta^2$$
, and
 $||r(v_i) - r(v_j)||^2 \le ||q(v_i) - q(v_j)||^2$.

This means that any alternate embedding of G'(p) in $\mathbb{R}^d \times [-\Delta, \Delta]$ corresponds to an alternate embedding of G'(p) in \mathbb{R}^d where the edge lengths are allowed to vary by at most $4\Delta^2$. Hence, if G'(p) is locked in \mathbb{R}^d , even when the edge lengths are allowed to vary by up to $4\Delta^2$, then G'(p) is locked in $\mathbb{R}^d \times [-\Delta, \Delta]$.

Because G'(p) is rigid, we know that there is some constant c such that any other embedding G'(q) that is not a rigid transformation of p has ||p - q|| > c. Let U_p be a neighborhood of p such that $\forall r \in U_p$, there is a rigid transformation T such that ||T(r) - p|| < c. Then U_p is a rigidity neighborhood of p. Let $\varepsilon < c$ be the value we use for Theorem 2, and let $\delta > 0$ be the resulting sloppiness. Consider any $r \in U_p$ satisfying these constraints:

$$\begin{aligned} \forall (v_i, v_i) \in B(G') \cup S(G') : \\ \|r(v_i) - r(v_j)\|^2 &\geq \|p(v_i) - p(v_j)\|^2 - 4\Delta^2, \\ \forall (v_i, v_i) \in B(G') \cup C(G') : \\ \|r(v_i) - r(v_j)\|^2 &\leq \|p(v_i) - p(v_j)\|^2. \end{aligned}$$

Then if $4\Delta^2 \leq \delta$, there is a rigid transformation T such that $||T(r) - p|| < \varepsilon$. This means that if $4\Delta^2 \leq \delta$, then G'(p) is locked in $\mathbb{R}^d \times [-\Delta, \Delta]$.

Theorem 6 For any $d \ge 3$ and any $\Delta > 0$, there is a non-self-touching chain in d dimensions that cannot be straightened using an expansive motion in $\mathbb{R}^d \times [-\Delta, \Delta]$.

Proof. By Theorem 4, there is a non-self-touching chain $G(p^*)$ in d dimensions that is rigid under expansive motions. By Lemma 5, there exists some Δ^* such that $G(p^*)$ cannot be expansively straightened in $\mathbb{R}^d \times [-\Delta^*, \Delta^*]$. Multiply the coordinates of p^* by Δ/Δ^* to get a new framework G(p). For the sake of contradiction, say that G(p) can be straightened in $\mathbb{R}^d \times [-\Delta, \Delta]$ using some expansive motion q^t . Multiply all coordinates of q^t by a factor of Δ^*/Δ to get a motion r^t in $\mathbb{R}^d \times [-\Delta^*, \Delta^*]$. Because q^t is a motion of p, the scaled r^t is a motion of p^* . Hence, if G(p) can be straightened in $\mathbb{R}^d \times [-\Delta, \Delta]$, then $G(p^*)$ can be straightened in $\mathbb{R}^d \times [-\Delta^*, \Delta^*]$, which results in a contradiction. \Box

4 Expansive Motions in Higher Dimensions

We have shown that for any Δ , there is a *d*-dimensional chain that cannot be expansively straightened in $\mathbb{R}^d \times$ $[-\Delta, \Delta]$. In this section, we show that any *d*dimensional chain can be expansively straightened in \mathbb{R}^{d+1} , thus resolving the question of how much space is required to straighten expansively.

Theorem 7 Any chain G(p) in d dimensions can be straightened using an expansive motion when embedded in (d + 1)-dimensional space.

Proof. For each v_i , we define z_i to be the length along the chain between v_1 and v_i . More formally, we let

$$z_i = \sum_{j=1}^{i-1} \|p(v_j) - p(v_{j+1})\|$$

Note that for any *i* and *j*, $|z_i - z_j| \ge ||p(v_i) - p(v_j)||$. The motion *q* used to straighten the chain will be:

$$q_k^t(v_i) = \begin{cases} z_i \cdot \sin\left(\frac{\pi t}{2}\right) & \text{if } k = d+1, \\ p_k(v_i) \cdot \cos\left(\frac{\pi t}{2}\right) & \text{otherwise.} \end{cases}$$

At time t = 0, the location of vertex v_i will be $(p_1(v_i), \ldots, p_d(v_i), 0)$. At time t = 1, the location of

vertex v_i will be $(0, 0, \ldots, 0, z_i)$. So our motion will cause all of the points to form a line. Consider how the square of the distance between v_i and v_j will change over time. A short derivation shows that

$$\frac{\|q^t(v_i) - q^t(v_j)\|^2}{\|p(v_i) - p(v_j)\|^2} = \left(\frac{(z_i - z_j)^2}{\|p(v_i) - p(v_j)\|^2} - 1\right)\sin^2\left(\frac{\pi t}{2}\right) + 1.$$

For adjacent v_i, v_j , we know that $||p(v_i) - p(v_j)|| = z_i - z_j$, and therefore the length of the edge is preserved. But is the motion expansive over $t \in [0, 1]$? We know that $\sin^2\left(\frac{\pi t}{2}\right)$ is a non-decreasing function over the interval [0, 1]. We also know that $(z_i - z_j)^2 \ge ||p(v_i) - p(v_j)||^2$. Hence, the distance between v_i and v_j is non-decreasing, meaning that the motion is expansive.

Lemma 8 For any $\Delta > 0$, any three-dimensional open chain can be straightened in $\mathbb{R}^3 \times [-\Delta, \Delta]$.

Proof. Let G(p) be a three-dimensional open chain that is not straight. To straighten G(p), we first use a modified version of the motion from Theorem 7. Define z_i as we did in Theorem 7. Let $\theta = \arcsin(\Delta/z_n)$. The motion we use is

$$q_k^t(v_i) = \begin{cases} z_i \cdot \sin(\theta t) & \text{if } k = 4, \\ p_k(v_i) \cdot \cos(\theta t) & \text{otherwise.} \end{cases}$$

A similar analysis to Theorem 7 shows that this motion preserves the lengths of all bars, and does not cause the chain to become self-intersecting. In addition, we have selected the motion function so that $0 \leq p_{d+1}(v_i) \leq \Delta$.

The result of this motion is an alternate embedding G(q) with the following coordinates:

$$q_k(v_i) = \begin{cases} (z_i \cdot \Delta)/z_n & \text{if } k = 4, \\ p_k(v_i) \cdot \sqrt{1 - (\Delta/z_n)^2} & \text{otherwise} \end{cases}$$

To straighten this new chain, we hold v_2, \ldots, v_n fixed in place, and swing v_1 around so that it is collinear with v_2 and v_3 . More specifically, consider the set of possible locations for v_1 that preserve the fourth coordinate of v_1 and the length of the rigid bar between v_1 and v_2 . This corresponds to a sphere. The initial location of v_1 is one point on the sphere; the location that will straighten the bars (v_1, v_2) and (v_2, v_3) is another point on the sphere. We can use a motion along the shortest path to get from one to the other. Once the two bars (v_1, v_2) and (v_2, v_3) have been straightened, we can treat them as a single bar and repeat until the whole chain is straightened. \Box

5 Open Problem

The main question left open by this work is whether every *closed* chain initially in *d* dimensions has an expansive motion in d+1 dimensions to a (planar) convex configuration. Such a result would be a natural extension to our positive result for open chains. (Our negative results extend to closed chains, simply by doubling our open chains into an Euler tour.)

Acknowledgments

These problems were originally invented in collaboration with Robert Brasseur and Stefan Langerman in 2009. This research was initiated during the openproblem sessions organized around MIT class 6.849: Geometric Folding Algorithms in Fall 2010. We thank the other participants of these sessions — Zachary Abel, Martin Demaine, Isaac Ellowitz, Jason Ku, Jayson Lynch, and TB Schardl — for their helpful ideas and for providing a conducive research environment.

References

- T. G. Abbott, Z. Abel, D. Charlton, E. D. Demaine, M. L. Demaine, and S. D. Kominers. Hinged dissections exist. *Discrete & Computational Geometry*. To appear.
- [2] J. Cantarella and H. Johnston. Nontrivial embeddings of polygonal intervals and unknots in 3-space. *Journal* of Knot Theory and Its Ramifications, 7(8):1027–1039, 1998.
- [3] R. Cocan and J. O'Rourke. Polygonal chains cannot lock in 4D. Discrete & Computational Geometry, 20(3):105–129, November 2001.
- [4] R. Connelly. Rigidity and energy. Inventiones Mathematicae, 66:11–33, 1982. 10.1007/BF01404753.
- [5] R. Connelly, E. D. Demaine, M. L. Demaine, S. Fekete, S. Langerman, J. S. B. Mitchell, A. Ribó, and G. Rote. Locked and unlocked chains of planar shapes. *Discrete* & Computational Geometry, 44(2):439–462, 2010.
- [6] R. Connelly, E. D. Demaine, and G. Rote. Infinitesimally locked self-touching linkages with applications to locked trees. In J. Calvo, K. Millett, and E. Rawdon, editors, *Physical Knots: Knotting, Linking, and Folding* of Geometric Objects in 3-space, pages 287–311. American Mathematical Society, 2002.
- [7] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205– 239, September 2003.
- [8] G. Rote, F. Santos, and I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations. In *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 699–736. Springer-Verlag, 2003.
- [9] B. Roth and W. Whiteley. Tensegrity frameworks. Transactions of the American Mathematical Society, 265(2):pp. 419–446, 1981.
- [10] I. Streinu. Pseudo-triangulations, rigidity and motion planning. Discrete & Computational Geometry, 34(4):587-635, November 2005.
- [11] I. Streinu and W. Whiteley. Single-vertex origami and spherical expansive motions. In *Revised Selected Pa*pers from the Japan Conference on Discrete and Computational Geometry, volume 3742 of Lecture Notes in Computer Science, pages 161–173, Tokyo, Japan, October 2004.

Making triangulations 4-connected using flips

Prosenjit Bose*

Dana Jansens^{*}

André van Renssen^{*}

Maria Saumell[§]

Sander Verdonschot^{*}

Abstract

We show that any triangulation on n vertices can be transformed into a 4-connected one using at most $\lfloor (3n-6)/5 \rfloor$ edge flips. We also give an example of a triangulation that requires $\lceil (3n-10)/5 \rceil$ flips to be made 4-connected, showing that our bound is tight. Our result implies a new upper bound on the diameter of the flip graph of 5.2n - 24.4, improving on the bound of 6n - 30 by Mori *et al.* [4].

1 Introduction

Given a triangulation (a maximal planar simple graph) on a set of n vertices, we define an *edge flip* as removing an edge (a, b) from the graph and replacing it with the edge (c, d), where c and d are the other vertices of the triangles that had (a, b) as an edge. Figure 1 shows an example of an edge flip.

Flips have been studied mostly in two different settings: the *geometric* setting, where we are given a fixed set of points in the plane and edges are straight line segments, and the *combinatorial* setting, where we are only given the clockwise order of edges around each vertex (a combinatorial embedding). In this paper, we concern ourselves with the number of flips required to transform one triangulation into another in the combinatorial setting. We give a brief overview of previous work on this problem. A more detailed overview, including applications and related work, can be found in a survey by Bose and Hurtado [2].



Figure 1: An example triangulation before and after flipping edge (a, b).

Given a set of n vertices, we can define its *flip graph* as the graph with a vertex for each distinct triangulation and an edge between two vertices if their corresponding triangulations differ by a single flip. Two triangulations are considered distinct if they are not isomorphic. In his seminal paper, Wagner [7] showed that there always exists a sequence of $O(n^2)$ flips that transforms a given triangulation into any other triangulation on the same set of vertices. In terms of the flip graph, Wagner showed that it is connected and has diameter $O(n^2)$. Komuro [3] was the first to show that the diameter is linear and Mori *et al.* [4] currently have the strongest upper bound of 6n - 30.

The above results all show how to transform any triangulation into a fixed canonical triangulation. Transformation of one triangulation into another is then straightforward by transforming the first into the canonical triangulation and transforming the canonical triangulation into the second by reversing the sequence of flips for the second. Mori *et al.*'s algorithm to transform a triangulation into the canonical one consists of two steps. They first make the given triangulation 4connected using at most n-4 flips. Since a 4-connected triangulation is always Hamiltonian [6], they then show how to transform this into the canonical triangulation by at most 2n-11 flips, using a decomposition into two outerplanar graphs that share a Hamiltonian cycle as their outer faces.

The problem of making triangulations 4-connected has also been studied in the setting where many edges may be flipped simultaneously [1]. Bose *et al.* showed that any triangulation can be made 4-connected by one simultaneous flip and that $O(\log n)$ simultaneous flips are sufficient and sometimes necessary to transform between two given triangulations.

In Section 2, we show that any triangulation can be made 4-connected using at most (3n - 6)/5 flips. This improves the first step of the construction by Mori *et al.* and results in a new upper bound on the diameter of the flip graph of 5.2n - 24.4. Then we show in Section 3 that there are triangulations that require (3n - 10)/5 flips to be made 4-connected. Since the difference with the upper bound is less than one flip, this bound is tight. Section 4 contains proofs for various technical lemmas that are necessary for the main result. Section 5 contains conclusions and future work.

^{*}School of Computer Science, Carleton University. This research was partially supported by NSERC.

[§]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya. Partially supported by projects MTM2009-07242 and Gen. Cat. DGR 2009SGR1040.

2 Upper Bound

In this section we prove an upper bound on the number of flips required to make any given triangulation 4-connected. Specifically, we show that (3n - 6)/5 flips always suffice. The proof references several technical lemmas whose proofs can be found in Section 4.

We are given a triangulation T, along with a combinatorial embedding specifying the clockwise order of edges around each vertex of T. In addition, one of the faces of T is marked as the outer face. A separating triangle D is a cycle in T of length three whose removal splits T into two non-empty connected components. We call the component that contains vertices of the outer face the *exterior* of D, and the other component the *interior* of D. A vertex in the interior of D is said to be *inside* D and likewise, a vertex in the exterior of Dis *outside* D. An edge is inside a separating triangle if one of its endpoints is inside. A separating triangle Acontains another separating triangle B if and only if the interior of B is a subgraph of the interior of A with a strictly smaller vertex set. If A contains B, A is called the *containing* triangle. A separating triangle that is contained by the largest number of separating triangles in T is called *deepest*. Since containment is transitive, a deepest separating triangle cannot contain any separating triangles, as these would have a higher number of containing triangles. Finally, we call an edge that does not belong to any separating triangle a *free edge*.

We will remove all separating triangles from T by repeatedly flipping an edge of a deepest separating triangle. This makes T 4-connected, as a triangulation is 4-connected if and only if it has no separating triangles. This technique was also used by Mori *et al.* [4], who proved the following lemma.

Lemma 1 In a triangulation with $n \ge 6$ vertices, flipping any edge of a separating triangle D will remove that separating triangle. This never introduces a new separating triangle, provided that the selected edge belongs to multiple separating triangles or none of the edges of D belong to multiple separating triangles.

Before we can prove our new upper bound, we need to prove another property of separating triangles.

Lemma 2 In a triangulation T, every vertex v of a separating triangle D is incident to at least one free edge inside D.

Proof. Consider one of the edges of D incident to v. Since D is separating, its interior cannot be empty and since D is part of T, there is a triangular face inside Dthat uses this edge. Now consider the other edge e of this face that is incident to v.

The remainder of the proof is by induction on the number of separating triangles contained in D. For the

base case, assume that D does not contain any other separating triangles. Then e must be a free edge and we are done.

For the induction step, there are two further cases. If e does not belong to a separating triangle, we are again done, so assume that e belongs to a separating triangle D'. Since D' is itself a separating triangle contained in D and containment is transitive, the number of separating triangles contained by D' must be strictly smaller than that of D. Since v is also a vertex of D', our induction hypothesis tells us that there is a free edge incident to v inside D'. Since D' is contained in D, this edge is also inside D.

Theorem 3 A triangulation on $n \ge 6$ vertices can be made 4-connected using at most $\lfloor (3n-6)/5 \rfloor$ flips.

Proof. We prove this using a charging scheme. We begin by placing a coin on every edge of the triangulation. Then we flip an edge of a deepest separating triangle (preferring edges that belong to multiple separating triangles) until no separating triangles are left. We pay 5 coins for every flip. During this process, we maintain two invariants:

- Every edge of a separating triangle has a coin.
- Every vertex of a separating triangle has an incident free edge that is inside the triangle and has a coin.

These invariants have several nice properties. First, an edge can either be a free edge or belong to a separating triangle, but not both. So at any given time, only one invariant applies to an edge. Second, an edge only needs one coin to satisfy the invariants, even if it is on multiple separating triangles or is a free edge for multiple separating triangles. These two properties imply that the invariants hold initially, since by Lemma 2, every vertex of a separating triangle has an incident free edge. Third, flipping an edge that satisfies the criteria of Lemma 1 cannot upset the invariants, since its separating triangle is removed and no new separating triangles are introduced. Finally, since we pay 5 coins per flip and there are 3n - 6 edges, by placing a coin on each edge, we can flip at most |(3n - 6)/5| edges.

Now let us take a closer look at the kind of edges we can use to pay for flipping an edge of a deepest separating triangle D. We identify four types of edges here:

Type 1 (\blacksquare). The flipped edge *e*. By Lemma 1, *e* cannot belong to any separating triangle after the flip, so the first invariant still holds if we remove *e*'s coin. Before the flip, *e* was not a free edge, so the second invariant was satisfied even without *e*'s coin. Since the flip did not introduce any new separating triangles, this is still the case.

Type 2 (\Box). A non-flipped edge *e* of *D* that is not shared with any other separating triangle. By Lemma 1, the flip removed *D* and did not introduce any new separating triangles. Therefore *e* cannot belong to any separating triangle, so the first invariant still holds if we remove *e*'s coin. By the same argument as for the previous type, *e* is also not required to have a coin to satisfy the second invariant.

Type 3 (O). A free edge e of a vertex of D that is not shared with any containing separating triangle. Since e did not belong to any separating triangle and the flip did not introduce any new ones, e is not required to have a coin to satisfy the first invariant. Further, since the flip removed D and e is not incident to a vertex of another separating triangle that contains it, it is no longer required to have a coin to satisfy the second invariant. Therefore we can remove its coin without violating either invariant.

Type 4 (•). A free edge e incident to a vertex v of D, where v is an endpoint of an edge e' of D that is shared with a non-containing separating triangle B, provided that we flip e'. Any separating triangle that contains D but not B must share e' (Lemma 10) and is therefore removed by the flip. So every separating triangle after the flip that contains D also contains B. In particular, this also holds for containing triangles that share v. Since the second invariant requires only one free edge with a coin for each vertex, we can safely charge the one inside D, as long as we do not charge the free edge in B.

To decide which edges we flip and how we pay for each flip, we distinguish five cases, based on the number of edges shared with other separating triangles and whether any of these triangles contain D. These cases are illustrated in Figures 2, 3, and 4.



Figure 2: The edges that are charged if the deepest separating triangle does not share any edges with other separating triangles. The flipped edge is dashed and the charged edges are marked with red boxes (Type 1), white boxes (Type 2), white disks (Type 3) or red disks (Type 4).

Case 1. D does not share any edges with other separating triangles (Figure 2). In this case, we flip any of D's edges. By the first invariant, each edge of D has a coin. These edges all fall into Types 1 and 2 above, so we use their coins to pay for the flip. Further, D can share at most one vertex with a containing triangle (Lemma 8),

so we charge two free edges, each incident to one of the other two vertices (Type 3).



Figure 3: The edges that are charged if the deepest separating triangle only shares edges with non-containing separating triangles.

Case 2. D does not share any edge with a containing triangle, but shares one or more edges with non-containing separating triangles (Figure 3). In this case, we flip one of the shared edges e. We charge e (Type 1) and two free edges inside D that are incident to the vertices of e (Type 4). This leaves us with two more coins that we need to charge.

Let B be the non-containing separating triangle that shares e with D. We first show that B must be deepest. There can be no separating triangles that contain D but not B, as any such triangle would have to share e (Lemma 10) and D does not share any edge with a containing triangle. Therefore any triangle that contains D must contain B as well. Since D is contained in the maximal number of separating triangles, this holds for B as well. This means that B cannot contain any separating triangles and to satisfy the second invariant we only need to concern ourselves with triangles that contain both B and D.

Now consider the number of vertices of the quadrilateral formed by B and D that can be shared with containing triangles. Since D does not share an edge with a containing triangle, it can share at most one vertex with a containing triangle (Lemma 8). Now suppose that B shares an edge with a containing triangle. Then one of the vertices of this edge is part of D as well. Since the other two vertices are both part of D, they cannot be shared with containing triangles. If Bdoes not share an edge with a containing triangle, it too can share at most one vertex with containing triangles. Thus, in both cases, at most two vertices of the quadrilateral can be shared with containing triangles and we charge two free edges, each incident to one of the other two vertices, for the last two coins (Type 3).

Case 3. D shares an edge with a containing triangle A and does not share the other edges with any separating triangle (Figure 4a). In this case, we flip the shared edge and charge all of D's edges, since one is the flipped edge (Type 1) and the others are not shared (Type 2).



Figure 4: The edges that are charged if the deepest separating triangle shares an edge with a containing triangle.

The vertex of D that is not shared with A cannot be shared with any containing triangle (Lemma 9), so we charge a free edge incident to this vertex (Type 3).

Further, if A shares an edge with a containing triangle, it either shares the flipped edge, which means that the containing triangle is removed by the flip, or it shares another edge, in which case the vertex that is not an endpoint of this edge cannot be shared with any containing triangle. If A does not share an edge with a containing triangle, it can share at most one vertex with a containing triangle (Lemma 8). In both cases, one of the vertices of the flipped edge is not shared with any containing triangle (Type 3), so we charge a free edge incident to it.

Case 4. D shares an edge with a containing triangle A and one other edge with a non-containing separating triangle B (Figure 4b). In this case, we flip the edge that is shared with B. Let v be the vertex of D that is not shared with A. We charge the flipped edge (Type 1), the unshared edge of D (Type 2) and two free edges inside D that are incident to the vertices of the flipped edge (Type 4). We charge the last coin from a free edge in B that is incident to v. We can charge it, since v cannot be shared with a triangle that contains D (Lemma 9) and every separating triangle that contains B but not D must share the flipped edge as well (Lemma 10) and is therefore removed by the flip.

All that is left is to argue that there can be no separating triangle contained in B that requires the charge to satisfy the second invariant. Every separating triangle that contains D but not B must share the flipped edge (Lemma 10). Since D already shares another edge with a containing triangle and it cannot share two edges with containing triangles (Lemma 7), all separating triangles that contain D must also contain B. Since D is deepest, B must be deepest as well and therefore cannot contain any separating triangles.

Case 5. D shares one edge with a containing triangle A and the other two with non-containing separating triangles (Figure 4c). In this case we also flip the edge

shared with one of the non-containing triangles. The charged edges are identical to the previous case, except that there is no unshared edge any more. Instead, we charge the last free edge in D.

Before we argue why we are allowed to charge it, we need to give some names. Let e be the edge of D that is not shared with A and is not flipped. Let B be the noncontaining triangle that shares e and let v be the vertex that is shared by A, B and D. Now, any separating triangle that shares v and contains D must contain Bas well. If it did not, it would have to share e with D, but D already shares an edge with a containing triangle and cannot share more (Lemma 7). Since the second invariant requires only a single charged free edge for each vertex of a separating triangle, it is enough that vstill has an incident free edge in B.

This shows that we can charge 5 coins for every flip, while maintaining the invariants. Now all that we need to show is that after performing these flips we have indeed removed all separating triangles. As long as our triangulation has a separating triangle, we can always find a deepest separating triangle D. Since D shares at most one edge with separating triangles (Lemma 7), one of the cases above must apply to D. This gives us an edge of D to flip and five edges to charge, each of which is guaranteed by the invariants to have a coin. Therefore the process stops only after all separating triangles have been removed.

Corollary 4 The diameter of the flip graph of all triangulations on n vertices is at most 5.2n - 24.4.

Proof. Mori *et al.* [4] showed that any two 4-connected triangulations can be transformed into each other by at most 4n - 22 flips. By Theorem 3, we can make a triangulation 4-connected using at most $\lfloor (3n - 6)/5 \rfloor$ flips. Hence, we can transform any triangulation into any other using at most $2 \cdot (3n - 6)/5 + 4n - 22 \leq 5.2n - 24.4$ flips.

3 Lower Bound

In this section we present a lower bound on the number of flips that are required to remove all separating triangles from a triangulation. Specifically, we present a triangulation that has (3n-10)/5 edge-disjoint separating triangles, thereby showing that there are triangulations that require this many flips to make them 4-connected.

The triangulation that gives rise to the lower bound is constructed recursively and is similar to the Sierpiński triangle [5]. The construction starts with an empty triangle. The recursive step consists of adding an inverted triangle in the interior and connecting each vertex of the new triangle to the two vertices of the opposing edge of the original triangle. This is recursively applied to the three new triangles that share an edge with the inserted triangle, but not to the inserted triangle itself. After kiterations, instead of applying the recursive step again, we add a single vertex in the interior of each triangle we are recursing on and connect this vertex to each vertex of the triangle. We also add a single vertex in the exterior face so that the original triangle becomes separating. The resulting triangulation is called \mathcal{T}_k . Figure 5 illustrates this process for \mathcal{T}_1 and \mathcal{T}_2 .



Figure 5: Triangulations \mathcal{T}_1 (a) and \mathcal{T}_2 (b), before and after the final step of the construction.

Theorem 5 There are triangulations that require $\lceil (3n-10)/5 \rceil$ flips to make them 4-connected.

Proof. In the construction scheme presented above, each of the triangles we recurse on becomes a separating

triangle that does not share any edges with the original triangle or the other triangles that we recurse on. Thus all these separating triangles are edge-disjoint. But how many of these triangles do we get? Let L_i be the number of triangles that we recurse on after *i* iterations of the construction, so $L_0 = 1$, $L_1 = 3$, etc. Now let V_i be the number of vertices of \mathcal{T}_i . We can see that $V_1 = 10$ and if we transform \mathcal{T}_1 into \mathcal{T}_2 , we have to remove each of the interior vertices added in the final step and replace them with a configuration of 6 vertices. So to get \mathcal{T}_2 , we add 5 vertices in each of the L_1 triangles. This is true in general, giving

$$V_i = V_{i-1} + 5L_{i-1} = 10 + 5\sum_{j=2}^{i} L_{j-1}$$
 (1)

Let S_i be the number of separating triangles of \mathcal{T}_i . We can see that $S_1 = 4$ and each recursive refinement of a separating triangle leaves it intact, while adding 3 new ones. Therefore

$$S_i = S_{i-1} + 3L_{i-1} = 4 + 3\sum_{j=2}^{i} L_{j-1}$$
 (2)

From Equation (1), we get that

$$\sum_{j=2}^{i} L_{j-1} = \frac{V_i - 10}{5}$$

Substituting this into Equation (2) gives

$$S_i = 4 + 3\frac{V_i - 10}{5} = \frac{3V_i - 10}{5}$$

Since each flip removes only the separating triangle that the edge belongs to, we need (3n - 10)/5 flips to make this triangulation 4-connected.

4 Lemmas and proofs

This section contains proofs for the technical lemmas used in the proof of Theorem 3. The proofs use the following result, which is proven in Lemmas 11 and 12 in the appendix.

Lemma 6 A separating triangle A contains a separating triangle B if and only if there is a vertex of B inside A.

Lemma 7 A separating triangle can share at most one edge with containing triangles.

Proof. Suppose we have a separating triangle D that shares two of its edges with separating triangles that contain it. First of all, these triangles cannot be the same, since then they would be forced to share the third

edge as well, which means that they are D. Since a triangle does not contain itself, this is a contradiction. So call one of these triangles A and call one of the triangles that shares the other edge B. Let x, y and z be the vertices of D, such that x is shared with A and B, y is shared only with A and z is shared only with B. Let vbe the vertex of B that is not shared with D.

By Lemma 6, z must be inside A, while y must be inside B, since in both cases the other two vertices of D are shared and therefore not in the interior. But this means that A contains B and B contains A. This is a contradiction, since by transitivity it would imply that the interior of A is a subgraph of itself with a strictly smaller vertex set.

Lemma 8 A separating triangle D that shares no edge with containing triangles can share at most one vertex with containing triangles.

Proof. Suppose that D shares two of its vertices with containing triangles. First, both vertices cannot be shared with the same containing triangle, since then the edge between these two vertices would also be shared. Now let A be one of the containing triangles and let B be one of the containing triangles and let B be one of the containing triangles sharing the other vertex. By Lemma 6, there must be a vertex of D inside A. So then both vertices of D that are not shared with A must be inside A, otherwise there would be an edge between the interior and the exterior of A. In particular, the vertex shared by B and D lies inside A, which means that A contains B. But the reverse is also true, so B contains A as well, which is a contradiction.

Lemma 9 A separating triangle that shares an edge with a containing triangle cannot share the unshared vertex with another containing triangle.

Proof. Suppose we have a separating triangle D = (x, y, z) that shares an edge (x, y) with a containing triangle A and the other vertex z with another containing triangle B. By Lemma 6, x and y have to be inside B, since they cannot be outside B and they cannot be shared with B by Lemma 7. Since x and y are vertices of A, this means that B contains A. Similarly, z has to be inside A and since it is a vertex of B, A contains B. This is a contradiction.

Lemma 10 Given two separating triangles A and B that share an edge e, any separating triangle that contains A but not B must use e.

Proof. Suppose that we have a separating triangle D that contains A, but not B and that does not use one of the vertices v of e. By Lemma 6, v must be inside D. But then D would also contain B, as v is a vertex of B as well. Therefore D must share both vertices of e and hence e itself.

5 Conclusions and future work

We showed that any triangulation can be made 4connected using at most $\lfloor (3n-6)/5 \rfloor$ flips, while there are triangulations that require $\lceil (3n-10)/5 \rceil$ flips. Since the difference is less than a single flip, these bounds are tight. An obvious question is how to compute the necessary flips efficiently. If we only guarantee that we use at most n-4 flips, it is possible to compute the set of edges to be flipped in O(n) time. If we want to stay below the upper bound however, we only have an algorithm that computes the set of edges used in the proof in $O(n^2)$ time.

Another interesting problem is to minimize the number of flips to make a triangulation 4-connected. We showed that our technique is worst-case optimal, but there are cases where far fewer flips would suffice. There is a natural formulation of the problem as an instance of 3-hitting set, where the subsets correspond to separating triangles and we need to pick a minimal set of edges such that we include at least one edge from every separating triangle. This gives a simple 3-approximation algorithm that picks an arbitrary separating triangle and flips all shared edges or an arbitrary edge if there are no shared edges. However, it is not clear whether the problem is NP-hard, so it might even be possible to compute the optimal sequence in polynomial time.

Our result implies a new bound of 5.2n - 24.4 on the diameter of the flip graph. It is likely that this can be reduced further. For example, all of the current algorithms use the same, single, canonical form. This leaves several interesting questions open. Is there another canonical form that gives a better upper bound? Can we gain something from using multiple canonical forms and picking the closest? And can we find or approximate the actual shortest flip path?

References

- P. Bose, J. Czyzowicz, Z. Gao, P. Morin, and D. R. Wood. Simultaneous diagonal flips in plane triangulations. J. Graph Theory, 54(4):307–330, 2007.
- [2] P. Bose and F. Hurtado. Flips in planar graphs. Comput. Geom., 42(1):60–80, 2009.
- [3] H. Komuro. The diagonal flips of triangulations on the sphere. Yokohama Math. J., 44(2):115–122, 1997.
- [4] R. Mori, A. Nakamoto, and K. Ota. Diagonal flips in Hamiltonian triangulations on the sphere. *Graphs Com*bin., 19(3):413–418, 2003.
- [5] W. Sierpiński. Sur une courbe dont tout point est un point de ramification. CR Acad. Sci. Paris, 160:302–305, 1915.
- [6] W. T. Tutte. A theorem on planar graphs. Trans. Amer. Math. Soc., 82:99–116, 1956.
- [7] K. Wagner. Bemerkungen zum vierfarbenproblem. Jahresber. Dtsch. Math.-Ver., 46:26–32, 1936.

Appendix

Lemma 11 If a separating triangle A contains a separating triangle B, then there is a vertex of B inside A and no vertex of B can lie outside A.

Proof. Let z be a vertex in the interior of B and let y be a vertex of A that is not shared with B. Since the interior of B is a subgraph of the interior of A and y is not inside A, y must be outside B. Since every triangulation is 3-connected, there is a path from z to y that stays inside A. This path connects the interior of B to the exterior, so there must be a vertex of B on the path and hence inside A.

Now suppose that there is another vertex of B outside A. Since all vertices of a triangle are connected by an edge, there is an edge between this vertex and the vertex of B inside A. This contradicts the fact that A is a separating triangle, so no such vertex can exist. \Box

Lemma 12 If a vertex x of a separating triangle B is inside a separating triangle A, then A contains B.

Proof. Let y be a vertex of A that is not shared with B. There is a path from y to the outer face that stays in the exterior of A. There can be no vertex of B on this path, since this would create an edge between the interior and exterior of A. Therefore y is outside B.

Now suppose that A does not contain B. Then there is a vertex z inside B that is not inside A. There must be a path from z to x that stays inside B. Since x is inside A, there must be a vertex of A on this path. But since y is outside B, this would create an edge between the interior and exterior of B. Therefore A must contain B.

Approximating the Medial Axis by Shooting Rays: 3D Case

Svetlana Stolpner *

Kaleem Siddiqi[†]

Sue Whitesides[‡]

Abstract

We consider an algorithm, first presented in [13], that outputs regions intersected by the medial axis of a 3D solid. In practice, this algorithm is used to approximate the medial axis with a collection of points having a desired density. The quality of the medial axis approximation is supported by experimental results. Despite promising 2D results, the algorithm's theoretical guarantees are not understood in 3D. The contribution of this article is to initiate the 3D theoretical analysis by presenting properties of medial points that are not detected by the algorithm for a finite sampling rate.

1 Introduction

Consider an orientable 3D solid Ω with boundary \mathcal{B} .

Definition 1 The medial axis \mathcal{MA} of Ω is the set of centres of maximal (for the inclusion order) inscribed balls in Ω .

For example, Figure 1 shows the medial axis of a box. Figure 3 shows subsets of medial axes of more complex inputs. The medial axis, introduced in [3], is a valuable shape descriptor with applications to computer vision, computer graphics, GIS and robotics [11], as it captures local width information and part structure. Computing an accurate, robust, and useful shape descriptor based on the medial axis for complex 3D inputs remains a subject of ongoing research. In this article, we study the theoretical properties of an algorithm for medial axis approximation, which is shown to be successful at generating qualitatively meaningful approximations in practice in [14]. The following definition will be central:

Definition 2 The Euclidean distance transform of Ω is given by $D(p) = -\inf_{q \in \mathcal{B}} d(p,q)$, where $p \in \Omega$ and d(p,q) denotes Euclidean distance.

The gradient of $D, \nabla D : \mathbb{R}^3 \to \mathbb{R}^3$ is a vector field that assigns each point p the direction to its nearest point on \mathcal{B} , whenever this direction is uniquely defined. The vector field ∇D is uniquely defined for all points inside Ω except for those on the medial axis. As medial points have two or more nearest boundary locations, ∇D is multi-valued on the medial axis. This property is the basis for Algorithms 1 and 2 that locate medial points: we will look for regions where ∇D is multi-valued.

Given a medial point $m \in \mathcal{MA}$, equidistant from exactly two boundary points, the directions to its nearest boundary points are called the *spoke vectors*. The angle between the two spoke vectors is twice the *object an*gle of m. The object angle θ in Figure 1 is $\pi/4$. The distance from m to \mathcal{B} is the radius of m. Object angle



Figure 1: The medial axis of a box.

and radius are popular measures used to guide pruning of "insignificant" medial points [6, 8, 2, 12].

Previous Work When Ω is a polyhedron, [5] computes the edges of the medial axis of a polyhedron with a small number of faces using exact arithmetic. For objects whose boundary is given as a set of points, [1, 6] approximate the medial axis using a subset of the Voronoi vertices of the set of boundary points, and show convergence for a sufficient sampling density. However, these methods are very sensitive to noise in the boundary samples, and numerous techniques have been proposed to prune undesirable portions of the computed medial axes [15, 9]. Methods [7, 16, 4] recursively subdivide space and consider the nearest boundary elements to the spatial regions. Accuracy is guaranteed when approximating the generalized Voronoi diagram, where the diagram is localized by for a sufficiently small spatial resolution. The average outward flux of ∇D in a region shrinking to zero is used to decide the presence of medial points in [10]; this concept is generalized to non-zero regions for polyhedral inputs in [12]. Several methods consider angles between ∇D vectors for pairs of points p, q and conclude that a medial point exists on the midpoint of (p, q) if the two vectors' tails are closer than their tips [17, 8].

Organization and Main Results Section 2 reviews algorithms that analyze ∇D vectors for points sampled

^{*}School of Computer Science and Centre for Intelligent Machines, McGill University, sveta@cim.mcgill.ca

[†]School of Computer Science and Centre for Intelligent Machines, McGill University, siddiqi@cim.mcgill.ca

[‡]Department of Computer Science, University of Victoria, sue@uvic.ca.

on the boundary of a sphere to determine if the sphere contains a medial point, and if so, its approximate location. We include experimental results that show collections of medial points computed using this method, where the density of medial points is user-prescribed. Our main results are found in Section 3, which discusses the positions of points to be sampled on the sphere, and Section 4, which establishes the locations of nearest boundary points to medial points that are not detected by our algorithm for a finite sampling rate. Section 5 presents avenues for future work.

2 Shooting Rays Algorithm

Our algorithm for medial axis approximation is based on the following property of ∇D :

Lemma 1 ([13]) Let p be a point in Ω that is not a medial point. Let $q = p + \gamma \cdot \nabla D(p)$, such that γ is a scalar, q is not a medial point, and (p,q) lies inside Ω . A medial point of Ω lies on (p,q) if and only if $\nabla D(p) \neq \nabla D(q)$.

Consider a point p on a sphere S. Let l be a line through p with direction $\nabla D(p)$. Define the *opposite* of p, opp(p), to be the other point of intersection of lwith the surface of S. In case $\nabla D(p)$ is tangent to Sat p, opp(p) = p. Algorithm 1 uses a technique we call shooting rays to conclude that a sphere is intersected by the medial axis, when evidence of this is found. If a medial point lies in S, Algorithm 1 will necessarily return 'True' for a sufficiently dense set of points Φ . However, for a finite Φ , a medial point may lie in S, while Algorithm 1 returns 'Undecided'.

Algorithm 1 DECIDEMA(\mathcal{B}, S, Φ)

- **Require:** Boundary \mathcal{B} , sphere S, not intersecting \mathcal{B} , set of points Φ distributed on S.
- **Ensure:** 'True' if S contains a medial point, 'Undetermined' if no such conclusion can be drawn.

```
1: for all \phi_i \in \Phi do
```

- 2: **if** ϕ_i or $opp(\phi_i)$ is a medial point **then**
- 3: Return 'True'
- 4: **end if**
- 5: **if** $\nabla D(\phi_i) \neq \nabla D(opp(\phi_i))$ **then**
- 6: Return 'True'
- 7: end if
- 8: end for
- 9: Return 'Undetermined'

Algorithm 2 performs binary search to estimate the intersection of the medial axis with a line segment to a desired accuracy ϵ . As discussed in [14], we have successfully used Algorithm 2 to detect medial points for polyhedral inputs. In our implementation, the interior of a polyhedron is partitioned into regular sized

Algorithm 2 RETRACT $(p, q, \mathcal{B}, \epsilon)$ **Require:** Non-medial points p, q interior to \mathcal{B} s.t. q = $p + \gamma \cdot \nabla D(p)$ and $\nabla D(p) \neq \nabla D(q)$, tolerance ϵ . **Ensure:** A point within ϵ of the medial axis of \mathcal{B} . 1: while $d(p,q) > \epsilon$ do $m = \frac{1}{2}(p+q)$ 2: 3: if m is a medial point then 4: Return m. end if 5: if $\nabla D(m) \neq \nabla D(p)$ then 6: 7:q = melse 8: p = m9: 10: end if 11: end while

12: Return p.

cubes (voxels) and a sphere is circumscribed about each voxel. For those spheres deemed intersected by the medial axis, we compute the approximate locations of a medial point inside this sphere using Algorithm 2. Among the approximate medial points found in a sphere circumscribed about voxel v, we store a single point that lies inside v and has a sufficiently high object angle. The voxel size determines the density of the computed set of medial points. Figure 3 presents examples of the medial points computed by our method for several polyhedra of significant geometric complexity. Figure 2 shows the effect of varying the voxel size on the density of medial points computed.



Figure 3: Points on the medial axis of three solids with triangle mesh boundaries computed with our method, described in [14]. The object angle threshold used is 0.6 radians.



Figure 2: Medial points computed for the same solid with decreasing voxel size.

Ideally, if Algorithm 1 returns 'Undetermined' and a medial point m lies in S, m should not be a significant medial point, such as one of high object angle and radius. In [13], we described an algorithm based on an analysis of ∇D vectors in 2D and showed that the medial points missed by the algorithm become less significant as the density of samples on a circle increases. However, designing effective tools for detecting medial points in the 3D case is challenging. The next section describes a situation in which a significant medial point lies in S, while Algorithm 1 returns 'Undetermined'.

3 Deep Samples

Suppose that DECIDEMA(\mathcal{B}, S, Φ) returns 'Undetermined'. It may happen that none of the line segments $(\phi, opp(\phi))$ is long enough to penetrate deeply into Sand none intersects the medial axis. As a result, it is possible to fail to detect medial points in S, as shown in Figure 4. The medial points missed in this example are of the highest object angle possible ($\pi/2$ for the medial point at the sphere centre). Further, as the radius of Scan be chosen to be arbitrarily large, the medial points missed have arbitrarily large radius.

In order to improve the ability of Algorithm 1 to detect significant medial points, we propose to consider two additional query points c_{in} and c_{out} , defined as follows. Let the centre of S be c. Let the nearest point on the boundary \mathcal{B} to c be C, which is outside Sby the assumption that Sdoes not intersect \mathcal{B} . Define $c_{in}, c_{out} \in S$, where c_{out} is the intersection of S and the ray at c with direction (c, C) and c_{in} is the intersection of S and the ray at c with direction (C, c). Line



Figure 4: When \mathcal{B} consists of two points outside the sphere, the medial axis is shown as a dashed line. Points Φ are big dots on the sphere.

segment (c_{in}, c_{out}) is the longest line segment possible connecting a pair of points on S. In the example in Figure 4, $\nabla D(c_{in}) \neq \nabla D(c_{out})$. Therefore, in this example, by including c_{in} and c_{out} among the sampled points on S, we are guaranteed to detect a medial axis point in S. If we still do not detect a medial point in S, Lemma 2 characterizes where the set of nearest boundary points to points sampled on S must lie.

In the proof of the following lemma, we use B(a, A) to denote a closed ball centred at point a and having point A on its boundary. Let $\Theta = \Phi \bigcup \{opp(\phi_i) | \phi_i \in \Phi\}$ be the set of all sampled points considered on S.

Lemma 2 If $\nabla D(c_{in}) = \nabla D(c_{out})$ and DECIDEMA (\mathcal{B}, S, Φ) returns 'Undetermined', then all the nearest points on \mathcal{B} to points in Θ lie above the plane π through c_{in} with normal (c, C).

Proof. Consider point $p \in \Theta$ whose nearest boundary point is P. Consider the quantity $(P-p) \cdot N_S(p)$, where $N_S(p)$ is the outer normal to S at p. If $(P-p) \cdot N_S(p) >$ 0, let p be opp(p). Then $(P-p) \cdot N_S(p) \leq 0$.

The nearest point on \mathcal{B} to p, P, is inside or on the ball $B_p = B(p,C)$ and outside or on the ball $B_{c_{in}} = B(c_{in},C)$. Refer to Figure 5. Consider the plane of intersection of $B_{c_{in}}$ and B_p , π_1 . Consider also the tangent plane to S at p, π_2 . Consider the plane ρ passing through the points p, c_{in}, c_{out} . Plane ρ is orthogonal to planes π , π_1 and π_2 .

Consider the orthogonal projection of P into ρ . Let (c_{in}, c_{out}) be vertical in ρ . Then P's orthogonal projection lies in the half-plane left of $\pi_1 \cap \rho$ and in the half-plane bounded by $\pi_2 \cap \rho$ containing c. Let p' be the intersection of planes ρ , π_1 and π_2 . We will show that p' lies above π , and hence, P lies above π . Consider the line l through p and c_{in} . Note that $\angle c_{out}pc_{in} = \pi/2$ and $\angle Cpc_{in} > \pi/2$, since C is outside S. Let p'' be the intersection of the line l with π_1 . Since $\angle Cpc_{in} > \pi/2$ and l is orthogonal to π_1 , p'' is left of p on l. Hence, p'', just like p, is above π . Since π_2 is tangent to S at p and since p'' is left of p on l, p' is above π .

Lemma 2 explains how using the sample points c_{in}



Figure 5: Side view at the objects of interest in the proof of Lemma 2.

and c_{out} restricts the situations where Algorithm 1 returns 'Undetermined'. The next section explains how the set of all possible locations of the two nearest boundary points to a medial point missed by Algorithm 1 can be computed.

4 Nearest Boundary Points to Missed Medial Points

Suppose that Algorithm 1, DECIDEMA(\mathcal{B}, S, Φ), returns 'Undetermined'. Consider the convex hull of the points $\Theta = \Phi \bigcup \{opp(\phi_i) | \phi_i \in \Phi\}, CH(\Theta)$. Suppose that there is a medial point *m* inside $CH(\Theta)$. We would like to know the locations of *m*'s nearest points on \mathcal{B} .

Recall that $B_a = (a, A)$ is a closed ball with centre a having point A on its boundary and let d(a, b) be the Euclidean distance between points a and b. The following tool will prove helpful in locating the nearest boundary points to m:

Lemma 3 Consider two closed balls $B_a = B(a, Y)$ and $B_b = B(b, Y)$. Then for any ball $B_c = B(c, Y)$, $c \in (a, b)$, $B_c \subseteq B_a \cup B_b$.

Proof. ¹ Let x be the intersection of line segment (a, b) with $B_a \cap B_b$ (a disk). Let I be the boundary of $B_a \cap B_b$ (a circle). We want to show that the distance from c to I is less than or equal to d(c, Y). Let $X \in I$. Then $d(c, X)^2 = d(c, x)^2 + d(x, X)^2$. Also $d(c, Y)^2 = d(c, x)^2 + d(x, Y)^2$. However, note that $d(x, X)^2 = d(x, Y)^2$, since X and Y both lie on the circle I with centre x. It follows that $d(c, Y)^2 = d(c, X)^2 = d(c, X)^2$ and d(c, X) = d(c, Y). Thus, B_c is contained in the union of B_a and B_b .

Let \mathbb{B} be the set of closest points on \mathcal{B} to Θ . Let $N = |\Phi|$. We will assume that there are exactly N

distinct points in \mathbb{B} (this holds when the boundary \mathcal{B} is C^1). We now explain how to construct a region of \mathbb{R}^3 that contains all the possible nearest boundary points to a medial point m inside $CH(\Theta)$. This region will be found by subtracting the "empty foam" from the "full foam", which we define and explain how to compute in the following discussion.

Empty Foam For each point $p \in \Theta$, if $P \in \mathbb{B}$ is the nearest boundary point to p, then ball $B_p = (p, P)$ has an empty interior and the only point on its boundary is P. Let $\mathcal{F}_e = \bigcup B_p \setminus \mathbb{B}$ be the union of balls hereafter called the *empty foam*.

Full Foam Consider the Voronoi diagram of \mathbb{B} , $VD(\mathbb{B})$. Since m is a medial point, it is not one of the points in Θ . Suppose that m is in A's Voronoi region, $V(A), A \in \mathbb{B}$. Then m's nearest point on \mathcal{B} is no further than d(m, A), *i.e.* its nearest boundary point is on or inside the ball $B_m = (m, A)$. Using the information about A's Voronoi neighbours, we will find the region of space that contains B_m . This region of space will be a union of balls, which we call the *full foam of* A, \mathcal{F}_f^A . The set $\mathcal{F}_f = \{\bigcup \mathcal{F}_f^P | P \in \mathbb{B}\}$ is called the *full foam*.

We now explain how the full foam of A can be computed.

Let $\{a, opp(a)\} \subset \Theta$ be the points in Θ that have Aas their nearest boundary point. Let a' be the nearest point on the line segment (a, opp(a)) to m. It can be easily shown that the nearest boundary point to a' is A. Consider the ray at a' with direction (a', m). Let m'be the intersection of this ray with either the boundary of V(A), or $CH(\Theta)$, whichever occurs first. Let $B_{m'} = B(m', A)$. Let $B_{a'} = B(a', A)$. By Lemma 3, $B_m \subset B_{a'} \cup B_{m'}$. Let $B_a = B(a, A)$ and $B_{opp(a)} =$ B(opp(a), A). By Lemma 3, $B_{a'} \subset B_a \cup B_{opp(a)}$. We add B_a and $B_{opp(a)}$ to \mathcal{F}_f^A and now proceed to find spheres that contain $B_{m'}$.

There are several cases: (1) m' is on a Voronoi face, (2) m' is on a Voronoi edge, (3) m' is on a Voronoi vertex, or (4) m' is on $CH(\Theta)$. We consider each case in turn.

Case 1. Suppose that the Voronoi face is a bisector of points A and B in \mathbb{B} . Let bis(A, B) denote the bisector of points A and B. It is a plane. Consider a plane at m' with normal direction (a, A). This plane necessarily intersects bis(A, B) as (a', m) is necessarily not parallel to (a, A) and the intersection is a line on bis(A, B)passing through m'. By following this line, we will find two points m_1^* and m_2^* , where each point either lies on an edge of V(A) (Case 2), a vertex of V(A) (Case 3), or on the $CH(\Theta)$ (Case 4). Define $B_{m_1^*} = B(m_1^*, A)$ and $B_{m_2^*} = B(m_2^*, A)$. Then $B_{m'} \subset B_{m_1^*} \cup B_{m_2^*}$ by Lemma 3. We now proceed to the respective cases to find balls containing $B_{m_1^*}$ and $B_{m_2^*}$.

Case 2. Suppose that the Voronoi edge of V(A) is a trisector of points A, B and C in \mathbb{B} . Starting from a

 $^{^{1}}$ We thank Nina Amenta for the idea behind this proof.


Figure 6: If a medial point m is in the convex hull of the 6 sampled points on the boundary of the dark disk with nearest boundary points A, B, and C, then the nearest boundary points to m are inside the green disk and outside the grey disks. The dashed lines are the bisectors of A, B and C.

point m^* on the edge, we will move up and down this edge until either we hit a Voronoi vertex of V(A) (Case 3), or we hit the convex hull of Θ (Case 4) at points v_1 and v_2 . Then $m^* \in (v_1, v_2)$. Let $B_{v_1} = B(v_1, A)$ and $B_{v_2} = B(v_2, A)$. Then $B_{m^*} = B(m^*, A)$ is contained in $B_{v_1} \cup B_{v_2}$ by Lemma 3. We add B_{v_1} and B_{v_2} to the full foam of $A \mathcal{F}_f^A$.

Case 3. Any Voronoi vertex v of V(A) inside $CH(\Theta)$ defines a ball $B_v = B(v, A)$ which we add to the full foam of $A \mathcal{F}_f^A$.

Case 4. In this case, $m' \in CH(\Theta)$. Suppose, for a contradiction, that m' is a vertex of $CH(\Theta)$. This vertex cannot be *a* or opp(a) because we reached it by following the direction (a', m) from *a'*. Any other point in Θ is outside of V(A), and so is this vertex. But then we would have hit the boundary of V(A) before hitting this vertex when following the ray (a', m) from *a'*. Therefore, *m'* lies on an edge of $CH(\Theta)$ (Case 5) or on the interior of some triangle of $CH(\Theta)$ (Case 6).

Case 5. In this case, point $m' \in V(A)$ lies on an edge e of $CH(\Theta)$. In case e is (a, opp(a)), then $B_{m'} = B(m', A)$ is contained in $B_a = B(a, A)$ and $B_{opp(a)} = B(opp(a), A)$ and these balls have already been added to \mathcal{F}_f^A . Suppose edge e is (a, b) or (opp(a), b) for some point $b \in \Theta$ outside V(A). Then V(A) intersects (a, b) at some point x. By Lemma 3, either $B_{m'} \subset B_a \cup B_x$ or $B_{m'} \subset B_{opp(a)} \cup B_x$, where $B_x = B(x, A)$. In this case, we add B_x and either B_a or $B_{opp(a)}$ to \mathcal{F}_f^A . Now suppose edge e is (b, c), which is intersected by V(A), for some pair of points $b, c \in \Theta$ outside of V(A). In

this case there are two points v_1 and v_2 on (b,c) that are the intersections of V(A) with (b,c), such that $m' \in$ (v_1, v_2) . If $B_{v_1} = B(v_1, A)$ and $B_{v_2} = B(v_2, A)$, then $B_{m'} \subset B_{v_1} \cup B_{v_2}$. We add B_{v_1} and B_{v_2} to \mathcal{F}_f^A .

Case 6. In this case, point $m' \in V(A)$ lies on the interior of a triangle t of $CH(\Theta)$. At least one vertex of triangle t is a or opp(a). Suppose it is a. Then by following direction (a, m'), we will hit either (6-1) an edge of t at point m'', or (6-2) the boundary of V(A)at point m''. Ball $B_{m'} = B(m', A)$ is contained in $B_a =$ B(a, A) and $B_{m''} = B(m'', A)$. In case 6-1, we proceed to Case 5 for point m'' (recalling that B_a is already in \mathcal{F}_{f}^{A}). In case 6-2, if m'' is on an edge or vertex of V(A)and we add $B_{m''}$ to \mathcal{F}_f^A (recalling that B_a is already in \mathcal{F}_{f}^{A}). Otherwise, if m'' is on a face of V(A), then the intersection of this face and t is a line segment (v_1, v_2) , where v_1 and v_2 are either on a Voronoi edge or vertex, or on an edge of t. If we define $B_{v_1} = B(v_1, A)$ and $B_{v_2} = B(v_2, A)$, then $B_{m'} \subset B_{v_1} \cup B_{v_2}$. In this case, we add B_{v_1} and B_{v_2} to \mathcal{F}_f^A (recalling that B_a is already in \mathcal{F}_{f}^{A}).

In this argument, for a medial point m in $CH(\Theta)|V(A)$, we have added balls to \mathcal{F}_f^A passing through A centred at the following types of points q:

- Type 1: $q \in (a, opp(a))$
- **Type 2:** q is a vertex of V(A) inside or on $CH(\Theta)$
- **Type 3:** q is an intersection of an edge of V(A) with CH(Θ)
- **Type 4:** q is an intersection of a face of V(A) with edges of $CH(\Theta)$.

By the argument above, which uses multiple invocations of Lemma 3 to create a set of spheres that contain $B_m = B(m, A)$ for an arbitrarily positioned $m \in CH(\Theta)|V(A)$, it follows that $B_m \subset \mathcal{F}_f^A$. Starting with an arbitrary point $m \in V(A)$, we can construct the full foam of A by taking the union of the four types of balls described above.

The union of the full foams of each boundary point $P \in \mathbb{B}$ gives the full foam: $\mathcal{F}_f = \{\bigcup \mathcal{F}_f^P | P \in \mathbb{B}\}$. Recall that the empty foam is $\mathcal{F}_e = \{\bigcup B_p | p \in \Theta \setminus \mathbb{B}\}$, where $B_p = B(p, P)$, and $P \in \mathbb{B}$ is the nearest boundary point to p. The region \mathcal{F}_e does not contain any points in \mathcal{B} . We have shown the following lemma:

Lemma 4 Suppose that DECIDEMA(\mathcal{B}, S, Θ) returned 'Undetermined'. Let \mathbb{B} be the set of nearest boundary points to Φ and let $\Theta = \Phi \bigcup \{opp(\phi_i) | \phi_i \in \Phi\}$. If there are exactly $|\Phi|$ distinct points in \mathbb{B} , then for any medial point $m \in CH(\Theta)$, its nearest boundary points lie in $\mathcal{F}_f \setminus \mathcal{F}_e$.

Observe that when computing the full foam, we need only consider the vertices of $VD(\mathbb{B})$ inside or on $CH(\Theta)$,

the intersections of edges of $VD(\mathbb{B})$ with $CH(\Theta)$, and the intersection of faces of $VD(\mathbb{B})$ with edges of $CH(\Theta)$.

We can easily compute all the potential nearest boundary points to a medial point m in $CH(\Theta)$ by finding the intersection of the boundary \mathcal{B} with all the balls of the full foam of type 1-4. The quality of the approximation can be measured as the maximum distance between pairs of boundary points to missed medial points in $CH(\Theta)$, which is the maximum of the maximum distance between pairs of points in \mathcal{F}_f^A for all $A \in \mathbb{B}$.

5 Conclusions and Future Work

We have considered algorithms, based on the analysis of the gradient of the Euclidean distance transform on a sphere, that compute points within a user-chosen distance from the medial axis of an arbitrary 3D solid. Our experimental results on complex polyhedral inputs demonstrate that such an analysis of the ∇D vector field has a clear practical utility for the approximation of the medial axis. The contribution of this article has been to initiate the study of the quality of the approximation offered by such algorithms by establishing properties of the nearest boundary points to those medial points that are not detected. Much remains to be done in order to understand what theoretical guarantees can be offered by a method that studies a finite set of ∇D vectors in a fixed-radius spherical region in order to decide if this region is intersected by the medial axis in 3D and higher dimensions. Below we list several open problems. Medial point quality may be measured using either object angle, radius, or distance from the medial point to the query region.

- 1. Show that if DECIDEMA(\mathcal{B}, S, Φ) returns 'Undetermined', the quality of medial points present in sphere S decreases as the density of Φ increases.
- 2. Suppose that DECIDEMA(\mathcal{B}, S, Φ) returns 'Undetermined'. For each sphere in \mathcal{F}_f , we add its centre to the set of query points Φ to create a set of query points Φ' . Next, execute DECIDEMA(\mathcal{B}, S, Φ'). By carrying out this operation repeatedly, what can be said about the quality of the missed medial points as more iterations are considered?
- 3. Design other rules based on the analysis of a finite number of nearest boundary points to query points on a sphere that enable detection of medial points inside the sphere, such that the quality of the missed medial points can be shown to decrease by increasing the density of query points used, for a fixed-size query region S.

- N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2-3):127–153, 2001.
- [2] D. Attali and J.-O. Lachaud. Delaunay conforming isosurface, skeleton extraction and noise removal. Computational Geometry Theory and Applications, 19(2-3):175–189, 2001.
- [3] H. Blum. Biological shape and visual science. Journal of Theoretical Biology, 38:205–287, 1973.
- [4] I. Boada, N. Coll, N. Madern, and J. A. Sellarès. Approximations of 2D and 3D generalized Voronoi diagrams. *Computer Mathematics*, 85(7):1003–1022, 2008.
- [5] T. Culver, J. Keyser, and D. Manocha. Exact computation of the medial axis of a polyhedron. *Computer Aided Geometric Design*, 21(1):65–98, 2004.
- [6] T. K. Dey and W. Zhao. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica*, 38:387–398, 2004.
- [7] M. Etzion and A. Rappoport. Computing Voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry: Theory and Applications*, 21:87–120, 2002.
- [8] M. Foskey, M. C. Lin, and D. Manocha. Efficient computation of a simplified medial axis. In *Solid Modeling* and *Applications*, pages 96–107, 2003.
- [9] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3D geometry. In SIGGRAPH, 2010.
- [10] K. Siddiqi, S. Bouix, A. R. Tannenbaum, and S. W. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215– 231, 2002.
- [11] K. Siddiqi and S. Pizer, editors. Medial representations: mathematics, algorithms and applications. Springer, 2008.
- [12] S. Stolpner and K. Siddiqi. Revealing significant medial structure in polyhedral meshes. In *3DPVT*, pages 365– 372, 2006.
- [13] S. Stolpner and S. Whitesides. Medial axis approximation with bounded error. In *International Symposium* on Voronoi Diagrams, pages 171–180, 2009.
- [14] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci for 3D shape representation. *CVIU*, 115:695– 706, 2011.
- [15] R. Tam and W. Heidrich. Shape simplification based on the medial axis transform. In *Visualization*, page 63, 2003.
- [16] J. M. Vleugels and M. H. Overmars. Approximating generalized Voronoi diagrams in any dimension. Technical Report UU-CS-1995-14, Utrecht University, 1995.
- [17] Y. Yang, O. Brock, and R. N. Moll. Efficient and robust computation of an approximated medial axis. In *Solid Modeling and Applications*, pages 15–24, 2004.

An Incremental Algorithm for High Order Maximum Voronoi Diagram Construction

Khuong Vu *

Rong Zheng*

Abstract

We propose an incremental approach to compute the order-k maximum Voronoi diagram of disks in the plane. In our approach, we start with an order-k Voronoi diagram of disk centers and iteratively expand disks and update the changes of the diagram until all disks reach their targeted size. When disks expand continuously, the structure of the diagram changes discretely. The algorithm takes $O\left(\left\lceil \frac{r_{\max}-r_{\min}}{d_{\min}} \right\rceil k^2 N \log N\right)$ time complexity, where N, r_{\max} and r_{\min} are respectively the number of disks, the maximum and minimum radii of disks, and d_{\min} is the minimum distance between two disk centers. Our algorithm is amiable to distributed implementation.

1 Introduction

Consider a set of N disks $S = \{\mathcal{D}_1(o_1, r_1), \mathcal{D}_2(o_2, r_2), \mathcal{D}_2(o_2, r_2), \mathcal{D}_2(o_3, r_3), \mathcal{D}_3(o_3, r_3$ $\ldots, \mathcal{D}_n(o_N, r_N)$, where o_i and r_i are respectively the center and radius of disk \mathcal{D}_i $(1 \leq i \leq N)$. We define the distance from a point p to disk \mathcal{D}_i as $d_{\max}(p, \mathcal{D}_i) = d(p, \mathcal{D}_i)$ o_i) + r_i , where $d(\cdot, \cdot)$ is the Euclidean distance between two points. The locus of points closer to \mathcal{D}_i than to \mathcal{D}_i , $h(\mathcal{D}_i, \mathcal{D}_i)$, is one of the half-planes determined by the bisector $b(\mathcal{D}_i, \mathcal{D}_j) = \{p | d_{\max}(p, \mathcal{D}_i) = d_{\max}(p, \mathcal{D}_j)\}, \text{ or }$ $b(\mathcal{D}_i, \mathcal{D}_j) = \{p | d(p, o_i) - d(p, o_j) = r_j - r_i\}$. In general, $b(\mathcal{D}_i, \mathcal{D}_j)$ is a hyperbolic curve. Let $\mathcal{V}(\mathcal{D}_i)$ denote the locus of points closer to \mathcal{D}_i than to any other disk in S. Thus, $\mathcal{V}(\mathcal{D}_i) = \bigcap_{i \neq j} h(\mathcal{D}_i, \mathcal{D}_j)$. It has been shown in [4] that if \mathcal{D}_i does not contain any other disks, then $\mathcal{V}(\mathcal{D}_i) \neq \emptyset$. In addition, $\mathcal{V}(\mathcal{D}_i)$'s boundary consists of edges, which are hyperbolic segments, and vertices, which are intersections of adjacent edges. $\mathcal{V}(\mathcal{D}_i)$ is referred to as the Voronoi face associated with disk \mathcal{D}_i , and the set $\{\mathcal{V}(\mathcal{D}_i), 1 \leq i \leq n\}$ is referred to as the maximum Voronoi diagram, or max VD for short, of S. In [4], the authors proposed an algorithm to construct the max VD of n disks in $O(T(N) + N \log N)$, where T(N) is the time of the nearest neighbor query under $d_{\rm max}$ metric. An example of max VD is shown in Figure 1. As seen in the figure, $\mathcal{V}(\mathcal{D}_8) = \emptyset$ since it contains \mathcal{D}_6 .



Figure 1: The max Voronoi diagram of 8 disks.

Similar to the high order Voronoi diagram of points first studied by Lee [8], we generalize the concept of max VD such that a Voronoi face is associated with a set of disks, $H \subset S$ for |H| > 1. Denote $\mathcal{V}^k(H, S)$, where $|H| = k, H \subset S$, the locus of points closer to all disks of H than to any disk in $S \setminus H$. We define the order-k max VD of $S, V^k(S)$, as a collection of Voronoi faces corresponding to all subsets H of S (|H| = k), i.e., $V^k(S) = \bigcup_{H \subset S} \mathcal{V}^k(H, S), |H| = k$. We adopt the definition in [3] to formally define the order-k max VD as follows:

Definition 1 For $i, j \in S$, let $D(\mathcal{D}_i, \mathcal{D}_j) = \{p | d_{\max}(p, \mathcal{D}_i) < d_{\max}(p, \mathcal{D}_j)\}$. Let $H \subset S, |H| = k$. We define

$$\mathcal{V}^k(H,S) = \bigcap_{h \in H, i \notin H} D(\mathcal{D}_h, \mathcal{D}_i)$$

the order-k maximum-Voronoi face of a set of disks Hwith respect to S. The order-k maximum-Voronoi diagram of S is defined as

$$V^k(S) = \bigcup_{H,H' \subset S; H \neq H'; |H| = |H'| = k} \mathcal{V}^k(H,S) \bigcap \mathcal{V}^k(H',S)$$

In [10], Lee's incremental algorithm is applied to construct order-k max VD under the assumption that no disk contains any other disk. Accordingly, each Voronoi face of order-(k - 1), $\mathcal{V}^{k-1}(H, S)$, is tessellated by the order-1 Voronoi diagram of some disks to create the next order Voronoi faces. It has been shown that only disks associated with edges of $\mathcal{V}^k(H, S)$ need to be considered for tessellation. In general placement of disks, the

^{*}Department of Computer Science, University of Houston, {khuong.vu, rzheng}@cs.uh.edu



(a) order-1 max VD. The (b) order-2 max VD. The curve is the bisector of disks \mathcal{D}_1 and \mathcal{D}_2 . $\mathcal{V}(\mathcal{D}_3) = \emptyset$. \mathcal{D}_1 and \mathcal{D}_3 . $\mathcal{V}(\mathcal{D}_1, \mathcal{D}_2) \neq \emptyset$, $\mathcal{V}(\mathcal{D}_2, \mathcal{D}_3) \neq \emptyset$.

Figure 2: Incremental construction does not apply as disks are contained inside other ones.

assumption does not always hold. As illustrated in Figure 2, although \mathcal{D}_3 is associated with no edge in the order-1 max VD of $S = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$, we have $\mathcal{V}^2(\{\mathcal{D}_2, \mathcal{D}_3\}, S) \neq \emptyset$.

In the paper, we propose an incremental algorithm to construct order- $k \max VD$ of disks. The intuition is as follows. Consider a Voronoi region, $\mathcal{V}^k(H,S)$, in an order-k max VD whose edge set is E. The generation of a new edge in E or the disappearance of an existing edge in E is referred to as an *event* of E. We observe that changing a disk's radius continuously makes some Voronoi vertices move along an identifiable trajectory while the others do not change. More importantly, disks' expansion does not necessarily induce an event of E. A disk may expand *continuously* but events happen discretely. This is illustrated in Figure 3. There are two kinds of vertices, i.e., new and old, denoted by circles and solid squares, respectively. As \mathcal{D}_3 expands, vertices of both kinds move along particular edges (arrows in the figures). We observe that solid squares move *away* their corresponding opposite vertex, while circles move toward their corresponding opposite vertex. Vertices may meet while moving. In this case, they may "destroy" an edge and create another one simultaneously. Additionally, a face may degenerate due to the meeting of moving vertices. Another face may be born simultaneously. In the following sections, we provide details of events that happen when a disk expands.

We discuss the changes of the diagram as disks expand in Section 3. Then, we propose an incremental algorithm to construct the order-k maximum Voronoi diagrams in Section 4. We conclude the paper in Section 5. We next introduce the concepts of order-k max VD in Section 2.

2 Preliminary

In the following discussion, we assume that no more than 2 disks' centers are co-linear, and no point in the plane is equal-distant to more than 3 disks under the d_{max} metric. We summarize the notations used throughout the paper as follows:

- S: The set of N disks $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$.
- S_i^{ϵ} : The modified S, in which the radius of \mathcal{D}_i expands by a positive amount ϵ . $S_i^{\epsilon} = (S \setminus \{\mathcal{D}_i\}) \bigcup \{\mathcal{D}_{i'}(o_i, r_i + \epsilon)\}.$
- H: A subset of S.
- H_i^{ϵ} : The updated H. $H_i^{\epsilon} = (H \setminus \{\mathcal{D}_i\}) \bigcup \{\mathcal{D}_{i'}(o_i, r_i + \epsilon)\}.$
- $d_{\max}(p, \mathcal{D}_i)$: The maximum distance from p to \mathcal{D}_i . $d_{\max}(p, \mathcal{D}_i) = d(p, o_i) + r_i$, where $d(\cdot, \cdot)$ is the Euclidean distance between two 2D points.
- $\mathcal{V}_k(S)$: The max Voronoi face corresponding to disk \mathcal{D}_k in the max VD of S. We simply refer to this as \mathcal{V}_k , when no confusion occurs.
- V(S): The max VD of the disk set S.
- $\mathcal{V}^k(H, S)$: An order-k max Voronoi face associated with H, where |H| = k.
- $V^k(S)$: The order-k max VD of S, or "diagram" for short. When $k = 1, V^k(S) \equiv V(S)$.
- $v_{i,j,h}$: The max VD vertex corresponding to disks \mathcal{D}_i , \mathcal{D}_j , and \mathcal{D}_h .
- $e_{i,j}$: The edge of the max VD corresponding to disks \mathcal{D}_i and \mathcal{D}_j . $e_{i,j}$ is a hyperbola segment or an infinite hyperbola.
- $b_{i,j}$: The locus of points p such that $d_{\max}(p, \mathcal{D}_i) = d_{\max}(p, \mathcal{D}_j)$. $b_{i,j}$ is a hyperbola with foci being o_i and o_j , or a straight line when $r_i = r_j$. We refer to $b_{i,j}$ as the bisector of o_i and o_j .

Two circles $\mathcal{D}_1(o_1, r_1)$ and $\mathcal{D}_2(o_2, r_2)$ are internally tangent if $d(o_1, o_2) = |r_1 - r_2|$. In the rest of the paper, by stating that a circle \mathcal{D}_1 is internally tangent to \mathcal{D}_2 , we mean \mathcal{D}_2 lies interior to \mathcal{D}_1 unless stated otherwise. In addition, we say a circle \mathcal{D}_1 contains \mathcal{D}_2 if \mathcal{D}_2 lies interior to \mathcal{D}_1 but \mathcal{D}_1 is NOT internally tangent to \mathcal{D}_2 .

We review two kinds of vertices described in [8]. Assume p is equal-distant to 3 disks under the d_{\max} metric, i.e., $\mathcal{D}_i, \mathcal{D}_j$, and \mathcal{D}_q . Let C be the circle centered at p and internally tangent to the three disks. Assume that C contains k-1 other disks. By Definition 1, p belongs to 3 Voronoi faces of order k, namely, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_i\}, S)$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_j\}, S)$, and $\mathcal{V}^k(H \bigcup \{\mathcal{D}_q\}, S)$. In this case, p is referred to as a vertex of $V^k(S)$. Furthermore, p is also at the intersection of 3 Voronoi faces of order k+1, namely, $\mathcal{V}^{k+1}(H \bigcup \{\mathcal{D}_i, \mathcal{D}_j\}, S), \mathcal{V}^{k+1}(H \bigcup \{\mathcal{D}_j, \mathcal{D}_q\}, S)$, and $\mathcal{V}^{k+1}(H \bigcup \{\mathcal{D}_q, \mathcal{D}_i\}, S)$. We say that, p is a new vertex of $V^k(S)$ and is an old vertex of $V^{k+1}(S)$. A new vertex of order-k Voronoi diagram becomes an old vertex in the order-(k + 1) Voronoi diagram; an old vertex of order-k Voronoi diagram does not exist in the next order Voronoi diagram.

To facilitate our discussion on disk expansion later, we introduce the notion of *pseudo disk*. A pseudo disk, denoted by \mathcal{D}_{∞} , is a disk centered at the infinity with unit radius. Consider the infinite endpoint p of an infinite edge $e_{i,j}$ in an order-1 max VD. We have $d_{\max}(p,$ $\mathcal{D}_j) = d_{\max}(p, \mathcal{D}_i) = \infty$. Therefore, we can associate p with a pseudo disk, that is, p is a vertex corresponding to 3 disks, namely, \mathcal{D}_j , \mathcal{D}_i , and \mathcal{D}_{∞} . This way, we enclose the open end of each order-1 Voronoi face $\mathcal{V}(\{\mathcal{D}_i\}, S)$ with 2 edges, both associated with a pseudo disk. Therefore, all faces in any order-1 max VD are considered "closed". By similar arguments, the open end of the infinite edge $e_{i,j}$ is bounded by a new vertex corresponding to disks $\mathcal{D}_i, \mathcal{D}_j$, and \mathcal{D}_{∞} .

We study the evolution of the order-k max VD as a disk expands. More specifically, we investigate 2 cases, namely, i) the expanding disk shares edges with at least one disk, and ii) the expanding disk does not share edges with any disk. We refer the disks of the first case as *type-I*, and the latter as *type-II*. Expanding a type-II disk eventually makes it a type-I, while expanding a type-II disk possibly makes some type-I disks type-II. In the following sections, we establish fundamental properties as a disk in an order-k max VD expands. We study type-I disks in section 3.1 and discuss in section 3.2 the expansion of a type-II disk.

Before proceeding, we introduce the notation of maximum circumference. Consider an edge $e_{i,j}$ of 2 faces $\mathcal{V}^k(H_1, S)$, $\mathcal{V}^k(H_2, S)$. There exists a circle that is internally tangent to \mathcal{D}_i and \mathcal{D}_j , and contains $H_1 \bigcup H_2$. We refer to the circle as the maximum circumference of H_1 and H_2 associated with $e_{i,j}$, or simply maximum circumference. Similarly, a maximum circumference associated with a vertex is defined as the circle centered at the vertex and internally tangent to the disks constructing the vertex. If $e_{i,j}$ is an infinite edge, there exists a maximum circumference centered at infinity.

3 Geometrical analysis of order-k max VD

In this section, we study the changes of the order-k max VD as disks of type-I and type-II expand. The proofs are omitted due to space limit.

3.1 Expansion of a type-I disk

Expanding a disk \mathcal{D}_i , where $\mathcal{D}_i \in H$, makes some faces $\mathcal{V}^k(H, S)$ shrink.

Lemma 1 Let $\mathcal{V}^k(H, S)$ and $\mathcal{V}^k(H_i^{\epsilon}, S_i^{\epsilon})$ be the max Voronoi face of H and H_i^{ϵ} in S and S_i^{ϵ} , respectively. Then, $\mathcal{V}^k(H_i^{\epsilon}, S_i^{\epsilon}) \subset \mathcal{V}^k(H, S)$.



Figure 3: The change of order-2 max VD of four disks as disk \mathcal{D}_3 expands. Circles are old vertices, solid dots are new vertices. Dash curves are the diagram corresponding to the expanded \mathcal{D}_3 . Vertices move as shown in arrows.

Roughly speaking, expanding a disk leads to changes in vertices, edges and faces of $V^k(S)$. Vertices move along edges and meet other ones. Formally,

Lemma 2 Consider a vertex $v_{i,j,q}$ of $\mathcal{V}^k(H,S)$. As \mathcal{D}_i expands, $v_{i,j,q}$ moves along $b_{j,q}$. If $v_{i,j,q}$ is new, it moves away from the other end of $e_{j,q}$ elongating $e_{j,q}$. Otherwise, it moves towards the other end of $e_{j,q}$ to shorten $e_{j,q}$.

This is illustrated in Figure 3. As \mathcal{D}_3 expands, the next vertex v_1 moves to elongate edge e_1 , and old vertices v_2 and v_3 move to shorten edges e_2 and e_3 , respectively. As a high order max VD evolves, different sequences of events occur as different types of vertices and edges are involved. The meeting of 2 new vertices or 2 old vertices results in an edge-death/birth, while the meeting of an old vertex and a new one additionally leads to face-death/birth.

Lemma 3 Let $e_{i,j}$ be an edge of $\mathcal{V}^k(H, S)$ with 2 new vertices, e.g., $v_{i,j-1,j}$ and $v_{i,j,j+1}$ in counter-clockwise order in $\mathcal{V}^k(H, S)$. $e_{i,j-1}$ and $e_{i,j+1}$ are the edges of $\mathcal{V}^k(H, S)$ incident to $v_{i,j-1,j}$ and $v_{i,j,j+1}$, respectively. Let p be the intersection of $b_{j,j-1}$ and $b_{j,j+1}$. Assume that the next event as \mathcal{D}_i expands is the meeting of $v_{i,j-1,j}$ and $v_{i,j,j+1}$ at p. If $\mathcal{D}_{j-1} \neq \mathcal{D}_{j+1}$, then with further expansion of \mathcal{D}_i , $e_{i,j} = \emptyset$ and $e_{j-1,j+1} \neq \emptyset$. In addition, both vertices of $e_{j-1,j+1}$ are new.

Lemma 4 Consider an edge $e_{i,j}$ of 2 faces $\mathcal{V}^k(H_1, S)$ and $\mathcal{V}^k(H_2, S)$ with 2 old vertices $v_{i,j,n}$ and $v_{i,j,m}$, where $\{\mathcal{D}_i, \mathcal{D}_n, \mathcal{D}_m\} \subset H_1$ and $\{\mathcal{D}_j, \mathcal{D}_n, \mathcal{D}_m\} \subset H_2$, respectively. Let $\mathcal{V}^k(H_3, S)$ and $\mathcal{V}^k(H_4, S)$ be the other faces, incident to $v_{i,j,n}$ and $v_{i,j,m}$, respectively. If $\mathcal{D}_m \neq \mathcal{D}_n$, and \mathcal{D}_n expands such that the next event is the meeting of $v_{i,j,n}$ and $v_{i,j,m}$, further expansion of \mathcal{D}_n results in i) $e_{i,j} = \emptyset$, and ii) "new" edge $e_{n,m} \neq \emptyset$ of $\mathcal{V}^k(H_3, S)$ and $\mathcal{V}^k(H_4, S)$.

Lemmas 3 and 4 establish the changes in the diagram as vertices of the same kind meet. In general, the meeting of the 2 ends of an edge makes the edge disappear.



Figure 4: The evolution of edges $e_{i,j}$ and $e_{n,m}$ as new vertices move due to the expansion of \mathcal{D}_n .



Figure 5: The evolution of edges $e_{i,j}$ and $e_{n,m}$ as old vertices move due to the expansion of \mathcal{D}_n .

We refer to this as an *edge-death*. If the two vertices differ by one associated disk, another edge is born simultaneously. We refer to this as an *edge-birth*. Vertices of the newly born edge are new or old depending on the types of the meeting vertices. As two vertices are constructed by the same set of disks, the edge-death makes some face disappear. We refer to this as a *face-death* event. We skip the discussion on this kind of vertex meeting due to the space limit.

Figures 4 and 5 illustrate the results of Lemmas 3 and 4, respectively. Figures 4 shows the evolution of edge $e_{n,m}$ connecting 2 new vertices. Initially, $e_{n,m} \neq \emptyset$, and $e_{i,j} = \emptyset$ (Figure 4a). As \mathcal{D}_n expands, 2 vertices of $e_{n,m}$ moves along the corresponding edges toward $\mathcal{V}^k(H_1, S)$ (arrow). If they meet, $e_{n,m}$ disappears and $e_{i,j}$ is born (Figure 4b). Both vertices are new. Figure 5 shows the evolution of edge $e_{i,j}$ connecting 2 old vertices. Initially, $e_{i,j} \neq \emptyset$ and $e_{n,m} = \emptyset$ (Figure 5a). As \mathcal{D}_n expands, a vertex of $e_{i,j}$ moves toward the opposite end (arrow), which makes $e_{i,j}$ shrink. Eventually, $e_{i,j}$ degenerates as 2 vertices of $e_{i,j}$ meet, and $e_{n,m}$ is born (Figure 5b). Both vertices are old. Next, we present the change of the diagram as vertices of different kinds meet.

Lemma 5 Assume that the new vertex $v_{i,j,n}$ meets the old one $v_{i,j,m}$ as \mathcal{D}_n expands to $\mathcal{D}_{n'}(o_n, r_{n'})$, which results in the degeneration of edge $e_{i,j}$. The following holds: i) Either face incident to $e_{i,j}$, e.g., $\mathcal{V}^k(H_1, S)$, where $H_1 = H \bigcup \{\mathcal{D}_n, \mathcal{D}_i\}$ disappears; ii) Prior to the degeneration, $\mathcal{V}^k(H_1, S)$ shares edges $e_{j,n}$, $e_{m,n}$, $e_{m,i}$,



(a) The structure of the diagram prior to the disappearance of face $\mathcal{V}^k(H \bigcup \{\mathcal{D}_i, \mathcal{D}_n\})$.



(b) The structure of the diagram posterior to the disappearance of face $\mathcal{V}^k(H \bigcup \{\mathcal{D}_i, \mathcal{D}_n\})$.

Figure 6: The evolution of faces $\mathcal{V}^k(H \bigcup \{\mathcal{D}_i, \mathcal{D}_j\})$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_i, \mathcal{D}_m\}, S)$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_n, \mathcal{D}_m\}, S)$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_n, \mathcal{D}_m\}, S)$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_j, \mathcal{D}_n\}, S)$, and $\mathcal{V}^k(H \bigcup \{\mathcal{D}_j, \mathcal{D}_m\}, S)$ (|H| = k - 1). Squares denotes new vertices. Circles denotes old vertices.

and $e_{j,i}$ with only four neighbor faces $\mathcal{V}^k(H_2, S)$, $\mathcal{V}^k(H_3, S)$, $\mathcal{V}^k(H_4, S)$, and $\mathcal{V}^k(H_5, S)$, respectively, where $H_2 = H \bigcup \{\mathcal{D}_i, \mathcal{D}_j\}$, $H_3 = H \bigcup \{\mathcal{D}_i, \mathcal{D}_m\}$, $H_4 = H \bigcup \{\mathcal{D}_n, \mathcal{D}_m\}$, $H_5 = H \bigcup \{\mathcal{D}_j, \mathcal{D}_n\}$; and iii) Posterior to the degeneration of $\mathcal{V}^k(H_1, S)$, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_j, \mathcal{D}_m\}, S) \neq \emptyset$, and $e_{j,n}, e_{m,n}, e_{m,i}$, and $e_{j,i}$ of faces $\mathcal{V}^k(H_2, S)$, $\mathcal{V}^k(H_3, S)$, $\mathcal{V}^k(H_4, S)$, $\mathcal{V}^k(H_5, S)$ are replaced by $e_{i,m}, e_{i,j}, e_{n,j}$, $e_{n,m}$, respectively. Thus, $\mathcal{V}^k(H \bigcup \{\mathcal{D}_j, \mathcal{D}_m\}, S)$ consists of 4 edges, namely, $e_{i,m}, e_{i,j}, e_{n,j}$, and $e_{n,m}$, and 4 vertices, including 2 new vertices, namely, $v_{i,n,m}$ and $v_{i,j,n}$, and 2 old vertices, namely, $v_{i,j,m}$, and $v_{j,n,m}$.

Figure 6 illustrates the changes in the diagram as old vertices meet new vertices. Initially, $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{i}, \mathcal{D}_{n}\}, S) \neq \emptyset$ and $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{j}, \mathcal{D}_{m}\}, S) = \emptyset$. As \mathcal{D}_{n} expands, old vertices $v_{i,j,n}$ and $v_{i,n,m}$ move along edges $e_{i,j}$ and $e_{i,m}$ (arrows), respectively, and meet new vertex $v_{i,j,m}$ making face $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{i}, \mathcal{D}_{n}\}, S)$ disappear. Prior to its death, face $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{i}, \mathcal{D}_{n}\}, S)$ shares 4 edges, i.e., $e_{i,j}, e_{n,j}, e_{n,m}$, and $e_{i,m}$ with faces $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{j}, \mathcal{D}_{n}\}, S)$, and $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{n}, \mathcal{D}_{m}\}, S)$, respectively (Figure 6a). Posterior to the degeneration of face $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{i}, \mathcal{D}_{n}\}, S)$ is born, which respectively shares 4 edges, i.e., $e_{m,n}, e_{m,i}, e_{j,i}$, and $e_{j,n}$ with faces $\mathcal{V}^{k}(H \bigcup \{\mathcal{D}_{j}, \mathcal{D}_{m}\}, S)$ is born, which respectively shares 4 edges, i.e., $e_{m,n}, e_{m,i}, e_{j,i}$, and $e_{j,n}$ with faces $\mathcal{V}^{k}(H \cup \{\mathcal{D}_{j}, \mathcal{D}_{n}\}, S)$, $\mathcal{V}^{k}(H \cup \{\mathcal{D}_{i}, \mathcal{D}_{j}\}, S)$, $\mathcal{V}^{k}(H \cup \{\mathcal{D}_{i}, \mathcal{D}_{m}\}, S)$, and $\mathcal{V}^{k}(H \cup \{\mathcal{D}_{i}, \mathcal{D}_{m}\}, S)$.



Figure 7: Illustration of a type-II disk's expansion. Dashed lines are the order-4 diagram of 5 points. Solid lines are maximum circumferences centered at infinite vertices. Light solid circles are the maximum circumferences centered at 2 finite vertices (v_1 and v_2).

(Figure 6b).

3.2 Expansion of a type-II disk

A disk \mathcal{D}_i is a type-II disk if it does not shares any edge with other disks, i.e., $e_{i,j} = \emptyset$, $\forall \mathcal{D}_i \in S, j \neq i$. We can show that a disk \mathcal{D}_i is type-II in $\mathcal{V}^k(S)$ when \mathcal{D}_i contains k other disks, or \mathcal{D}_i corresponds to all faces in $\mathcal{V}^k(S)$, i.e., $\mathcal{D}_i \in H$ for all $\mathcal{V}^k(H, S)$'s in $\mathcal{V}^k(S)$. In this section, we only discuss the latter case. It will be shown later that limiting our consideration to this case is sufficient to construct the order-k Voronoi diagrams. The basic idea is, we expand type-II disks to make them type-I. and then apply the techniques developed for type-I disks as discussed earlier. We can show that when a type-II disk expands, it only touches a maximum circumference centered at an infinite vertex. We illustrate the claim in Figure 7, which shows the order-4 Voronoi diagram of 5 disks of zero radius, $S = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_5\}$. The Voronoi regions are $\mathcal{V}^4(\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_3\}, S), \mathcal{V}^4(\{\mathcal{D}_1, \mathcal{D}_3\}, S)$ $\mathcal{D}_2, \mathcal{D}_5, \mathcal{D}_3\}, S), \ \mathcal{V}^4(\{\mathcal{D}_1, \mathcal{D}_5, \mathcal{D}_4, \mathcal{D}_3\}, S), \text{ and } \mathcal{V}^4(\{\mathcal{D}_2, \mathcal{D}_3\}, S)\}$ $\mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_3\}, S$, which make \mathcal{D}_3 a type-II disk. The diagram (dashed) consists of 2 finite vertices, v_1 and v_2 , whose corresponding maximum circumferences are C_6 , and C_7 shown by light solid circles. The maximum circumferences centered at the infinite end of edges e_i are shown by straight lines C_i , $i \in \{1, 2, 3, 4\}$. As shown in the figure, as \mathcal{D}_3 expands (dashed circle), it first touches C_5 , the maximum circumference centered at the infinite end of edge e_5 at a point in segment $\mathcal{D}_2\mathcal{D}_5$. In fact, we can show that:

Lemma 6 It takes $O(k^2N)$ to process an order-k max VD of N disks so that it contains only type-I disks.

We are now in the position to sketch an algorithm for constructing order-k maximum Voronoi diagrams of disks.

4 The incremental algorithm for order-k max VD construction

In constructing the order- $k \max VD$ of disks S, we start with an order-k Voronoi diagram (VD) of disk centers and iteratively expand each disk in S by a fixed amount d_{\min} , where $d_{\min} \stackrel{\Delta}{=} \min_{i,j \in S} d(o_i, o_j) - \epsilon$. We stop when all disks reach their targeted size. The resulted diagram is the order-k max VD of S. Let $r_{\max} = \max_{i \in S} r_i$ and $r_{\min} = \min_{i \in S} r_i$. Clearly, the total number of rounds a disk needs to expand is bounded by $\left\lceil \frac{r_{\max} - r_{\min}}{d_{\min}} \right\rceil$. This implies that the algorithm terminates. Since disks expand by d_{\min} , they are not contained in other disks until they reach their targeted sizes. Furthermore, when a disk contains k other disks in its expansion, it becomes type-II since the k disks have reached their targeted sizes. Thus, its further expansion does not change the diagram. Therefore, the algorithm proceeds in such a way that all expanding disks are always type-I. As discussed in the previous sections, expanding disks do not make any new type-II disk. Thus, it is always possible to evaluate the expansion such that the next vertex meeting happens. The procedure of order-k max VD construction is summarized in Algorithm 1.

We first derive the number of edges and vertices in the order-k max VD by extending the results in [8].

Lemma 7 The number of vertices and edges in an order-k max VD of N disks is O(kN).

Algorithm 1 takes the set S of N disks as inputs. It starts by constructing the order-k VD of disk centers (line 2), which in fact is the order- $k \mod VD$ of disks whose radii are all equal to the minimum radius of Ndisks, denoted as S'. The process takes $O(k^2 N \log N)$ in running time ([8]). Since the order-k VD may contain type-II disks, we first make them type-I. This takes $O(k^2N)$. Then, we iteratively scan all disks in S and expand those whose radii are smaller than their respective sizes an amount d_{\min} (lines 6 - 19). d_{\min} can be computed in O(N) using the order-1 VD of S', which is a byproduct in the incremental construction of $V^k(S')$. As each disk increases by d_{\min} , a disk \mathcal{D}_i cannot contain disk \mathcal{D}_i unless \mathcal{D}_i has reached it targeted size. Therefore, it validates the earlier claim that we only need to consider type-II disks that do not contain other disks.

Theorem 1 The order-k max VD of N disks can be constructed in $O\left(\left\lceil \frac{r_{\max}-r_{\min}}{d_{\min}} \right\rceil k^2 N \log N\right)$, where r_{\max} and r_{\min} are respectively the maximum and minimum radii of N disks, and d_{\min} is the minimum distance between 2 disk centers.

Proof. (Sketch) We analyze the expansion phase (lines 6-end). Let \mathcal{D}_n be the disk that expands first, and $e_{n,m}$ be the edge that first disappears due to the

expansion of \mathcal{D}_n resulting in the birth of edge $e_{i,j}$ (Figure 4). When edge $e_{i,i}$ is born, the number of edges affected by the expansion of disks \mathcal{D}_i and \mathcal{D}_i increases by 2, while that of \mathcal{D}_m decreases by 1. In general, when an edge of a face disappears, the total number of edges needed to be processed as other disks expand increases by 1. This observation also applies to two old vertices, as well as the case when an old vertex and a new vertex meet. In the order-k max VD, a face corresponds to k disks, thus each edge in a face may be processed k times. Since the number of edges in an order-k max VD is O(kN), the total number of edges processed as N disks, each expands once, is $O(k^2N)$. Line 13 in Algorithm 1 requires a sorted list. Since the total number of edges processed is $O(k^2 N)$, line 13 takes $O(k^2 N \log N)$. Since a disk expands d_{\min} in each round, it needs at most $\lceil \frac{r_{\max} - r_{\min}}{d_{\min}} \rceil$ expansions. Therefore, the time complexity of Algorithm 1 is $O\left(\left\lceil \frac{r_{\max} - r_{\min}}{d_{\min}} \right\rceil k^2 N \log N\right)$.

Algorithm 1: Order-*k* Maximum Voronoi diagram of disks

input : A set of N disks $S = \{\mathcal{D}_1(o_1, r_1), \mathcal{D}_1(o_2, r_2), \dots, \mathcal{D}_N(o_N, r_N)\}$ **output**: The order-k max VD of S, $V^k(S)$ 1 $r_{\min} \leftarrow \min_i r_i;$ **2** $S' \leftarrow \{\mathcal{D}_1(o_1, r'_1 = r_{\min}), \dots, \mathcal{D}_N(o_N, r'_N = r_{\min})\};$ **3** Construct $V^k(S')$: 4 process $V^k(S')$ to transform type-II disks to type-I; 5 $d_{\min} \leftarrow \min_{i,j \in S} d(o_i, o_j) - \epsilon$; 6 repeat for each \mathcal{D}_i such that $r'_i < r_i$ do 7 if $(r'_i + d_{\min}) > r_i$ then 8 $max_inc \leftarrow r'_i - r_i;$ 9 else 10 $max_inc \leftarrow d_{\min};$ 11 while $r'_i < max_inc$ do 12find the smallest expansion e such that an 13 event happens; if $r'_i + e < max_inc$ then $\mathbf{14}$ $r'_i \leftarrow r'_i + e;$ 15update $V^k(S')$ due to the event's 16 consequences; else 17 $r'_i \leftarrow max_inc;$ 18 re-calculate edges/vertices 19 corresponding to \mathcal{D}_i ;

20 until until all disks reach their targeted size ;

5 Conclusion

We have proposed an incremental algorithm to construct order-k maximum Voronoi diagram of disks in the plane. In our approach, disks iteratively expand from zero radius until they reach the targeted size, and the diagram is updated at certain sizes of disks. The algorithm runs in $O\left(\left\lceil \frac{r_{\max} - r_{\min}}{d_{\min}} \right\rceil k^2 N \log N\right)$ time, where r_{\max} and r_{\min} are respectively the maximum and minimum radii of N disks, and d_{\min} is the minimum distance between 2 disk centers. Our contribution is two-fold. First, our algorithm provides a mechanism to quickly update the diagram of an order-k max VD as disk radii change (but disk centers are fixed). Second, our approach is amiable to distributed implementation. When a disk expands, it needs only information of the neighbors to update the diagram structure.

6 Acknowledgment

We thank the referees for providing constructive comments.

- F. Aurenhammer. Power diagrams: properties, algorithms and applications. SIAM J. Comput., 16(1):78– 96, 1987.
- [2] F. Aurenhammer. Improved algorithms for disc and balls using power diagrams. J. Algorithms, 9(2):151– 161, 1988.
- [3] F. Aurenhammer, T. U. Graz, R. Klein, F. Hagen, and P. I. Vi. Voronoi diagrams. In *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland.
- [4] M. I. Karavelas and M. Yvinec. Dynamic additively weighted voronoi diagrams in 2d. pages 586–598, 2002.
- [5] D.-S. Kim, D. Kim, and K. Sugihara. Voronoi diagram of a circle set from voronoi diagram of a point set: I. topology. *Computer Aided Geometric Design*, 18(6):541–562, 2001.
- [6] D.-S. Kim, D. Kim, and K. Sugihara. Voronoi diagram of a circle set from voronoi diagram of a point set: Ii. geometry. *Computer Aided Geometric Design*, 18(6):563– 585, 2001.
- [7] D.-S. Kim, B. Lee, C.-H. Cho, and K. Sugihara. Planesweep algorithm of o(nlogn) for the inclusion hierarchy among circles. In *ICCSA (3)*, pages 53–61, 2004.
- [8] D.-T. Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Trans. Comput.*, 31(6):478–487, 1982.
- [9] M. I. Shamos. Computational geometry. PhD thesis, New Haven, CT, USA, 1978.
- [10] K. Vu and R. Zheng. Robust coverage under uncertainty in wireless sensor networks. In 2011 Proceedings IEEE INFOCOM (INFOCOM 2011), pages 2015–2023, Shanghai, P.R. China, 4 2011.

Approximating a Motorcycle Graph by a Straight Skeleton

Stefan Huber*

Martin $Held^{\dagger}$

Abstract

We investigate how a straight skeleton can be used to approximate a motorcycle graph. We explain how to construct a planar straight-line graph G such that the straight skeleton of G reveals the motorcycle graph of M, for every given finite set M of motorcycles. An application of our construction is a proof of the Pcompleteness of the construction problem of straight skeletons of planar straight-line graphs and simple polygons with holes.

1 Introduction

1.1 Motivation

The straight skeleton S(G) of an *n*-vertex planar straight-line graph G is a skeleton structure similar to Voronoi diagrams, but consists of straight-line segments only. It was introduced by Aichholzer et al. [1] for polygons, and was generalized to planar straight-line graphs by Aichholzer and Aurenhammer [2]. The motorcycle graph problem was introduced by Eppstein and Erickson [5] in an attempt to extract the essential subproblem of constructing straight skeletons. Indeed, the algorithms by Cheng and Vigneron [4] and Huber and Held [6] use motorcycle graphs as a tool for the construction of straight skeletons.

In this work we reverse the question and ask how the motorcycle graph can be computed by employing straight skeletons: We show that for a given set of motorcycles we can construct a planar straight-line graph G such that certain parts of $\mathcal{S}(G)$ approximate the motorcycle graph up to a given tolerance. Since Eppstein and Erickson [5] proved that the motorcycle graph construction problem is P-complete, we can provide a proof that constructing the straight skeleton for planar straight-line graphs and polygons with holes is Pcomplete as well. As a consequence, there is no hope to find an efficient parallel algorithm for straight skeletons, unless $\mathsf{P} = \mathsf{NC}$.

We note that Eppstein and Erickson were the first to mention the P-completeness of straight skeletons: In [5] they claim that arguments similar to those used for proving the P-completeness of motorcycle graphs would apply to straight skeletons as well. Further, they suggest that a motorcycle graph can be approximated by the straight skeleton of a set of sharp isosceles triangles, with one triangle per motorcycle. No details and no proof are given, though. In this paper we pick up their suggestion and prove that the straight skeleton of a set of meticulously chosen triangles does indeed approximate the motorcycle graph of the motorcycles given.

1.2 Preliminaries and Definitions

Consider a planar straight-line graph G with n vertices, none of them being isolated. Vertices of degree one are called terminals. According to [2], the definition of the straight skeleton $\mathcal{S}(G)$ of G is based on a wavefrontpropagation process: Each edge e of G sends out two wavefronts, which are parallel to *e* and have unit speed. At terminals of G an additional wavefront orthogonal to the single incident edge is emitted. The wavefront $\mathcal{W}(G,t)$ of G at some time t can be interpreted as a 2regular kinetic straight-line graph. Except for the vertices originating from the terminals of G, all vertices of $\mathcal{W}(G,t)$ move along bisectors of straight-line edges of G, see Fig. 1. During the propagation of $\mathcal{W}(G,t)$ topological changes occur: a wavefront edge may collapse ("edge event") or a wavefront edge may be split by a wavefront vertex ("split event"). The straight-line segments traced out by the vertices of $\mathcal{W}(G,t)$ form $\mathcal{S}(G)$. The edges of $\mathcal{S}(G)$ are called "arcs" and bound the "faces" of $\mathcal{S}(G)$. Wavefront vertices are called reflex (resp. convex) if they have a reflex (resp. convex) angle at the side where the propagate to. The arcs of $\mathcal{S}(G)$ which are traced out by reflex (resp. convex) vertices of $\mathcal{W}(G,t)$ are called reflex (resp. convex) arcs.

Consider a set of moving points in the plane, called



Figure 1: Left: The straight skeleton (dotted) is defined by propagating wavefronts (gray) of the input G (bold). Right: The motorcycle graph (red) induced by G.

^{*}Universität Salzburg, FB Computerwissenschaften, Salzburg, Austria, shuber@cosy.sbg.ac.at

[†]Universität Salzburg, FB Computerwissenschaften, Salzburg, Austria, held@cosy.sbg.ac.at



Figure 2: The feasible area of a face. Left: only m_1 has e as an arm. Right: m_1 and m_2 have e as an arm.

"motorcycles", that drive along straight-line rays according to given speed vectors. Each motorcycle leaves a trace behind it and stops driving — it "crashes" when reaching the trace of another motorcycle. The arrangement of these traces is called motorcycle graph, cf. [5]. We denote by $\mathcal{M}(m_1, \ldots, m_n)$ the motorcycle graph of the motorcycles m_1, \ldots, m_n where each motorcycle m_k is given by a start point p_k and speed vector v_k . We adopt the assumption by Cheng and Vigneron [4] that no two motorcycles may crash simultaneously.

In [6] we defined a motorcycle graph on a planar straight-line graph G, by generalizing Cheng and Vigneron's concept [4]. The idea is that a motorcycle mis defined for each reflex wavefront vertex v in $\mathcal{W}(G, 0)$ that starts from v and has the same speed vector as v. We call the wavefront edges incident to v the arms of m; the arm left (resp. right) of the speed ray of m is called left (resp. right) arm. Additionally, we assume that motorcycles crash if they hit an edge of G. We denote the resulting motorcycle graph by $\mathcal{M}(G)$, see Fig. 1. The following two theorems were proved in [4] for non-degenerate¹ polygons with holes and in [6] for general planar straight-line graphs G. (The concept of the feasible area of an edge e of G is illustrated in Fig. 2.)

Theorem 1 $\mathcal{M}(G)$ covers the reflex arcs of $\mathcal{S}(G)$.

Theorem 2 Consider a face f(e) of S(G) corresponding to the wavefront edge e. Then f(e) is contained in a "feasible area" which is bounded by (i) e at time zero, (ii) by traces of motorcycles m that have e as an arm and (iii) by rays perpendicular to e starting the end of those motorcycle traces, if existing, and at the end of eotherwise.

2 Approximating a motorcycle graph by a straight skeleton

Let us consider n motorcycles m_1, \ldots, m_n , where each motorcycle m_i has a start point p_i and a speed vector v_i . We assume that no two motorcycles crash simultaneously into each other. Can we find an appropriate planar straight-line graph G such that (a subset of the reflex arcs of) S(G) and $\mathcal{M}(m_1, \ldots, m_n)$ cover each other up to some given tolerance?



Figure 3: At each point p_i we set an isosceles triangle Δ_i with an angle of $2\alpha_i$, such that $\lambda |v_i| = 1/\sin \alpha_i$ and the motorcycle trace s_i bisects the angle of Δ_i at p_i .

Theorem 1 tells us that the reflex arc of $\mathcal{S}(G)$ that corresponds to a motorcycle m_i approximates the trace of m_i up to some gap. One observes that the faster mmoves the better its trace tends to be approximated by the corresponding reflex arc in $\mathcal{S}(G)$. It is easy to see that $\mathcal{M}(m_1, \ldots, m_n)$ remains unchanged if we multiply each speed vector v_i by a positive constant λ . In order to obtain reflex arcs of $\mathcal{S}(G)$ that overlap with the traces in $\mathcal{M}(m_1, \ldots, m_n)$ we put at each start point p_i an isosceles triangle Δ_i with an angle of $2\alpha_i$ at p_i , where

$$\alpha_i := \arcsin \frac{1}{\lambda |v_i|},\tag{1}$$

and λ large enough such that $\lambda |v_i| \ge 1$ for all $1 \le i \le n$, see Fig. 3. The lengths of the arms of Δ_i will be specified later.

By definition of S(G), a reflex wavefront vertex u_i is emanated from each p_i and its speed vector equals λ times the speed vector of the motorcycle m_i . However, note that further motorcycles are introduced at the additional corners of each triangle. We can now rephrase our initial question: can we always find λ large enough such that those reflex arcs that are traced out by u_i approximate $\mathcal{M}(m_1, \ldots, m_n)$ up to some given tolerance?

Let us denote by s_i the trace of m_i and by $s_i + D_r$ the Minkowski sum of s_i and the disk with radius r centered at the origin. The motorcycle traces of $\mathcal{M}(m_1, \ldots, m_n)$ are closed sets and two traces s_i, s_j intersect if and only if m_i crashed into m_j or vice versa. Hence there is an $\mu > 0$ such that for all $1 \leq i < j \leq n$ it holds that m_i crashes into m_j or vice versa if and only if $s_i + D_{\mu}$ intersects $s_j + D_{\mu}$. For example, let μ be a third of the minimum of the pairwise infimum distances of disjoint traces s_i, s_j . We further assume that μ is small enough such that $p_i + D_{\mu}$ does not intersect any $s_j + D_{\mu}$ for all $1 \leq i, j \leq n$, except if $p_i \in s_j$. (This will be needed for Lemma 4.)

We define the planar straight-line graph G by the set of triangles Δ_i at each start point p_i , where the length of the arms incident to p_i are set to $\mu/2$ and the angle of Δ_i at p_i is given by Eqn. (1).

Lemma 3 For any $1 \le i \le n$ the wavefronts of Δ_i stay within $s_i + D_{\mu}$ until time $\frac{\mu}{4}$ for $\lambda \ge \frac{2}{|v_i|}$.

Proof. Consider a triangle Δ_i at some start point p_i . Note that λ is large enough such that $2\alpha_i$ is at most

 $^{^1{\}rm A}$ polygon is called non-degenerate if no two of the resulting motorcycles crash simultaneously into each other.



Figure 4: The wavefronts of Δ_i are bounded to $s_i + D_{\mu}$ until time $\mu/4$. Top: the motorcycle m_i did not crash until that time. Bottom: the motorcycle m_i did crash into another trace (dotted line segment) until that time.

60°. Hence, the other two angles of Δ_i are at least 60° and, therefore, the two additional motorcycles at Δ_i have a speed of at most 2. Since the start points of those motorcycles are $\mu/2$ away from p_i and they drive at most a distance of $\mu/2$ in time $\mu/4$, they stay within $p_i + D_{\mu}$.

According to Thm. 2, we consider the feasible areas of the edges of Δ_i restricted to an orthogonal distance of at most $\mu/4$ to the edges of Δ_i , see Fig. 4. We distinguish two cases: the motorcycle m_i (i) did crash or (ii) did not crash until time $\mu/4$. However, in both cases the corner points of these restricted feasible areas are contained within $s_i + D_{\mu}$, the restricted feasible areas are convex and $s_i + D_{\mu}$ is convex. Hence, the wavefronts of Δ_i until time $\mu/4$ are contained within $s_i + D_{\mu}$.

We denote by $\overrightarrow{s_i}$ the ray starting at p_i in direction v_i and define

$$L := \max_{1 \le i, j \le n} d(p_i, \overrightarrow{s_i} \cap \overrightarrow{s_j}).$$
(2)

Note that we may only consider indices i, j for which $\overrightarrow{s_i} \cap \overrightarrow{s_j}$ is not empty. If no such indices i, j exist then we set L to zero. Further, let us denote by $\varphi_{i,j} \in [0, \pi]$ the non-oriented angle spanned by v_i and v_j , with $\varphi_{i,j} = \varphi_{j,i}$. Next we define

$$\Phi := \min_{1 \le i < j \le n} \mathbb{R}^+ \cap \{\varphi_{i,j}, \pi - \varphi_{i,j}\}.$$
 (3)

If this set it is empty, i.e. if all motorcycles are driving on parallel tracks, then we set $\Phi := \pi/2$.

Lemma 4 Let m_i denote a motorcycle crashing into the motorcycle m_j . The wavefronts of Δ_i do not cause a split event for u_j until time $\mu/4$ for $\lambda \geq \frac{2}{\min_k |v_k| \sin \Phi}$.

Proof. We note that $\lambda \geq \frac{2}{|v_k|\sin\Phi}$ holds for any $1 \leq k \leq n$ which means that

$$\sin \alpha_k \le \frac{1}{|v_k|\lambda} \le \frac{1}{2} \sin \Phi \le \sin \frac{\Phi}{2},$$

since sin is concave on $[0, \pi]$. By further noting that sin is monotone on $[0, \pi/2]$ we see that

$$\alpha_k \le \frac{\Phi}{2} \qquad \forall \ 1 \le k \le n.$$

The case where m_j also crashes into m_i is excluded since two motorcycles do not crash simultaneously by assumption. The cases where s_i and s_j are collinear are either trivial or excluded by assumption. Without loss of generality, we may assume that s_i is right of $\vec{s_j}$, see Fig. 5. We denote by q the endpoint of the reflex straight-skeleton arc incident to p_i . Let us consider the left (resp. right) bisector between the left (resp. right) arm of m_i and the right arm of m_j , starting from q.

From the proof of Lem. 3 it follows that in order that u_i is involved in a split event with a wavefront from Δ_i until time $\mu/4$ it is necessary that one of both bisectors intersects $\overrightarrow{s_j}$. (Check Figure 4: The two additional motorcycles from Δ_i stay within $p_i + D_{\mu}$. Hence, we only have to consider the arms of m_i .) Let us consider the right bisector. Recall that $\alpha_i, \alpha_i \leq \Phi/2$ and that $\pi - \varphi_{i,j} \geq \Phi$. In the extremal case, where equality is attained for all three inequalities, the right bisector is just parallel to s_i , but strictly right of $\vec{s_i}$. In all other cases the bisector rotates clockwise at q such that our assertion is true in general. Analogous arguments hold for the left bisector. Summarizing, the vertex u_i does not lead to a split event with the wavefronts of Δ_i until time $\mu/4$. \square



Figure 5: The wavefronts of Δ_i do not cause a crash with u_j .

Lemma 5 Let m_i denote a motorcycle crashing into the motorcycle m_i . For any $\epsilon > 0$ and

$$\lambda \geq \frac{1}{\min_k |v_k| \cdot \sin \Phi} \cdot \max\left\{2, \frac{L}{\min\{\mu/4, \epsilon\}}\right\},\,$$

the trace s_i is covered up to a gap size ϵ by the reflex arc traced out by u_i .

Proof. We will prove the following: consider an arbitrary point q on s_i whose distance to the endpoint of s_i is at least ϵ . Then we show that q is reached by u_i until time $\frac{\mu}{4}$.

Consider Fig. 6. We first show that until time $\mu/4$ the vertex u_i may only cause a split event with the wavefronts of Δ_j . By Lem. 3, we know that until time $\mu/4$ only the wavefronts of a triangle Δ_k could cause a split event with u_i if s_k and s_i intersect. Hence, m_k crashed against s_i . However, by Lem. 4 it follows that u_i does not lead to a split event with the wavefronts from Δ_k .

W.l.o.g., we may assume that s_i lies right to $\vec{s_j}$. To show that u_i reaches q until time $\mu/4$ it suffices to prove that q has a smaller orthogonal distance to the left arm of m_i than to the right arm of m_j and that the orthogonal distance of q to the left arm of m_i is at most $\mu/4$.

The orthogonal distance of q to the left arm of m_i is at most $L \cdot \sin \alpha_i$. The orthogonal distance of q to the right arm of m_j is at least the orthogonal distance of q to s_j . However, this distance is at least $\epsilon \cdot \sin \varphi_{i,j}$. Summarizing, our assertion holds if

$$L \cdot \sin \alpha_i \leq \min\{\frac{\mu}{4}, \epsilon \sin \varphi_{i,j}\}$$

which is

$$\lambda \ge \frac{L}{|v_i| \cdot \min\{\frac{\mu}{4}, \epsilon \sin \varphi_{i,j}\}}$$

However, our choice for λ fulfills this condition. The case where s_i and s_i are collinear such that m_i crashes

Figure 6: The point q has a smaller orthogonal distance to the left arm of m_i than to the right arm of m_j .

at p_j is similar. The wavefront of Δ_j reaches q at least in time $\epsilon/2$ and v_i reaches q in at most $\frac{L}{\lambda|v_i|}$ time.

Let us define by $S^*_{\lambda}(m_1, \ldots, m_n) \subset S(G)$ the union of the reflex straight-skeleton arcs which emanate from p_1, \ldots, p_n , where G is given as described above. Then we get the following corollary of Lem. 5.

Corollary 6

$$\lim_{\lambda \to \infty} \mathcal{S}_{\lambda}^*(m_1, \dots, m_n) = \mathcal{M}(m_1, \dots, m_n)$$

This corollary also asserts that a point q on a motorcycle trace s_i of a motorcycle m_i that never crashed is covered by an arc of $S^*_{\lambda}(m_1, \ldots, m_n)$ for large enough λ . However, this is easily proved by applying Lem. 3 and Lem. 4, and by finally finding λ large enough such that the point q is reached by u_i until time $\mu/4$.

3 Computing the motorcycle graph

In order to actually compute the motorcycle graph $\mathcal{M}(m_1, \ldots, m_n)$ from $\mathcal{S}^*_{\lambda}(m_1, \ldots, m_n)$ for some big λ , we still have to cope with the remaining gaps within $\mathcal{S}^*_{\lambda}(m_1, \ldots, m_n)$. For deciding whether a motorcycle m_i actually escapes or crashes into some trace s_j we want to determine λ large enough such that the following two conditions hold:

- If m_i crashes into a trace s_j then u_i leads to a split event until the time $\mu/4$ and the reflex arc traced out by u_i has an endpoint in a straight-skeleton face of an edge of Δ_j . (The right arm of m_j if s_i is right of \vec{s}_j and the left arm if s_i is left of \vec{s}_j .)
- If m_i escapes then u_i did not lead to a split event until the time μ/4.

Lemma 7 Consider $\mathcal{S}(G)$ with

$$\lambda \ge \frac{\max\left\{2, \frac{8L}{\mu}\right\}}{\min_k |v_k| \cdot \sin \Phi}.$$

Then m_i crashes into s_j if and only if u_i leads to a split event with a wavefront emanated by Δ_j until time $\mu/4$. In particular, m_i escapes if and only if u_i does not lead to a split event until time $\mu/4$.

Proof. We distinguish two cases. First, suppose that the motorcycle m_i crashed into the trace s_j , see Fig. 7. We may assume without loss of generality that s_i is right of $\overrightarrow{s_j}$. First we note that by our choice of λ we may apply Lem. 3. We denote by p the intersection $s_i \cap s_j$. Further, we set $\epsilon := \mu/s$ which allows us to apply Lem. 5 since

$$\max\left\{2,\frac{8L}{\mu}\right\} \ge \max\left\{2,\frac{L}{\min\{\mu/4,\epsilon\}}\right\}$$





Figure 7: The reflex wavefront vertex at p_i is forced to cause a split event until time $\mu/4$.

Thus, the endpoint q of the reflex arc traced out by u_i has a distance of at most μ/s to p.

On the other hand, u_j reaches p in at most $\frac{L}{\lambda |v_j|}$ time. We conclude that the wavefront edge from the right arm of m_j reaches q in at most $\frac{L}{\lambda |v_j|} + \frac{\mu}{8}$ time which is bounded from above by

$$\frac{L \cdot \mu \cdot \min_k |v_k| \cdot \sin \Phi}{8 \cdot L \cdot |v_j|} + \frac{\mu}{8} \le \frac{\mu}{4}$$

by our choice of λ . Summarizing, the point q is swept by the wavefront of the right arm of m_j and is reached by u_i until $\mu/4$ time. Hence, the vertex u_i must have caused a split event until the requested time by crashing into the wavefront of the right arm of m_j .

For the second case assume that m_i escapes. Lemmas 3 + 4 imply that u_i does not lead to a split event until time $\mu/4$.

In order to compute the motorcycle graph by employing a straight skeleton algorithm, we would have to compute the appropriate values for L, Φ, μ in order to determine a sufficiently large λ . While L and Φ are already given independent of $\mathcal{M}(m_1, \ldots, m_n)$, the following lemma gives a formula for μ for which the actual motorcycle graph is not needed to be known. (In the following lemma we take $d(\vec{s_i}, \emptyset)$ to be infinity.)

Lemma 8 For any two disjoint motorcycle traces s_i and s_j the Minkowski sums $s_i + D_{\mu}$ and $s_j + D_{\mu}$ are disjoint if

$$\mu := \frac{1}{3} \min_{1 \le i, j, k \le n} \mathbb{R}^+ \cap \{ d(\overrightarrow{s_i}, p_j), d(\overrightarrow{s_i}, \overrightarrow{s_j} \cap \overrightarrow{s_k}) \}.$$

Proof. In order to guarantee that the Minkowski sums are disjoint, it suffices to set μ to a lower bound of a third of the minimum of all pairwise infimum distances of disjoint traces s_i and s_j .

Let us consider two disjoint traces s_i and s_j . We choose two points $q_i \in s_i, q_j \in s_j$ for which $d(s_i, s_j) = d(q_i, q_j)$ holds. We may assume that either q_i is an endpoint of s_i or q_j is an endpoint of s_j . (If s_k is a ray,

the only endpoint is p_k .) If q_j is the start point of s_j then we have $d(s_i, s_j) = d(s_i, p_j) \ge 3\mu$. If q_j is the opposite endpoint of s_j — and hence s_j is a segment then s_j crashed into some other motorcycle trace. So there is a trace s_k such that $q_j = s_j \cap s_k$. Again we get $d(s_i, s_j) = d(s_i, q_j) \ge 3\mu$. Analogous arguments hold if q_i is an endpoint of s_i .

After computing appropriate values for L, Φ and μ for a set of motorcycles m_1, \ldots, m_n , we can determine a sufficiently large λ and build the input graph G by constructing the triangles $\Delta_1, \ldots, \Delta_n$ as described. After computing the straight skeleton S(G) we determine the length of each motorcycle's trace by applying the conditions listed in Lemma 7.

4 Constructing the straight skeleton is P-complete

Atallah et al. [3] described a framework for reductions of the P-complete PLANAR CIRCUIT VALUE problem and used it to prove the P-completeness of several geometric problems. However, investigating the P-completeness of geometric problems often requires the availability of exact geometric computations which are not in NC. In order to investigate the P-completeness of geometric problems Attalah et al. [3] propose that the answers to basic geometric queries are provided by an oracle.

A basic building block for showing that the straight skeleton is P-complete is the construction of the triangles Δ_i . Assume p_i, α_i, v_i, μ , and λ are given. We further assume that an oracle determines the intersection points of two circles with given centers and radii. Then we can construct Δ_i as follows. We first compute the point $q_i = p_i + \lambda v_i$, which is the position of m_i at time one, see Figure 3. Then we construct the circle C_1 with $[p_i, q_i]$ as diameter and the circle C_2 centered at p_i with radius 1. The two circles C_1, C_2 intersect at two points, say a_i, b_i . The triangle Δ_i^* with vertices a_i, b_i, q_i is an isosceles triangle with angle $2\alpha_i$ at q_i and therefore similar to Δ_i . The length of the arms of Δ_i^* at q_i are at most $\lambda |v_i|$. By scaling the triangle by the factor $\mu/2\lambda |v_i|$ and by translating it accordingly, we get a triangle with the desired geometry. (Actually, the arms of the constructed triangle are a bit shorter than $\mu/2$, but this is only to our advantage.)

Eppstein and Erickson [5] proved that the computation of the motorcycle graph is P-complete by presenting a LOGSPACE reduction of the CIRCUIT VALUE problem to the computation of the motorcycle graph. The CIR-CUIT VALUE problem asks for the output value of a gate in a binary circuit with n input gates, which is presented an input vector of binary values. Eppstein and Erickson demonstrated how to translate the CIRCUIT VALUE problem to the motorcycle graph construction problem by simulating each gadget using motorcycles. The values 1 and 0 on a wire are represented by the presence or absence of a motorcycle on a track. The original question for the output value of a particular gate of the circuit can be translated to the question whether a specific motorcycle crashes until some distance from its start point. In other words, Eppstein and Erickson proved that the decision problem whether a specific motorcycle crashes until some distance from its start point is P-complete.

Lemma 9 The construction of the straight skeleton of a planar straight-line graph is P-complete under LOGSPACE reductions.

Proof. Eppstein and Erickson reduced the CIRCUIT VALUE problem to a specific motorcycle graph problem. The next step is to reduce the motorcycle graph problem to the straight-skeleton problem: we construct a suitable input graph G which allows us to apply Lem. 7 for deciding whether a specific motorcycle crashes until some distance from its start point.

According to [5] all O(1) different types of motorcycle gadgets are arranged in an $n \times n$ grid, and each gadget takes constant space and consists of O(1) motorcycles. For determining a sufficiently large λ we need bounds on L, Φ and μ . An upper bound on L is the length of the diagonal of the $n \times n$ grid. Further, $\sin \Phi \geq 1/2$ since the direction angles of the motorcycles are all multiples of $\pi/4$. A lower bound on μ can be found by considering each gadget independently and taking the minimum among them. Finally, we build G by constructing for each motorcycle (independently from each other) an isosceles triangle, as described in Section 2.

We can easily extend the construction of G to form a polygon with holes, by adding a sufficiently large bounding box to G. As remarked in [5], only one motorcycle m may leave the bounding box B of the $n \times n$ grid. The motorcycle m encodes the output of the binary circuit by leaving B if the circuit evaluates to 1 and by crashing within B if the circuit evaluates to 0.

By Lemma 7, the reflex wavefront vertex u, which corresponds to m, encodes the output of the binary circuit by leading to a split event until time $\mu/4$ if and only if the circuit evaluates to 0. Lemma 3 implies that the wavefront vertices stay within $B + D_{\mu}$, except possibly u. Hence, we could enlarge B by 2μ at each side and add it to G such that the wavefronts of B do not interfere with the wavefronts of the triangles until time $\mu/4$, except for u. Still, we can determine the output of the binary circuit by checking whether the reflex straightskeleton arc that corresponds to u ends within $B + D_{\mu}$ until time $\mu/4$. (Recall that the end of a reflex straightskeleton arc marks the place where the reflex wavefront vertex led to a split event.)

Corollary 10 The construction of the straight skeleton of a polygon with holes is *P*-complete under LOGSPACE reductions. Unfortunately, our P-completeness proof cannot be applied easily to simple polygons. Consider the five motorcycles depicted in Fig. 8. A polygon, whose reflex straight-skeleton arcs would approximate the motorcycle traces, would need to connect the start points of mand m_1, \ldots, m_4 . But in order to decide where the red square formed by the traces of m_1, \ldots, m_4 can be penetrated by the polygon, while avoiding to stop a motorcycle too early, it would be necessary to know a specific pair m_i, m_j of motorcycles such that m_i is guaranteed to crash into the trace of m_j . However, deciding whether a specific motorcycle crashes does not seem much easier than computing the whole motorcycle graph. Hence, it remains open whether the computation of straight skeletons of polygons is P-complete.



Figure 8: These motorcycle traces cannot be approximated easily by a straight skeleton of a simple polygon.

- O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner. Straight Skeletons of Simple Polygons. In *Proc. 4th Internat. Symp. of LIESMARS*, pages 114–124, Wuhan, P.R. China, 1995.
- [2] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In A. Samoilenko, editor, *Voronoi's Impact on Modern Science, Book 2*, pages 7–21. Institute of Mathematics of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998.
- [3] M. Atallah, P. Callahan, and M. Goodrich. P-complete Geometric Problems. Internat. J. Comput. Geom. Appl., 3(4):443–462, 1993.
- [4] S.-W. Cheng and A. Vigneron. Motorcycle Graphs and Straight Skeletons. Algorithmica, 47:159–182, Feb. 2007.
- [5] D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete Comput. Geom.*, 22(4):569–592, 1999.
- [6] S. Huber and M. Held. Theoretical and Practical Results on Straight Skeletons of Planar Straight-Line Graphs. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 171–178, Paris, France, June 2011.

Small Octahedral Systems

Grant Custard *

Antoine Deza[†]

Tamon Stephen[‡]

Feng Xie[§]

Abstract

We consider set systems that satisfy a certain octahedral parity property. Such systems arise when studying the colourful simplices formed by configurations of points of in \mathbb{R}^d ; configurations of low colourful simplicial depth correspond to systems with small cardinality. This construction can be used to find lower bounds computationally for the minimum colourful simplicial depth of a configuration, and, for a relaxed version of colourful depth, provide a simple proof of minimality.

1 Introduction

We are interested in set systems of the following type: the base set **S** is partitioned into *colours* $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_m$ for some m, and the sets consist of one element from each \mathbf{S}_i . In other words, these are m-uniform hypergraphs where each hyperedge has a unique intersection with each colour \mathbf{S}_i , we will sometimes refer to the sets that belong to a given system as *edges*. We call a subset of **S** *colourful* if it contains at most one point from each \mathbf{S}_i . Thus the edges of any system are colourful. When a colourful set has a point from \mathbf{S}_i , we will call this point the *i*th *coordinate* of the set.

We call a colourful set of m-1 points which misses \mathbf{S}_i an \hat{i} -transversal, and call any pair of disjoint \hat{i} -transversal an octahedron. We say that an *m*-uniform collection of colourful edges forms an octahedral system if it satisfies the following property:

Property 1 For any octahedron Ω , the parity of the set of edges using points from Ω and a fixed point s_i for the ith coordinate is the same for all choices of s_i .

The term octahedron comes from the following geometric motivation. A point $p \in \mathbb{R}^d$ has simplicial depth k relative to a set S if it is contained in k closed simplices

[‡]Department of Mathematics, Simon Fraser University, British Columbia, Canada tamon@sfu.ca generated by (d + 1) sets of S. This was introduced by Liu [21] as a statistical measure of how representative pis of S, and is a source of challenging problems in computational geometry – see for instance [1], [14] and [22]. More generally, we consider *colourful simplicial depth*, where the single set S is replaced by (d + 1) sets, or colours, $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$, and the *colourful* simplices containing p are generated by taking one point from each set.

From any such colourful configuration, we can form a system of vectors \mathcal{V} where $\mathbf{v} = (s_1, \ldots, s_{d+1})$ is in ${\mathcal V}$ if and only if the colourful simplex described by ${\bf v}$ contains **0**. In this context, \hat{i} -transversals are simply vectors with the *i*th coordinate removed, and *octahedra* are pairs of disjoint *i*-transversals. It is a topological fact that such a system satisfies Property 1, see for instance the Octahedron Lemma of [4] for a proof. Thus \mathcal{V} is an octahedral system with m = d + 1. When the points of an octahedron Ω from \mathcal{V} considered as points in \mathbb{R}^d form a cross-polytope, i.e. a *d*-dimensional octahedron, in the geometric sense that $\operatorname{conv}(\Omega)$ is a cross-polytope and same coloured points are not adjacent in the skeleton of the polytope, then the even and odd case correspond to **0** lying inside and outside Ω respectively. Figure 1 illustrates this in a two dimensional case where $\mathbf{0}$ is at the centre of a circle that contains points of the three colours.



Figure 1: Two-dimensional cross-polytopes Ω containing 0 and not.

It is interesting to get lower bounds for the number of colourful simplices containing p for given configurations, for instance satisfying convexity properties as described in Section 1.1 below. Besides the intrinsic appeal of the problem, its solution is a bound on the number of solutions to a colourful linear program in the sense of [5] and [11]. One strategy for establishing this bound is to show that certain small octahedral systems cannot

^{*}Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada lounsbg@mcmaster.ca

[†]Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada and Equipe Combinatoire et Optimisation, Université Pierre et Marie Curie, Paris, France, deza@mcmaster.ca

[§]Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada xief@mcmaster.ca

exist. In particular, it leads to two nice combinatorial questions: what is the smallest non-empty octahedral system in terms of the number of edges on m (i.e. d+1) sets of m points, and what is the smallest such system where every point is contained in some edge. In Section 2 we show that the answer to the first question is m and use this to prove a conjecture about point configurations. The second question suggests a method of computationally attacking the colourful simplicial depth problem, see below, at least for small dimension. Some progress on this is described in Section 3. Finally, in Section 4 we consider some further questions about octahedral systems.

1.1 Colourful Simplicial Depth Problems

Consider the colourful configurations described above. Without loss of generality we assume that p = 0 and that the points in $\mathbf{S} \cup \{\mathbf{0}\}$ are in general position. If the convex hulls of the S_i 's contain 0 in their interior, we say that the configuration satisfies the *core* condi-Bárány's Colourful Carathéodory Theorem [3] tion. shows that the core conditions imply that $\mathbf{0}$ must be contained in some colourful simplex. In other words, we have $\mu(d) \geq 1$ where $\mu(d)$ denotes the minimum number of colourful simplices drawn from $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ that contain 0 for all configurations with the core condition. The sets $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ must each contain at least (d+1) points for **0** to be in the interior of their convex hulls, and since we are minimizing we can assume they contain no additional points, i.e. that $|\mathbf{S}_i| = d + 1$ for each i.

The quantity $\mu(d)$ was investigated in [10], where it is shown that $2d \leq \mu(d) \leq d^2 + 1$, that $\mu(d)$ is even for odd d, and that $\mu(2) = 5$. This paper also conjectures that $\mu(d) = d^2 + 1$ for all $d \geq 1$. Subsequently, [4] verified the conjecture for d = 3 and provided a lower bound of $\mu(d) \geq \max(3d, \left\lceil \frac{d(d+1)}{5} \right\rceil)$ for $d \geq 3$, while [24] independently provided a lower bound of $\mu(d) \geq \left\lfloor \frac{(d+2)^2}{4} \right\rfloor$, before [12] showed that $\mu(d) \geq \lceil \frac{(d+1)^2}{2} \rceil$. A recent generalization of the Colourful Carathéodory

A recent generalization of the Colourful Carathéodory Theorem in [2] and [17] relaxes the condition of **0** being in the convex hull of each \mathbf{S}_i to require only that **0** is in the convex hull of $\mathbf{S}_i \cup \mathbf{S}_j$ for all i and j, and \mathbf{S}_i not empty for all i. The analogous quantity $\mu^{\diamond}(d)$, which denotes the minimum number of colourful simplices drawn from $\mathbf{S}_1, \ldots, \mathbf{S}_{d+1}$ that contain $\mathbf{0} \in \mathbb{R}^d$ given that $|\mathbf{S}_i| = d + 1$ for all i and $\mathbf{0} \in \mathbf{S}_i \cup \mathbf{S}_j$ for each $i \neq j$, has been investigated in [12] where it is shown that $\mu^{\diamond}(d) \leq d + 1$, $\mu^{\diamond}(2) = 3$, and $\mu^{\diamond}(3) = 4$. The associated octahedral system of (d+1) points in (d+1)colours satisfies Property 1.

Remark 2 Colourful simplicial depth was introduced in the context of lower bounds for ordinary simplicial depth. This problem is quite challenging even in two dimensions: it has been studied at least since Kártesi [19]; the bound of $\frac{1}{27}n^3 + O(n^2)$ was established in [6], but the the construction in that paper of a set of points meeting this bound needed to be revised, see [7]. For general d, finding a tight bound remains a challenging problem. Recently Gromov [16] introduced a topological method which among other things improves the lower bound. See also [18].

1.2 Octahedral Problems

The strong version of Bárány's Colourful Carathéodory Theorem says that when a colourful configuration satisfies the core condition that every point in \mathbf{S} is part of some colourful simplex. Thus the octahedral system generated by such a colourful configuration must satisfy:

Property 3 Every element of $\{1, 2, ..., d+1\}$ appears as the *i*th coordinate of some $\mathbf{v} \in \mathcal{V}$ for each $i \in \{1, 2, ..., d+1\}$.

In particular, any colourful configuration satisfying the core condition must generate a system \mathcal{V} satisfying Property 1 and Property 3. For example, the low colourful simplicial depth configurations of [10] generate such a system with (d+1) sets of (d+1) points, containing $(d^2 + 1)$ vectors. We define $\nu(d)$ to be the minimum number of vectors in an octahedral system of (d+1) points in (d+1) colours satisfying Properties 1 and 3, and $\nu^{\Diamond}(d)$ to be the minimum number of vectors of a similar system satisfying Property 1 only. Then we have $\nu(d) \leq \mu(d) \leq d^2 + 1$ and $\nu^{\diamond}(d) \leq \mu^{\diamond}(d) \leq d + 1$. In Section 2 we show that $\nu^{\diamond}(d) = \mu^{\overline{\diamond}}(d) = d + 1$. In Section 3 we show that $\nu(d) = d^2 + 1$ for d = 2, 3, and conjecture that it holds for all d. In particular, computation of $\nu(d)$ for small d gives us a finite procedure that can prove lower bounds for $\mu(d)$.

Remark 4 In [10] it was observed that $\mu(d)$ is even for odd d. Similarly it is easy to see that when m = d + 1is even, all octahedral systems have an even number of vectors. In particular, both $\nu(d)$ and $\nu^{\diamond}(d)$ are even for odd d.

2 Proof that $\mu^{\Diamond}(d) = d + 1$

A construction in [12] shows that $\nu^{\diamond}(d) \leq \mu^{\diamond}(d) \leq d+1$ for $d \geq 2$. In fact, in this section we show that any nonempty octahedral system of (d+1) sets of (d+1) points has at least (d+1) vectors, and hence that $\nu^{\diamond}(d) =$ $\mu^{\diamond}(d) = d+1$.

Proposition 5 For any $d \ge 2$, we have $\nu^{\Diamond}(d) = \mu^{\Diamond}(d) = d+1$.

Proof. Assume that there is an octahedron Ω consisting of two \hat{i} -transversals and a point $s \in \mathbf{S}_i$ such that there are an odd number of edges using points from Ω and s. Then it follows immediately from Property 1 that there is at least one edge that uses points from Ω and any point in \mathbf{S}_i . Therefore $\nu^{\Diamond}(d) \geq d+1$ as $|\mathbf{S}_i| = d+1$.

Assume then that there exists no such octahedron with odd parity, but that the system contains some edge E. We view E as being formed by a \hat{i} -transversal Tand a point $s \in \mathbf{S}_i$ and generate edges in the following way. Consider the d disjoint \hat{i} -transversals T_j for j = $1, 2, \ldots, d$ generated from the remaining points, and the d octahedra $\Omega_1, \Omega_2, \ldots, \Omega_d$ given by pairing T_j with Tfor $j = 1, 2, \ldots, d$. For each j, besides E, there is at least one other edge that uses s and the points from Ω_j due to the even parity. Therefore $\nu^{\Diamond}(d) \geq d + 1$.

In both cases $\nu^{\Diamond}(d) \ge d+1$. Thus we have $\nu^{\Diamond}(d) = \mu^{\Diamond}(d) = d+1$ as $\nu^{\Diamond}(d) \le \mu^{\Diamond}(d) \le d+1$. \Box

In Figure 2 we illustrate the 2-dimensional configuration described in [12] where $\mathbf{0} \in \operatorname{conv}(S_i \cup S_j)$ for all $i \neq j$ and is contained in exactly 3 colourful simplices. In general, the construction is to place one point of each



Figure 2: Minimal 2-dimensional configuration for the relaxed core condition.

of the first d colours below the equator in such a way that $\mathbf{0} \in \operatorname{conv}(\mathbf{S}_i)$. Then the conditions are satisfied regardless of the position of the points of \mathbf{S}_{d+1} . These points are placed near the north pole in order that each one generates a unique colourful simplex containing $\mathbf{0}$: the simplex is formed using the d points below the equator.

We remark that if we remove the condition that $|\mathbf{S}_i| \geq d + 1$ for each *i* then it is easy to modify the proof to show that **0** lies in at least $\min_i |\mathbf{S}_i|$ colourful simplices, and the example can be modified to show that this is tight.

3 Computational Approach

For a given d, the computational approach consists of ruling out a given value k for $\nu(d)$ via an exhaustive computer search showing that no system \mathcal{V} of size kcan satisfy Property 3 and Property 1. This approach was used in [12] on a laptop to show in a few seconds that $\nu(2) > 3$ and in a few hours that $\nu(3) > 8$. In other words, this approach verifies computationally that $\nu(2) = \mu(2) = 5$ and $\nu(3) = \mu(3) = 10$ – using the fact that $\nu(3)$ must be even, see Remark 4. Instances of higher dimensions are currently under computation.

In this section we propose ways to normalize the vector system which significantly speed up the enumeration. We also present a constraint programming formulation of the problem.

3.1 Normalization of vector system

Recolouring and relabelling of the points does not change the combinatorics of the point configuration. This symmetry will result in many duplicates in enumeration. In order to speed up the enumeration of vector systems for $\nu(d)$ we normalize the vector system in the following ways.

- (i) First, since the vector system \mathcal{V} is not empty, we can assume vector $(0, 0, \dots, 0) \in \mathcal{V}$.
- (ii) If there is a *covering* octahedron, i.e. one that generates an odd number of vectors for each point of the excluded colour, we can take the excluded colour to be the final one, an octahedron of the system to be $\{(0, \ldots, 0), (1, \ldots, 1)\}$, with the labellings of the points of colours $1, \ldots, d$ chosen so that (i) is satisfied.

A Python routine that searches for small octahedral systems using these normalization is available at [25].

3.2 Pivoting

We may also be able to take advantage of the following pivoting structure of octahedral systems. Given a particular \hat{i} -transversal T, we can *pivot* from the current octahedral system Ω to an *adjacent* one Ω' by removing all vectors containing T and replacing them with vectors $T \cup \{s\}$ for each $s \in \mathbf{S}_i$ such that $T \cup \{s\}$ was not in Ω .

If we have a transversal T which forms vectors with more than half the points of colour i, then pivoting on T will reduce the number of vectors in the system, although it may also break Property 3. We remark that pivoting is also seen in the setting of colourful simplicial, it corresponds to moving a point of colour i across a hyperplane defined by and \hat{i} -transversal.

3.3 Constraint programming approach

The other computational approach for $\nu(d)$ is to exploit the fact that there is a sphere covering octahedron for each missing colour and model the search for a valid vector system as a constraint programming problem.

We can start with the following collection of vectors \mathcal{V}° . Each block of (d+1) vectors represents the simplices

derived from a sphere covering octahedron for a missing colour.

$$\begin{array}{c} (1, x_{1,1}^2, x_{1,1}^3, \dots, x_{1,1}^{d+1}), \ (2, x_{1,2}^2, x_{1,2}^3, \dots, x_{1,2}^{d+1}), \ \dots, \\ (d+1, x_{1,d+1}^2, x_{1,d+1}^3, \dots, x_{1,d+1}^{d+1}); \\ (x_{2,1}^1, 1, x_{2,1}^3, \dots, x_{3,1}^{d+1}), \ (x_{2,2}^1, 2, x_{2,2}^3, \dots, x_{2,2}^{d+1}); \\ (x_{2,d+1}^1, d+1, x_{2,d+1}^3, \dots, x_{2,d+1}^{d+1}); \\ (x_{d+1,1}^1, \dots, x_{d+1,1}^d, 1), \ (x_{d+1,2}^1, \dots, x_{d+1,2}^d, 2), \ \dots, \\ (x_{d+1,d+1}^1, \dots, x_{d+1,d+1}^d, d+1). \end{array}$$

The domain of each variable is $\{1, 2, \ldots, d+1\}$. Then we have a constraint programming satisfaction problem: Given a value k, find an assignment of values to the variables such that $|\mathcal{V}^{\circ}| \leq k$ and the following constraints are satisfied:

- (1) $x_{1,1}^i = 1$ for all i and $x_{1,j}^i \in \{1,2\}$ for all i and $j \ge 2$. These constraints are derived from the normalization of the vector system.
- (2) $|\{x_{j,1}^i, x_{j,2}^i, \dots, x_{j,d+1}^i\}| \leq 2$ for all i and j because they are from an octahedron.
- (3) Constraints corresponding to Property 1.

If no solution is found, then $\nu(d) \neq k$.

4 Conclusions and remarks

Octahedral systems appear to be interesting combinatorial objects. Using the observation that colourful point configurations generate small octahedral systems, we propose a computational approach to establishing lower bounds for colourful simplicial depth. We can ask several other questions about octahedral systems.

We remark that the maximum cardinality octahedral system is the set of all possible edges; if we have m(i.e. d+1) sets of cardinality m it has size m^m . As with the other configurations discussed in this paper, it can be realized as arising from a colourful configuration of points in \mathbb{R}^d , in this case the one that places the sets $\mathbf{S}_1, \ldots \mathbf{S}_{d+1}$ close to vertices $v_1, \ldots v_{d+1}$ respectively of a regular simplex containing **0**.

Question 6 Can all octahedral systems of (d+1) sets of (d+1) points be obtained as the vectors of point configurations in \mathbb{R}^d , and can all such configurations covering all points be obtained as the vectors of configurations satisfying a core condition?

Question 7 How many octahedral systems and covering octahedral systems are there for a given m? We remark that for m = 1 we have 2 systems, 1 of which is covering, and for m = 2 we have 8 and 3; if we count only up to isomorphism these numbers are 4 and 2 respectively. **Question 8** Finally, it would be interesting to explore the pivoting structure of octahedral systems by understand its adjacency graph. For instance, we can ask about connectedness, i.e. can we get to any octahedral system from the empty octahedral system via a sequence of pivots? If so, how long must that sequence be?

We conclude by mentioning that many aspects of colourful simplices are just beginning to be explored. For instance, the combinatorial complexity of a system of colour simplices is analysed in [23]. As far as we know the algorithmic question of computing colourful simplicial depth is untouched, even for d = 2 where several interesting algorithms for computing the monochrome simplicial depth have been developed, see for instance [1], [8], [9], [13], [15] and [20].

5 Acknowledgments

The combinatorial setting used in this paper was initiated by Imre Bárány. This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and MITACS, and by the Canada Research Chairs program. We thank the anonymous referees for helpful comments.

- G. Aloupis. Geometric measures of data depth. In Data depth: robust multivariate analysis, computational geometry and applications, volume 72 of DI-MACS Ser. Discrete Math. Theoret. Comput. Sci., pages 147–158. Amer. Math. Soc., Providence, RI, 2006.
- [2] J. L. Arocha, I. Bárány, J. Bracho, R. Fabila, and L. Montejano. Very colorful theorems. *Discrete and Comput. Geom.*, 42(2):142–154, 2009.
- [3] I. Bárány. A generalization of Carathéodory's theorem. Discrete Mathematics, 40(2-3):141–152, 1982.
- [4] I. Bárány and J. Matoušek. Quadratically many colorful simplices. SIAM Journal on Discrete Mathematics, 21(1):191–198, 2007.
- [5] I. Bárány and S. Onn. Colourful linear programming and its relatives. *Math. Oper. Res.*, 22(3):550–567, 1997.
- [6] E. Boros and Z. Füredi. The number of triangles covering the center of an n-set. Geom. Dedicata, 17(1):69-77, 1984.
- [7] B. Bukh, J. Matoušek, and G. Nivasch. Stabbing simplices by points and flats. *Discrete Comput. Geom.*, 43(2):321–338, 2010.

- [8] M. A. Burr, E. Rafalin, and D. L. Souvaine. Simplicial depth: an improved definition, analysis, and efficiency for the finite sample case. In *Data depth: robust multivariate analysis, computational geometry and applications*, volume 72 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 195– 209. Amer. Math. Soc., Providence, RI, 2006.
- [9] A. Y. Cheng and M. Ouyang. On algorithms for simplicial depth. In Proceedings of the 13th Canadian Conference on Computational Geometry, pages 53–56, 2001.
- [10] A. Deza, S. Huang, T. Stephen, and T. Terlaky. Colourful simplicial depth. *Discrete and Comput. Geom.*, 35(4):597–604, 2006.
- [11] A. Deza, S. Huang, T. Stephen, and T. Terlaky. The colourful feasibility problem. *Discrete Appl. Math.*, 156(11):2166–2177, 2008.
- [12] A. Deza, T. Stephen, and F. Xie. More colourful simplices. Discrete Comput. Geom., 45(2):272–278, 2011.
- [13] K. Elbassioni, A. Elmasry, and K. Makino. Finding simplices containing the origin in two and three dimensions. *Internat. J. Comput. Geom. Appl.*, 2011. To appear.
- [14] K. Fukuda and V. Rosta. Data depth and maximal feasible subsystems. In D. Avis, A. Hertz, and O. Marcotte, editors, *Graph Theory and Combinatorial Optimization*, chapter 3, pages 37–67. Springer-Verlag, New York, 2005.
- [15] J. Gil, W. Steiger, and A. Wigderson. Geometric medians. Discrete Math., 108(1-3):37–51, 1992.
- [16] M. Gromov. Singularities, expanders and topology of maps. part 2: from combinatorics to topology via algebraic isoperimetry. *Geom. Funct. Anal.*, 20(2):416–526, 2010.
- [17] A. F. Holmsen, J. Pach, and H. Tverberg. Points surrounding the origin. *Combinatorica*, 28(6):633– 644, 2008.
- [18] R. Karasev. A simpler proof of the Boros-Füredi-Bárány-Pach-Gromov theorem. Discrete Comput. Geom., 2011. To appear.
- [19] F. Kárteszi. Extremalaufgaben über endliche Punktsysteme. Publ. Math. Debrecen, 4:16–27, 1955.
- [20] S. Khuller and J. S. B. Mitchell. On a triangle counting problem. *Inf. Process. Lett.*, pages 319– 321, 1990.

- [21] R. Y. Liu. On a notion of data depth based on random simplices. Ann. Statist., 18(1):405–414, 1990.
- [22] E. Rafalin and D. L. Souvaine. Computational geometry and statistical depth measures. In *Theory and applications of recent robust methods*, Stat. Ind. Technol., pages 283–295. Birkhäuser, Basel, 2004.
- [23] A. Schulz and C. D. Tóth. The union of colorful simplices spanned by a colored point set. In CO-COA (1), pages 324–338, 2010.
- [24] T. Stephen and H. Thomas. A quadratic lower bound for colourful simplicial depth. J. Comb. Opt., 16(4):324–327, 2008.
- [25] F. Xie. Python code for octrahedral system computation. available at: http://optlab.mcmaster.ca/om/csd/.

Combinatorics of Minkowski decomposition of associahedra

Carsten E. M. C. Lange*

Abstract

Realisations of associahedra can be obtained from the classical permutahedron by removing some of its facets and the set of these facets is determined by the diagonals of certain labeled convex planar n-gons as shown by Hohlweg and Lange (2007). Ardila, Benedetti, and Doker (2010) expressed polytopes of this type as Minkowski sums and differences of dilated faces of a standard simplex and computed the corresponding coefficients y_I by Möbius inversion. Given an associahedron of Hohlweg and Lange, we give a new combinatorial interpretation of the values y_I : they are the product of two signed lengths of paths of the labeled n-gon. We also discuss an explicit realisation of a cyclohedron to show that the formula of Ardila, Benedetti, and Doker does not hold for generalised permutahedra not in the deformation cone of the classical permutahderon.

1 Introduction

Consider the convex (n-1)-dimensional polytope

$$P_n(\{z_I\}) := \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid \frac{\sum_{i \in [n]} x_i = z_{[n]} \text{ and}}{\sum_{i \in I} x_i \ge z_I \text{ for } \varnothing \subset I \subset [n]} \right\},$$

where [n] denotes the set $\{1, 2, \dots, n\}$. The classical permutahedron, as described for example by G. M. Ziegler, [21], corresponds to $z_I = \frac{|I|(|I|+1)}{2}$ for $\emptyset \subset I \subseteq [n]$. Generalised permutahedra were first studied by A. Postnikov, [14]. They are polytopes $P_n(\{z_I\})$ and are contained in the deformation cone of the classical permutahedron, [15]. We focus our study on special realisations of associahedra denoted by As_{n-1}^{c} , which form a subclass of generalised permutahedra. Two examples of 3-dimensional polytopes As_3^c are shown in Figure 1. In Section 5, we give an example to explain the notion of a deformation cone and to show that the approach to compute the coefficients of the Minkowski decomposition fails for polytopes $P_n(\{z_I\})$ not contained in the deformation cone of the classical permutahedron.

The Minkowski sum of two polytopes P and Q is defined as $\{p + q \mid p \in P, q \in Q\}$. On the other hand, we define the Minkowski difference P - Q of polytopes P and Q if and only if there is a polytope R such that P = Q + R, for more details on Minkowski differences we refer to [18]. We are interested in decompositions of As_{n-1}^c into Minkowski sums and differences of dilated faces of the (n-1)-dimensional standard simplex $\Delta_n = \operatorname{conv} \{e_1, e_2, \cdots, e_n\},$

where e_i is a standard basis vector of \mathbb{R}^n . The faces Δ_I of Δ_n are given by $\operatorname{conv}\{e_i\}_{i\in I}$ for $I \subseteq [n]$. If a polytope P is the Minkowski sum and difference of dilated faces of Δ_n , we say that P has a Minkowski decomposition into faces of the standard simplex. The following is a general result on Minkowski decompositions of a generalised permutahedron $P(\{z_I\})$ where we assume that the values z_I for redundant inequalities of $P(\{z_I\})$ are tight.

Proposition 1 ([1, Proposition 2.3])

Every generalised permutahedron $P_n(\{z_I\})$ can be written uniquely as a Minkowski sum and difference of faces of Δ_n :

$$P_n(\{z_I\}) = \sum_{I \subseteq [n]} y_I \Delta_I$$

where $y_I = \sum_{J \subseteq I} (-1)^{|I \setminus J|} z_J$ for each $I \subseteq [n]$.

To put it differently, the functions $I \mapsto z_I$ and $I \mapsto y_I$ of the boolean lattice are Möbius inverses. A weaker version of Proposition 1 that requires $y_I \ge 0$ for all $I \subseteq [n]$ was established by A. Postnikov, [14]. Obviously, the formula of Proposition 1 is computationally expensive in general. The formula describes a beautiful relation between the z_I - and y_I -coordinates of generalised permutahedra, but there is more hidden. The author showed that the formula for y_I of Proposition 1



Figure 1: Two different realisations As_3^c according to [8] after application of an orthogonal transformation. The realisations correspond to different labelings of a hexagon and have distinct Minkowski decompositions into dilated faces of the standard simplex.

^{*}Fachbereich Mathematik und Informatik, Freie Universität Berlin, clange@math.fu-berlin.de, partially supported by a DFG-grant (Forschergruppe 565 *Polyhedral Surfaces*)

simplifies to four terms for all I if $P(\{z_I\}) = As_{n-1}^c$, see Theorem 2 and [11]. But even better: we do not even have to compute the four values z_J that remain after simplification, the multiplication of two (signed) numbers of edges connecting points on the boundary of a polygon suffices. The precise statement is given in Theorem 4.

We end this introduction with some general remarks. S. Fomin and A. Zelevinsky introduced generalised associahedra in the context of cluster algebras of finite type, [5], and it is known that associahedra and generalised associated to cluster algebras of type A are combinatorially equivalent. The construction of [8] was generalised by C. Hohlweg, C. Lange, and H. Thomas to generalised associahedra, [9]. The construction depends on choosing a Coxeter element cand the normal vectors of the facets are determined by combinatorial properties of c. Since the normal fans of these realisations turn out to be Cambrian fans as described by N. Reading and D. Spever, [16], the obtained realisations are generalised associahedra associated to some cluster algebra of finite type. N. Reading and D. Speyer conjectured a linear isomorphism between Cambrian fans and g-vector fans associated to cluster algebras of finite type with acyclic initial seed introduced by S. Fomin and A. Zelevinsky, [6]. They proved their conjecture up to an assumption of another conjecture of [6]. In 2008, S.-W. Yang and A. Zelevinsky gave an alternative proof of the conjecture of Reading and Speyer, [20]. We remark in this context that the results of Section 2 and 3 of [11] can be read along these lines: the computations of z_I and y_I for fixed Iand varying c involve sums over different choices of $\tilde{z}_{R_s}^c$ where the diagonals δ that have to be considered depend on c. Moreover, the values for \tilde{z}_{Rs}^c that occur in these sums should be tight for the polytope but can be choosen within a large class of possible values as described for example in [9], not just the specific value chosen here in Section 2. The formula of Theorem 4 of this manuscript could be rewritten in this sense by introducing extra parameters. From this point of view, we suggest that combinatorial properties of the q-vector fan for cluster algebras of finite type A with respect to an acyclic initial seed are reflected by the Minkowski decompositions studied in [11] and in this manuscript.

Some instances of As_{n-1}^c have been studied earlier. For example, J.-L. Loday computes vertex coordinates from planar binary trees, [12]. This generalises $\mathsf{As}_2^{c_1}$ studied in Section 3 to higher dimensions. G. Rote, F. Santos, and I. Streinu relate associahedra to onedimensional point configurations, [17]. Both realisations are affinely equivalent to As_{n-1}^c if $\mathsf{U}_c = \emptyset$ or $\mathsf{U}_c = [n] \setminus \{1, n\}$. Moreover, Rote et.al. point out that a realisation of F. Chapoton, S. Fomin, and A. Zelevinsky, [4], is not affinely equivalent to their realisation. But in fact, it is affinely equivalent to some As_{n-1}^{c} , i.e. $U_{c} = \{2\}$ or $U_{c} = \{3\}$ for n = 4. F. Santos and V. Pilaud recently constructed a family of polytopes called *brick polytopes* that are related to multitrangulations, [13]. As a special case, they obtain translates of the associahedra As_{n-1}^{c} studied in this paper. They describe brick polytopes as Minkowski sums of brick polytopes and in particular, they achieve a Minkowski decomposition different from ours. The precise relation of these two decompositions is not clear at the time of writing.

2 The associahedra As_{n-1}^c

Associahedra form a class of combinatorially equivalent simple polytopes and can be realised as generalised permutahedra. They are often defined by specifying their 1-skeleton or graph. A theorem of G. Kalai, [10], implies that the face lattice of an (n-1)-dimensional associahedron As_{n-1} is completely determined by this graph. Now, the graph of an associahedron is isomorphic to a graph with all triangulations (without new vertices) of a convex and plane (n+2)-gon Q as vertex set and all pairs of distinct triangulations that differ in precisely one proper diagonal¹ as edge set. Alternatively, the edges of As_{n-1} are in bijection with the set of triangulations with one proper diagonal removed. Similarly, k-faces of As_{n-1} are in bijection to triangulations of Q with k proper diagonals deleted. In particular, the facets of As_{n-1} are in bijection with proper diagonals of Q. J.-L. Loday published a beautiful algorithm to obtain explicit vertex coordinates for associahedra from planar binary trees, [12]. This algorithm was generalised by C. Hohlweg and C. Lange and explicitly describes realisations of As_{n-1} as generalised permutahedra that depend on combinatorics induced by the choice of a Coxeter element c of the symmetric group Σ_n on n elements, [8]. A Coxeter element is a permutation obtained by multiplying the generators of S_n in some order.

We now outline the construction of [8] and avoid to use Coxeter elements explicitly. Nevertheless, we use them to distinguish different realisations in our notation. The choice of a Coxeter element c corresponds to a partition of [n] into a *down set* D_c and an *up set* U_c :

and

 $\mathsf{U}_c = \{ u_1 < u_2 < \dots < u_m \}.$

 $\mathsf{D}_c = \{ d_1 = 1 < d_2 < \dots < d_\ell = n \}$

This partition induces a labeling of the vertices of Qwith label set $[n+1]_0 := [n+1] \cup \{0\}$ as follows. Pick

¹A proper diagonal is a line segment connecting a pair of vertices of Q whose relative interior is contained in the interior of Q. A non-proper diagonal is a diagonal that connects vertices adjacent in ∂Q and a degenerate diagonal is a diagonal where the end-points are equal.

two vertices of Q which are the end-points of a path of $\ell + 2$ vertices on the boundary of Q, label the vertices of this path counter-clockwise increasing using the label set $D_c := D_c \cup \{0, n+1\}$ and label the remaining path clockwise increasing using the label set U_c . Without loss of generality, we shall always assume that the label set D_c is to the right of the diagonal $\{0, n+1\}$ oriented from 0 to n + 1, examples are given in Section 3. We derive values z_I for some subsets $I \subset [n]$ obtained from this labeled (n+2)-gon Q using proper diagonals of Q as follows. Orient each proper diagonal δ from the smaller to the larger labeled end-point of δ , associate to δ the set R_{δ} that consists of all labels on the strict right-hand side of δ , and replace the elements 0 and n+1 by the smaller respectively larger label of the end-points contained in U_c if possible. For each proper diagonal δ we have $R_{\delta} \subseteq [n]$ but obviously not every subset of [n] is of this type if n > 2. We set

$$\tilde{z}_{I}^{c} := \begin{cases} \frac{|I|(|I|+1)}{2} & \text{if } I = R_{\delta}, \, \delta \text{ proper diagonal,} \\ -\infty & \text{else.} \end{cases}$$

In [8] it is shown that $P_n(\{\tilde{z}_I^c\})$ is in fact an associahedron of dimension n-1 realised in \mathbb{R}^n for every choice of c and the inequalities that correspond to finite values \tilde{z}_I^c are precisely the non-redundant facet-defining inequalities of As_{n-1}^c . This ends the summary of results found in [8].

To compute the coefficients of the Minkowski decomposition of $\operatorname{As}_{n-1}^c$ according to Proposition 1, we have to find tight values for z_I that correspond to all inequalities (redundant and non-redundant) first. Fortunately enough, this is not necessary. As outlined by the author in an extended abstract, [11], it suffices to know the finite values of \tilde{z}_I^c defined above. To state and prove Theorem 4, we have to review some facts from [11] and start with two key definitions given there.

Suppose from now on that $[n] = \mathsf{D}_c \sqcup \mathsf{U}_c$ is a partition of [n] induced by a Coxeter element c with

$$\mathsf{D}_c = \{ d_1 = 1 < d_2 < \dots < d_\ell = n \}$$

and

$$\mathsf{U}_{c} = \{ u_{1} < u_{2} < \dots < u_{m} \}.$$

Definition 1 (up and down intervals)

- (a) A set $S \subseteq [n]$ is a non-empty interval of [n] if $S = \{r, r+1, \cdots, s\}$ for some $0 < r \le s < n$. We write S as closed interval [r, s] (end-points included) or as open interval (r-1, s+1) (end-points not included). An empty interval is an open interval (k, k+1) for some $1 \le k < n$.
- (b) A non-empty open down interval is a set $S \subseteq D_c$ such that $S = \{d_r < d_{r+1} < \cdots < d_s\}$ for some $1 \leq r \leq s \leq \ell$. We write S as open down interval $(d_{r-1}, d_{s+1})_{D_c}$ where we allow $d_{r-1} = 0$ and $d_{s+1} = n + 1$, i.e. $d_{r-1}, d_{s+1} \in \overline{D}_c$. For

 $1 \leq r \leq \ell - 1$, we have the empty down interval $(d_r, d_{r+1})_{\mathsf{D}_c}$.

(c) A closed up interval is a non-empty set $S \subseteq U_c$ such that $S = \{u_r < u_{r+1} < \cdots < u_s\}$ for some $1 \leq r \leq s \leq m$. We write $[u_r, u_s]_{U_c}$.

We emphasize that up intervals are always non-empty, while down intervals may be empty. Moreover, it turns out to be convenient to distinguish the empty down intervals $(d_r, d_{r+1})_{\mathsf{D}_c}$ and $(d_s, d_{s+1})_{\mathsf{D}_c}$ if $r \neq s$ although they are equal as sets.

Definition 2 (up & down interval decomposition) Let I be a non-empty subset of [n].

- (a) An up and down interval decomposition of type (v, w) of I is a partition of I into disjoint up and down intervals $I_1^{\mathsf{U}}, \dots, I_w^{\mathsf{U}}$ and $I_1^{\mathsf{D}}, \dots, I_v^{\mathsf{D}}$ obtained by the following procedure.
 - 1. Suppose there are \tilde{v} non-empty inclusion maximal down intervals contained in I that we denote by $\tilde{I}_k^{\mathsf{D}} = (\tilde{a}_k, \tilde{b}_k)_{\mathsf{D}_c}, 1 \leq k \leq \tilde{v}, \text{ with } \tilde{b}_k \leq \tilde{a}_{k+1}$ for $1 \leq k < \tilde{v}$. Let $E_i^{\mathsf{D}} = (d_{r_i}, d_{r_i+1})_{\mathsf{D}_c}$ denote all empty down intervals with $\tilde{b}_k \leq d_{r_i} < d_{r_i+1} \leq$ \tilde{a}_{k+1} for $0 \leq k \leq \tilde{v}, \tilde{b}_0 = 0$, and $\tilde{a}_{\tilde{v}+1} = n + 1$. Denote the open intervals $(\tilde{a}_i, \tilde{b}_i)$ and (d_{r_i}, d_{r_i+1}) of [n] by \tilde{I}_i and E_i respectively.
 - 2. Consider all up intervals of I which are contained in (and inclusion maximal within) some interval \tilde{I}_i or E_i obtained in Step 1 and denote these up intervals by

$$I_1^{\mathsf{U}} = [\alpha_1, \beta_1]_{\mathsf{U}_c}, \cdots, I_w^{\mathsf{U}} = [\alpha_w, \beta_w]_{\mathsf{U}_c}.$$

We assume $\alpha_i \leq \beta_i < \alpha_{i+1}$.

- 3. A down interval $I_i^{\mathsf{D}} = (a_i, b_i)_{\mathsf{D}_c}, 1 \leq i \leq w$, is a down interval obtained in Step 1 that is either a non-empty down interval \tilde{I}_k^{D} or an empty down interval E_k^{D} with the additional property that there is some up interval I_j^{U} obtained in Step 2 such that $I_j^{\mathsf{U}} \subseteq E_k$. Without loss of generality, we assume $b_i \leq a_{i+1}$ for $1 \leq i < w$.
- (b) An up and down interval decomposition of type(1,w) is called nested. A nested component of I is an inclusion-maximal subset J of I such that the up and down decomposition of J is nested.

The up and down interval decomposition of $I \subseteq [n]$ enables us to compute tight values \tilde{z}_I^c of $\operatorname{As}_{n-1}^c$ for all Iusing only \tilde{z}_I^c that correspond to non-redundant inequalities. These values can be substituted in the formula for y_I of Proposition 1 and the formula can be simplified significantly. Before we state the resulting theorem, it makes sense to extend our notion of R_{δ} and $\tilde{z}_{R_{\delta}}^c$ to non-proper and degenerate diagonals δ .

For a diagonal $\delta = \{x, y\}$ that is not proper, we set

$$R_{\delta} := \begin{cases} \varnothing & \text{if } x, y \in \overline{\mathsf{D}}_c \\ [n] & \text{otherwise,} \end{cases}$$

and

$$\tilde{z}_{R_{\delta}}^{c} := \begin{cases} 0 & \text{if } R_{\delta} = \emptyset \\ \frac{n(n+1)}{2} & \text{if } R_{\delta} = [n]. \end{cases}$$

Let $I \subseteq [n]$ be a non-empty subset with up and down interval decomposition of type (v, k). If I has a nested up and down interval decomposition, then, in particular, v = 1 and

$$I = (a, b)_{\mathsf{D}_c} \cup \bigcup_{i=1}^k [\alpha_i, \beta_i]_{\mathsf{U}_c}$$

with $\alpha_k < \beta_k \leq \alpha_{k+1}$ as before. In this situation, we denote the smallest (respectively largest) element of I by γ (respectively Γ) and consider the diagonals

$$\delta_1 := \{a, b\}, \qquad \delta_2 := \{a, \Gamma\},$$

$$\delta_3 := \{\gamma, b\}, \text{ and } \qquad \delta_4 := \{\gamma, \Gamma\}.$$

We can now state the main result of [11] which we use to prove Theorem 4.

Theorem 2 ([11, Theorem 3.1])

Let I be a non-empty subset of [n] with a nested up and down interval decomposition of type (1, k). Then

$$y_I = (-1)^{|I \setminus R_{\delta_1}|} \left(z_{R_{\delta_1}}^c - z_{R_{\delta_2}}^c - z_{R_{\delta_3}}^c + z_{R_{\delta_4}}^c \right)$$

Corollary 3 ([11, Corollary 3.2])

Let I be a non-empty subset of [n] with an up and down interval decomposition of type (v, k) and v > 1. Then $y_I = 0$.

3 Main theorem and examples

We continue to use the notation introduced in the previous section. Moreover, we need the notion of signed lengths K_{γ} and K_{Γ} for sets I with interval decomposition of type (1, k) that is needed in Step 2. (b) of Theorem 4. They denote integers and have the following meaning: $|K_{\Gamma}|$ is the length, i.e. the number of edges, of the path in ∂Q connecting b and Γ that does not use the vertex labeled a and K_{Γ} is negative if and only if $\Gamma \in (a, b)_{\mathbb{D}}$. Similarly, $|K_{\gamma}|$ is the length of the path connecting a and γ that does not use the vertex labeled band K_{γ} is negative if and only if $\gamma \in (a, b)_{\mathbb{D}}$.

Theorem 4 Let Q be the (n + 2)-gon labeled according to the construction of As_{n-1}^c and $I \subseteq [n]$ be non-empty. To compute y_I perform the following two steps:

- 1. Determine the type (v, w) of the up and down interval decomposition of I.
- 2. (a) If v > 1 then $y_I = 0$. (b) If v = 1 then

$$\begin{split} y_I &= (-1)^{|I \setminus (a,b)_{\mathsf{D}}|} \left(K_{\gamma} K_{\Gamma} - (n+1) \right) \\ \text{if } |I| &= 1 \text{ and } I \subseteq \mathsf{U}, \text{ while} \\ y_I &= (-1)^{|I \setminus (a,b)_{\mathsf{D}}|} K_{\gamma} K_{\Gamma} \end{split}$$

otherwise.



Figure 2: The Minkowski decomposition of the 2dimensional associahedron $As_2^{c_1}$ into faces of the standard simplex is in fact a Minkowski sum.

Theorem 4 is the third to relate combinatorics of labeled *n*-gons to different aspects of realisations of associahedra. Firstly, the coordinates of the vertices can be extracted, [12, 8]. Secondly, the facet normals and the right-hand sides for their inequalities can be read off, [8]. Thirdly, the coefficients of a Minkowski decomposition are obtained according to Theorem 4.

Before giving the proof, we give an example of two 2-dimensional associahedra $As_2^{c_1}$ and $As_2^{c_2}$. The first example $As_2^{c_1}$ corresponds to $D_{c_1} = [n]$ and $U_{c_1} = \emptyset$. Minkowski decompositions of $As_2^{c_1}$ and its higher dimensional analogues were already studied earlier as mentioned by A. Postnikov and it is known that $y_I \in \{0, 1\}$, so these polytopes are actually a Minkowski sum of faces of the standard simplex. We have

$$\mathsf{As}_2^{c_1} = \Delta_{\{1\}} + \Delta_{\{2\}} + \Delta_{\{3\}} + \Delta_{\{1,2\}} + \Delta_{\{2,3\}} + \Delta_{\{1,2,3\}},$$

see Figure 2. Although $As_2^{c_2}$ is isometric to $As_2^{c_1}$, it does not decompose into a Minkowski sum of dilated faces of a standard simplex but into a Minkowski sum and difference of dilated faces of the standard simplex:

$$\mathsf{As}_{2}^{c_{2}} = \begin{pmatrix} \Delta_{\{1\}} + \Delta_{\{3\}} + 2 \cdot \Delta_{\{1,2\}} \\ + \Delta_{\{1,3\}} + 2 \cdot \Delta_{\{2,3\}} \end{pmatrix} - \Delta_{\{1,2,3\}},$$

see Figure 3. The up and down sets in this situation are

 $U_c = \{2\}$ and $D_c = \{1, 3\},\$

so we obtain the following labeled pentagon Q:



We now compute the coefficients $y_{\{2\}}$, $y_{\{1,2\}}$, and $y_{\{1,2,3\}}$ in order to demonstrate Theorem 4.

The up and down interval decompositions for $\{2\}$, $\{1,2\}$, and $\{1,2,3\}$ are of type (1,1):

$$\begin{split} \{2\} &= (1,3)_D \sqcup [2,2]_{\mathsf{U}}, \\ \{1,2\} &= (0,3)_D \sqcup [2,2]_{\mathsf{U}}, \\ 1,2,3\} &= (0,4)_D \sqcup [2,2]_{\mathsf{U}}. \end{split}$$

Hence we obtain the following table:

{



Figure 3: The Minkowski decomposition of the 2dimensional associahedron $As_2^{c_2}$ into dilated faces of the standard simplex.

Ι	a	b	γ	Γ	K_{γ}	K_{Γ}	$ I \setminus (a,b)_{D} $
$\{2\}$	1	3	2	2	2	2	1
$\{1,2\}$	0	3	1	2	-1	2	1
$\{1,2,3\}$	0	4	1	3	-1	-1	1

Since n = 3 in this example, we compute

$$y_{\{2\}} = (-1)^1 (2 \cdot 2 - (3+1)) = 0,$$

$$y_{\{1,2\}} = (-1)^1 \cdot (-1) \cdot 2 = 2,$$

$$y_{\{1,2,3\}} = (-1)^1 \cdot (-1) \cdot (-1) = -1.$$

4 Proof of the main theorem

The strategy of the proof is clear: Suppose $I \subseteq [n]$ is non-empty, we compute the up and down interval decomposition (Step 1. of Theorem 4) and then reinterpret Theorem 2 and Corollary 3 in terms of K_{γ} and K_{Γ} . If the up and down decomposition of I is of type (v, w)with $v \ge 2$ then the claim of Step 2. (a) follows immediately from Corollary 3. We therefore assume that I has an up and down interval decomposition of type (1, k), the associated down interval is $(a, b)_D$ and the minimal and maximal elements of I are γ and Γ . We also use the notation of δ_i , $1 \le i \le 4$, from Section 2 and define

$$\widetilde{K}_{\Gamma}:=|R_{\delta_2}|-|R_{\delta_1}| \quad \text{as well as} \quad \widetilde{K}_{\gamma}:=|R_{\delta_3}|-|R_{\delta_1}|.$$

A simple case-by-case analysis shows

- 1. $K_{\gamma} > 0$ if and only if $\gamma \in \mathsf{U}_c$.
- 2. $\widetilde{K}_{\Gamma} > 0$ if and only if $\Gamma \in \mathsf{U}_c$.
- 3. $\widetilde{K}_{\gamma} = -1$ if and only if $\gamma \in \mathsf{D}_c$.
- 4. $\widetilde{K}_{\Gamma} = -1$ if and only if $\Gamma \in \mathsf{D}_c$.

as well as $K_{\Gamma} = \widetilde{K}_{\Gamma}$ and $K_{\gamma} = \widetilde{K}_{\gamma}$. We additionally define $K := |R_{\delta_1}|$ and a direct computation allows to express $z_{R_{\delta_1}}^c$, $1 \le i \le 3$, in terms of K, K_{Γ} , and K_{γ} :

$$\begin{split} z^c_{R_{\delta_1}} &= \frac{K(K+1)}{2}, \\ z^c_{R_{\delta_2}} &= \frac{(K+K_{\Gamma})(K+K_{\Gamma}+1)}{2}, \\ \text{and} \ z^c_{R_{\delta_3}} &= \frac{(K+K_{\gamma})(K+K_{\gamma}+1)}{2}. \end{split}$$

Another direct computation yields

$$K_{\Gamma}K_{\gamma} = z_{R_{\delta_1}} - z_{R_{\delta_2}} - z_{R_{\delta_3}} + \frac{(K + K_{\Gamma} + K_{\gamma})(K + K_{\Gamma} + K_{\gamma} + 1)}{2}$$

To express $\tilde{z}_{R_{\delta_4}}^c$ in terms of K, K_{Γ} , and K_{γ} , we observe

$$\begin{split} \frac{(K+K_{\Gamma}+K_{\gamma})(K+K_{\Gamma}+K_{\gamma}+1)}{2} \\ &= \begin{cases} \frac{|R_{\delta_4}|(|R_{\delta_4}|+1)}{2} & \text{if } I \neq \{u_s\}, \\ \frac{|R_{\delta_4}|(|R_{\delta_4}|+1)}{2} + (n+1) & \text{if } I = \{u_s\}, \end{cases} \end{split}$$

and obtain

$$z_{R_{\delta_4}}^c = \frac{(K + K_{\Gamma} + K_{\gamma})(K + K_{\Gamma} + K_{\gamma} + 1)}{2}$$

if $I \neq \{u_s\}$, and

$$z_{R_{\delta_4}}^c = \frac{(K + K_{\Gamma} + K_{\gamma})(K + K_{\Gamma} + K_{\gamma} + 1)}{2} - (n+1)$$

if $I = \{u_s\}$. The claim follows now from Theorem 2.

5 A Remark on Cyclohedra

Cyclohedra are also known as Bott-Taubes polytopes or type *B* generalised associahedra, [3, 4, 19]. They can be realised using some As_{n-1}^c by intersection with *type B hyperplanes* $x_i + x_{2n+1-i} = 2n + 1$, $1 \le i < n$. We refer to [8] for details. A 2-dimensional cyclohedron Cy_2^c obtained from some As_3^c by intersection with $x_1 + x_4 = 5$ is shown in Figure 4. Tight right-hand sides for Cy_2^c are the right-hand sides of As_2^c except $z_{\{1,4\}}$ and $z_{\{2,3\}}$ whose tight value is 5 instead of 2. The inequalities $x_1 + x_4 \ge 2$ and $x_2 + x_3 \ge 2$ are redundant for As_2^c and altering the level sets for these inequalities from 2 (for As_2^c) to 5 (for Cy_2^c) means that we move past the four vertices *A*, *B*, *C*, and *D*. As explained in [15], this



Figure 4: A 2-dimensional cyclohedron Cy_2^c (indicated in black) obtained from an associahedron As_3^c by intersection with type *B* hyperplanes.

implies that Cy_2^c is not in the deformation cone of the classical permutahdron. Applying Proposition 1 to the function z_I on the boolean lattice for Cy_2^c , we compute the Möbius inverse y_I . We obtain

$$\mathsf{Cy}_2^c + \left(\begin{smallmatrix} \Delta_2 + 4\Delta_{123} \\ + 3\Delta_{124} + 2\Delta_{134} + \Delta_{234} \end{smallmatrix}\right) = \left(\begin{smallmatrix} \Delta_1 + \Delta_3 + \Delta_4 + 3\Delta_{12} + \Delta_{13} \\ + 3\Delta_{14} + 5\Delta_{23} + \Delta_{34} + 5\Delta_{1234} \end{smallmatrix}\right)$$

if Proposition 1 were true for polytopes $P_n(\{z_I\})$ not contained in the deformation cone of the classical permutahedron. One way to see that this equation does not hold is to compute the number of vertices of the polytope on the left-hand side (27 vertices) and on the right-hand side (20 vertices) using polymake, [7].

6 Concluding remarks

There are some questions related to the coefficients y_I . Firstly, how do Minkowski decompositions of generalised associahedra obtained in [9] look like and how can we compute them if they exit? In particular, how to decompose the cyclohedron of [8]?

Secondly, the computation of the Minkowski decomposition depends on a good understanding of computational aspects of Möbius inversions. Efficient computations of Möbius functions on lattices were studied by A. Blass and B. Sagan, [2]. Does this extend somehow to Möbius inversion? Moreover, Möbius inversions are often used in proofs but because of the computational complexity rarely used in computations. Is there some theory to compute the Möbius inverse more efficiently?

- F. Ardila, C. Benedetti, and J. Doker. Matroid polytopes and their volumes. *Discrete Comput Geom*, 43:841–854, 2010.
- [2] A. Blass and B. E. Sagan. Möbius functions of lattices. Adv Math, 127:94–123, 1997.
- [3] R. Bott and C. Taubes. On the self-linking of knots. J. Math. Phys., 35:5247–5287, 1994.
- [4] F. Chapoton, S. Fomin, and A. Zelevinksy. Polytopal realizations of generalized associahedra. *Canad Math Bull*, 45:537–566, 2002.
- [5] S. Fomin and A. Zelevinksy. Y-systems and generalized associahedra. Ann. of Math., 158:977–1018, 2003.
- [6] S. Fomin and A. Zelevinksy. Cluster algebras iv. coefficients. *Compos Math*, 143:112–164, 2007.
- [7] E. Gawrilow and M. Joswig. polymake: a framework for analyzing convex polytopes. In G. Kalai and G. M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

- [8] C. Hohlweg and C. Lange. Realizations of the associahedron and cyclohedron. *Discrete Comput Geom*, 37:517–543, 2007.
- [9] C. Hohlweg, C. Lange, and H. Thomas. Permutahedra and generalized associahedra. Adv Math, 226:608–640, 2011.
- [10] G. Kalai. A simple way to tell a simple polytope from its graph. J. Combin Theory Ser A, 49:381– 383, 1988.
- [11] C. Lange. Minkowski decompositions of associahedra (extended abstract). FPSAC 2011, pages 611– 622, 2011.
- [12] J.-L. Loday. Realizations of the Stasheff polytope. Arch Math, 83:267–278, 2004.
- [13] V. Pilaud and F. Santos. The brick polytope of a sorting network. *Proceedings of FPSAC 2011*, pages 777–788.
- [14] A. Postnikov. Permutahedra, associahedra, and beyond. Int Math Res Not, pages 1026–1106, 2009.
- [15] A. Postnikov, V. Reiner, and L. Williams. Faces of generalized permutahedra. *Documenta Mathematica*, 13:207–273, 2008.
- [16] N. Reading and D. Speyer. Cambrian fans. J Europ Math Soc, 11:411–447, 2009.
- [17] G. Rote, F. Santos, and I. Streinu. Expansive motions and the polytope of pointed pseudotriangulations. Number 25 in Algorithms and Combinatorics, pages 699–736. Springer-Verlag, Berlin, 2003.
- [18] R. Schneider. Convex bodies: the Brunn-Minkowski theory, volume 44 of Encyclopedia of Mathematics and Applications. Cambridge University Press, Cambridge, 1993.
- [19] R. Simion. A type-B associahedron. Adv. Appl. Math., 30:2–25, 2003.
- [20] S.-W. Yang and A. Zelevinksy. Cluster algebras of finite type via coxeter elements and principal minors. *Transform Groups*, 13:855–895, 2008.
- [21] G. M. Ziegler. Lectures on Polytopes, volume 152 of Graduate Texts in Mathematics. Springer-Verlag, Heidelberg, 1998.

A Fourier-Theoretic Approach for Inferring Symmetries

Xiaoye Jiang^{*}

Jian Sun[†]

Leonidas Guibas[‡]

Abstract

In this paper, we propose a novel Fourier-theoretic approach for estimating the symmetry group \mathbb{G} of a geometric object X. Our approach takes as input a geometric similarity matrix between low-order combinations of features of X and then searches within the tree of all feature permutations to detect the sparse subset that defines the symmetry group \mathbb{G} of X. Using the Fourier-theoretic approach, we construct an efficient marginal-based search strategy, which can recover the symmetry group \mathbb{G} effectively. The framework introduced in this paper can be used to discover symmetries of more abstract geometric spaces and is robust to deformation noise. Experimental results show that our approach can fully determine the symmetries of many geometric objects.

1 Introduction

Symmetries are extremely common in both man-made and natural objects. In the context of computational geometry, we often consider the symmetry group of a geometric object with a pre-defined metric. One easy way of describing all symmetries of geometric objects is to look at *group actions*, where we use a set to represent the object, and the symmetries of the object are described by bijective mappings on the set. In this paper, assume we have a discrete set $X = \{x_i\}_{i=1}^n$ which describes a geometric object. As shown in Figure 1-(a), the five tip points on a star can be used as a discrete set X to study the symmetry of such a 3D star model. This is because each symmetry of the star model can be identified with a permutation of the elements in X^1 . For convenience, we say a permutation is "good" if it can be identified as a symmetry of the geometric object. It can be shown that all good permutations of X consist a group \mathbb{G} , which is the symmetry group of X.

In practice, we are often limited to verifying low-order information about X, such as the similarity of curvatures between pairs of points (first order), or the consistency of distances between pairs of pairs of points (second order). But how can we integrate such low-order pieces of symmetry evidence together to identify all the good permutations of X and derive its symmetry group? This question is quite challenging since the space of all permutations grows factorially with the number of elements in X so that directly searching among all permutations is impossible, unless n is small. In this paper, we propose a Fourier-theoretic approach to address this problem, based on low-order similarities of points in X.

The symmetry group of X is a subgroup of the permutation group \mathbb{S}_n , where n = |X|. To search for \mathbb{G} , we naturally have the following simple strategy: we organize the elements of \mathbb{S}_n in a tree where each node represents all the permutations in its sub-tree, and then search those in \mathbb{G} within the tree, see Figure 1-(b) for an illustration of the tree. Whenever we reach a permutation of X, we check whether it is a good one. However, such a brute force strategy would be computationally intractable for all but very small n.

In this paper, we propose a novel search strategy within the tree of S_n , called the marginal probability search, which fully exploits the algebraic structure of the groups \mathbb{G} and S_n . The main contribution of the paper are the following two ways of making use of algebraic structures to facilitate the search of symmetries.

Firstly, we consider the symmetry group \mathbb{G} as an *in*dicator distribution (see Theorem 1 in Section 2) over the permutation group S_n . This novel point of view enables us to utilize techniques from the group representation theory to convert low-order information into a set of Fourier coefficients which characterizes the low-frequency components of the distribution over \mathbb{S}_n . Those Fourier coefficients can thus be used to efficiently estimate the marginal probability of the permutations represented by an internal node, which serves as the criterion for pruning the sub-tree rooted at that node. Unlike other traditional pruning criteria [4], the marginal probability is much more informative as it not only evaluates the part of the permutation which is already determined but also summarizes the remaining part which is undecided, and thus provides a more efficient pruning.

Secondly, we exploit the group structure of \mathbb{G} and show that the internal nodes on the same level have either the same marginal probability as the node containing the identity permutation or 0 marginal probability for the indicator distribution \mathbb{G} . This ensures the correctness of taking the marginal probability of the in-

^{*}Institute of Computational and Mathematical Engineering, Stanford University, xiaoyej@stanford.edu

[†]Mathematical Sciences Center, Tsinghua University, jsun@math.tsinghua.edu.cn

[‡]Department of Computer Science, Stanford University, guibas@cs.stanford.edu

¹Choosing different set X can result in different group actions. In this paper, however, we assume such a set X is given where each symmetry can be identified as permuting X.

ternal nodes as the reference for pruning.

The approach proposed in this paper generalizes the existing work on graph automorphism, in the sense that we can deal with noisy similarity information caused by heavy distortion or deformation of the geometric object and robustly estimate the symmetry group G from the input. Moreover, any arbitrary order of similarities, e.g., triple-wise or even higher order similarities can be taken as input in our framework. Different orders of similarities can be combined together easily because Fourier analysis can fully disentangle and decompose the information over permutations of different orders into orthogonal components. In addition, our approach does not require a concrete realization of the geometric object. As long as a discrete set X, which characterizes the symmetry of the geometric object, can be effectively extracted, our approach can be used for inferring the symmetry group G.

Related work We note that a great amount of research has already been done on Euclidean symmetry detection in the geometry processing community [7, 10]. However, those approaches often suffer from the curse of dimensionality. The problem of inferring the global symmetry from low-order similarities, is closely related to the graph automorphism problem, or more generally, the colored graph automorphism problem. However, there are no known polynomial time algorithms for finding the automorphism group of a general graph except for certain special cases such as the triply connected planar graph [2, 16]. The problem we consider also connects to the *orbit partitioning problem* whose goal is to determine whether two vertices or two pairs of vertices lie in the same orbits. However, those problems are generally very difficult, and there are no known polynomial time algorithms [1, 11, 13].

2 Marginal Probability of Cosets

In this section, we give a detailed description on how to organize all the elements in \mathbb{S}_n in a tree. We also show that the marginal probabilities of the nodes on the same level only take two possible values for the *indicator distribution* of \mathbb{G} .

We consider a tree decomposition of all permutations in \mathbb{S}_n as depicted in figure 1-(b). All permutations are classified into n sub-trees according to their mappings on the last element, i.e., $\sigma(n)$, where σ denotes a permutation. The n sub-trees are further classified according to their mappings on the last two elements, i.e., $\sigma(n-1)$ and $\sigma(n)$. In general, a node on the k-th level stands for all permutations that maps the tuple $(n - k + 1, \dots, n)$ to a particular k-tuple. Thus, the leaves in the tree represent all the permutations.

Let f be a distribution over S_n . We consider the marginal probability of a node on the *k*th level:

$$\sum_{\sigma \in \mathbb{S}_n} f(\sigma) I\bigg(\sigma(n-k+1,\cdots,n) = (t_{n-k+1},\cdots,t_n)\bigg), \tag{1}$$

which sums up all $f(\sigma)$ such that σ maps the k-tuple $(n-k+1, \dots, n)$ to the k-tuple (t_{n-k+1}, \dots, t_n) where t_i 's are all distinct and each $t_i \in \{1, \dots, n\}$. Here, I is an indicator function which is 1 if and only if σ maps i to t_i for all $n-k+1 \leq i \leq n$. Each node in the tree of permutations is associated with such a marginal probability. The following theorem characterizes specific properties of these marginal probabilities.

Theorem 1 Let
$$f(\sigma) = \begin{cases} \frac{1}{|\mathbb{G}|}, & \sigma \in \mathbb{G} \\ 0, & \sigma \notin \mathbb{G} \end{cases}$$
 (2)

be the indicator distribution for \mathbb{G} in \mathbb{S}_n , and let m_k be the marginal probability of all permutations that fix $(n-k+1,\dots,n)$, i.e., the quantity in (1) with $t_i =$ $i (n-k+1 \leq i \leq n)$. Then, we have $m_k \neq 0$; and for every node on the k-th level, its marginal probability (1) is either 0 or m_k .

The proof of Theorem 1 is based on coset representation theorems. This theorem immediately translates into a search strategy for estimating the group \mathbb{G} . Basically, we perform a top-down search in the tree of \mathbb{S}_n . A node which represent all permutations that map $(n-k+1,\cdots,n)$ to a k-tuple (t_{n-k+1},\cdots,t_n) will be kept only if its marginal probability is nonzero. The group \mathbb{G} can be fully decided if all those marginal information is available.

3 Inference with the Similarity Matrix

In real applications, the primary challenge for estimating the group \mathbb{G} is that the marginal probabilities needed for search are not directly observable. Instead, we typically can only verify low-order similarities. In this section, we introduce two related concepts: the *similarity matrix* and the *marginal probability matrix*.

Definition 2 A low-order similarity matrix S_k of order k (k is usually very small) for X (|X| = n) is an N-by-N matrix where $N = n(n-1)\cdots(n-k+1)$ and the (i, j)-entry is a similarity measure s_{ij} for two k-tuples $(t_1^{(i)}, \cdots, t_k^{(i)})$ and $(t_1^{(j)}, \cdots, t_k^{(j)})$ indexed by i, j.

We can construct various similarity measures s_{ij} for two k-tuples indexed by i and j, for example:

• k = 1: we can use a binary rule by letting $s_{ij} = 1$ if and only if points *i* and *j* have the same curvature; or use a continuous Gaussian kernel $s_{ij} = \exp(-|c_i - c_j|^2)$ where c_i, c_j are the curvatures of the point *i*, *j*.

• k = 2: we can use a binary rule by letting $s_{ij} = 1$ if and only if distances d_i and d_j are the same, where d_i (d_j) is the distance between two points in the pair i(j); or use a Gaussian kernel $s_{ij} = \exp(-|d_i - d_j|^2)$.

Definition 3 Given a distribution f on permutations $(\sum_{\sigma} f(\sigma) = 1)$, the k-th order marginal probability matrix H_k of f is an N-by-N matrix where $N = n(n-1)\cdots(n-k+1)$ and the (i,j)-entry equals



Figure 1: (a) Star. (b) Tree Decomposition of \mathbb{S}_n .

 $\begin{array}{l} \sum_{\sigma} f(\sigma) I(\sigma(i)=j), \mbox{ where } i \mbox{ and } j \mbox{ index two } k\mbox{-tuples:} \\ (t_1^{(i)}, \cdots, t_k^{(i)}) \mbox{ and } (t_1^{(j)}, \cdots, t_k^{(j)}).^2 \\ \mbox{ For a given geometric object } X, \mbox{ we can compute its} \end{array}$

For a given geometric object X, we can compute its low order similarity matrix. We hope such a matrix can approximate the marginal probability matrix of the indicator distribution if we normalize the similarity matrix so that each row sum equals one. As two real examples, we look at the low order similarity matrices for the star and human (see Figure 1-(a) and Figure 3-(d)).

For the star example, we compute the similarity matrices using the binary rule (see Figure 2-(a,c)). The first order similarity matrix is an all-one matrix which has no information about the symmetry, however, the second order similarity matrix S_2 can completely reveal the symmetry – if we normalize S_2 so that each row-sum is one, then the normalized similarity matrix exactly equals the marginal probability matrix H_2 of the distribution indicating the five-fold dihedral group $\mathbb{G} = \mathbb{D}_5$ as in (2) which characterizes the symmetry.

For the human example (see Figure 2-(b,d)), we compute the similarity matrices using the Gaussian kernel. The first order similarity matrix takes a block diagonal form, which partially reveals the symmetry of the human. However, there are still ambiguities that can not be resolved by first order information – whenever we map the left hand to the right hand, we have to map the left foot to the right foot. However, the second order similarity matrix S_2 constructed by computing $\exp(-|d_i - d_j|^2)$ can help us further clarify the symmetry group of the human – if we normalize S_2 , then it well approximates (there are tiny noises within the human model) the marginal probability matrix H_2 of the distribution indicated by one-fold dihedral group $\mathbb{G} = \mathbb{D}_1$.

In the above two examples, we observe that the normalized similarity matrix is a good approximation of the marginal probability matrix for the distribution indicating \mathbb{G} if we use a good similarity measure. Such a matrix can reveal \mathbb{G} better if we incorporate higher order information because in the extremal case the *n*-th order marginal probability matrix can exactly pinpoint the distribution over permutations. Theoretically, we may prove that normalized low order similarity matrix in the noiseless case (computed using the binary rule) equals the marginal probability matrix in the manifold setting, as long as the signatures we use to construct similarity measures are powerful enough [9].

In this sequel, we assume the normalized low-order similarity matrix estimated from geometric objects ap-



Figure 2: (a,b) First order similarity matrix for the star and human; (c,d) Second order similarity matrix for the star and human. The normalized similarity matrix are of the same block structures except that each row sum equals one. (e,f) Reconstructed distribution over permutations from the normalized second order similarity matrix for the star and human. Red dots denote good permutations.

proximates a marginal probability matrix of the indicator distribution of \mathbb{G} .³ By using the Fourier transforms over permutation group, we can extract a set of low frequency Fourier coefficients from the normalized loworder similarity matrix, which provides a band-limited approximation (ℓ_2 projection in the Fourier space) for the indicator distribution of \mathbb{G} over \mathbb{S}_n [5]. With such a set of Fourier coefficients, we estimate all the marginal probabilities and search elements in \mathbb{G} in the tree of all permutations.

3.1 Fourier Approach

In this section, we consider the problem of estimating all the marginal probabilities needed for search based on the normalized low-order similarity matrix S. We first translate the matrix S into a set of Fourier coefficients \hat{f}_{λ} 's using Specht modules [6]. Those Fourier components (indexed by λ) characterize the low-frequency components of the distribution f over permutations. After that, we compute a pointwise product of f with the indicator $I(\sigma(n) = (t_n))$ in the Fourier domain, so that the distribution over all permutations $\{\sigma\}$ such that σ maps n to t_n can be extracted. The result of such a pointwise distribution can be summarized by a distribution over \mathbb{S}_{n-1} if one relabels $1, 2, \dots, n$ so that t_n becomes n. We use an algorithm called Kronecker Con*ditioning* to compute the pointwise product completely in the Fourier domain [5]. A theorem by [8] gives us a bound on which representations can appear in the result of such a pointwise product. We finally apply an FFT based approach, which will be described later, to compactly summarize such a distribution over \mathbb{S}_{n-1} .

The above procedure decomposes the distribution implied by S to n distributions over S_{n-1} . Such a proce-

²The m_k defined in Theorem 1 is one element of H_k where $(t_1^{(i)}, \dots, t_k^{(i)})$ and $(t_1^{(j)}, \dots, t_k^{(j)})$ both equals $(n-k+1, \dots, n)$.

 $^{^{3}}$ We make sure that the normalized similarity matrix is a valid marginal probability matrix by imposing certain inherent constraints such as doubly stochasticity [5].

dure can be used iteratively on each \mathbb{S}_{n-1} , until we reach the bottom of the tree decomposition of \mathbb{S}_n . At each node, a set of Fourier coefficients are maintained to characterize the distribution over permutations dominated by that node. The key benefit of using Fourier coefficients to summarize the information is due to the simplicity of evaluating marginals in the Fourier domain [5].

In this process, we see that the relabeling is used extensively, so that we can view the permutations as if they are always permuting $(1, \dots, n-1)$, rather than mapping from $(1, \dots, n-1)$ to $(1, \dots, \hat{t}_n, \dots, n)$. Whenever such a relabeling operation is used, the Fourier coefficients of f on \mathbb{S}_n also change accordingly. It turns out that there is a class of operations, called *shift* operations [3] which can compute the Fourier transform with respect to the reordered sets.

3.2 FFT-Based Method

In this section, we detail how to extract the Fourier coefficients of f restricted on \mathbb{S}_{n-1} from \hat{f}_{λ} , which is an essential step in estimating the marginals.

In the tree decomposition of \mathbb{S}_n , we see that $\mathbb{S}_n = \bigcup_{i=1}^n [\![i,n]\!] \mathbb{S}_{n-1}$, where $[\![i,n]\!]$ denotes the cyclic permutation $(i, i+1, \cdots, n)$ (*i* is mapped to i+1, i+1 is mapped to i+2, etc, *n* is mapped to *i*), and $[\![i,n]\!] \mathbb{S}_{n-1}$ is the so-called left \mathbb{S}_{n-1} -coset

$$\llbracket i, n \rrbracket \mathbb{S}_{n-1} = \{ \sigma \in \mathbb{S}_n | \sigma(n) = i \}$$
(3)

The fast Fourier transform (FFT) for S_n works by relating the Fourier transform over S_n to Fourier transforms over the above *n* cosets. This idea can be applied recursively, computing the Fourier transform on each S_{n-1} -coset from n-1 Fourier transforms on S_{n-2} cosets, etc., all the way down to S_1 -cosets, which are individual permutations. We will present a method to estimate the high order marginals using this approach.

More precisely, we can define the restriction of f to the $[\![i,n]\!]\mathbb{S}_{n-1}$ -coset as $f_i(\tau) = f([\![i,n]\!]\tau)$ (which is now a function on \mathbb{S}_{n-1}), and observing that the Fourier transform of f can be broken up as

$$\hat{f}_{\lambda} = \sum_{\sigma \in \mathbb{S}_n} f(\sigma) \rho_{\lambda}(\sigma) = \sum_{i=1}^n \sum_{\tau \in \mathbb{S}_{n-1}} f(\llbracket i, n \rrbracket \tau) \rho_{\lambda}(\llbracket i, n \rrbracket \tau)$$
(4)

$$= \sum_{i=1}^{n} \rho_{\lambda}(\llbracket i, n \rrbracket) \sum_{\tau \in \mathbb{S}_{n-1}} f_i(\tau) \rho_{\lambda}(\tau)$$
(5)

The inner summation on the right of this equation looks almost like the Fourier transform of f_i over the smaller group \mathbb{S}_{n-1} , except that ρ_{λ} is an irreducible representation of \mathbb{S}_n instead of \mathbb{S}_{n-1} . In fact, the $\rho_{\lambda}(\tau)$ matrices do form a representation of \mathbb{S}_{n-1} , but in general this representation is not irreducible. Maschke's theorem [14] tells us that we can express it in terms of the ρ_{μ} irreducible representations of \mathbb{S}_{n-1} in the form

$$\rho_{\lambda}(\tau) = \bigoplus_{\mu \in \lambda \downarrow_{n-1}} \rho_{\mu}(\tau) \tag{6}$$

if a particular system of irreducible representations for \mathbb{S}_n , called Young's Orthogonal Representation (YOR) [14] is used. Here $\lambda \downarrow_{n-1}$ denotes the set of all partitions of n-1 dominated by λ , i.e., those partitions that we can get from λ by removing a single box from λ 's diagram. Plugging (6) into (5) gives the relationship between \hat{f} and $\hat{f}_1, \hat{f}_2, \dots, \hat{f}_n$:

$$\hat{f}_{\lambda} = \sum_{i=1}^{n} \rho_{\lambda}(\llbracket i, n \rrbracket) \bigoplus_{\mu \in \lambda \downarrow_{n-1}} (\hat{f}_{i})_{\mu}$$

$$\tag{7}$$

Such a formula can also be inverted to express \hat{f}_1 , \hat{f}_2 , \cdots , \hat{f}_n in terms of \hat{f} :

$$(\hat{f}_i)_{\mu} = \frac{n-1}{nd_{\mu}} \sum_{\lambda \in \mu \uparrow n} d_{\lambda} \rho_{\lambda}(\llbracket i, n \rrbracket)^{-1} (\hat{f}_{\lambda})_{\mu}$$
(8)

where $\mu \uparrow^n$ is the set of all partitions of *n* that dominate μ , i.e., which can be derived from μ by adding a single box, and $(\hat{f}_{\lambda})_{\mu}$ is the block of \hat{f}_{λ} for μ .

The ideas in FFTs can be used to identify a restricted components. For the function given by $f(\sigma)I(\sigma(n) = n)$, we know that f only takes nontrivial values on \mathbb{S}_{n-1} . We have the following result to exactly calculate the Fourier coefficients for the function f restricted on \mathbb{S}_{n-1} .

Theorem 4 Given a distribution f on \mathbb{S}_n that only takes nontrivial values on \mathbb{S}_{n-1} , then function restricted on \mathbb{S}_{n-1} has Fourier coefficients

$$(\widehat{f|_{\mathbb{S}_{n-1}}})_{\mu} \propto \frac{1}{d_{\mu}} \sum_{\lambda \in \mu \uparrow n} \sum_{j=1}^{z_{\lambda,\mu}} d_{\lambda}(\widehat{f}_{\lambda})_{\mu}$$
(9)

The operation involved in computing the Fourier coefficients for $f|_{\mathbb{S}_{n-1}}$ amounts to finding certain blocks within the \hat{f}_{λ} matrices and adding them together weighted by the appropriate d_{λ} and d_{μ}^{-1} constants. We can upper bound the computational complexity by the total size $\sum_{\lambda} d_{\lambda}^2$ of the \hat{f}_{λ} matrices. Since we only store the first few low-frequency Fourier components, the computing complexity is thus strongly polynomial.

In summary, the FFT-based method provides a scalable algorithm for computing Fourier coefficients for $f|_{\mathbb{S}_{n-1}}$. The representations that can appear in the computation result is also guaranteed, see proposition 5.

Proposition 5 Given a set of Fourier coefficients for a distribution f over \mathbb{S}_n whose order dominates $\lambda = (n - p, 1, \dots, 1)$, the FFT-based method computes a set of Fourier coefficients whose order dominates $\lambda = (n - p - 1, 1, \dots, 1)$ for $f|_{\mathbb{S}_{n-1}}$ with complexity $\mathcal{O}(\sum_{\lambda} d_{\lambda}^2)$.

4 Marginal Probability Search

Based on previous sections, by exploiting the algebraic structure of \mathbb{G} , we now formally propose a new algorithm – the marginal-based search. This algorithm searches within the tree of the permutation group \mathbb{S}_n , from the root towards deeper levels, until the group \mathbb{G} is fully identified. Starting from the root, we iteratively search and build deeper level nodes based on the marginal information. If the estimated marginal defined in (1) is no less than ϵ times m_k (see Theorem 1 from which we know the left-most node must have nonzero



Figure 3: (a) Icosahedron. (b) Octopus. (c) Hand. (d) Human.

Algorithm 1 Marginal-Based Search

Input: A normalized similarity matrix S, ϵ .
Output: A list of automorphisms characterizing G.
Procedure:
Initialize the tree T of \mathbb{S}_n
Estimate a set of Fourier coefficients $\{\hat{f}_{\lambda}\}$.
$k \leftarrow 0$
for k from 1 to n do
Build child nodes for every node on the $(k-1)$ -th level.
for Each node on the k -th level do
$\{\hat{f}_{\mu}\} \leftarrow$ Fourier coefficients of a distribution over \mathbb{S}_{n-k} .
$m_k \leftarrow 0$ -th order Fourier coefficient of the left-most node.
if The marginal of the node is less than ϵm_k then
Prune the node.
end if
end for
Prune any node which does not have any k-th level children.
end for

=	 			
l	Name	#Vertices	G	Running time
	Tetrahedron	4	24	0.07s
ſ	Hexahedron	8	48	1.07s
	Octahedron	6	48	0.61s
	Dodecahedron	20	120	131.2s
l	Icosahedron	12	120	49.5s

Table 1: Detecting Symmetries of Regular Polyhedra

marginal), we keep this node; otherwise, we drop it off.⁴ Such an iterative search algorithm is essentially doing a sparse pursuit of \mathbb{G} within \mathbb{S}_n , which takes account of the group structures of \mathbb{G} , see algorithm 1 for the pseudo code.

Such a marginal-based search algorithm uses a set of Fourier coefficients to approximate a distribution over \mathbb{S}_{n-k} for a node on the k-th level. However, since low-frequency Fourier coefficients characterize a smooth distribution over all the permutations. We typically observe that ϵ will be choosen to be very large, e.g., around 0.8. However, we still have theoretical guarantees about our approach.

Theorem 6 Suppose S is the first order marginal probability matrix reconstructed from \hat{f}_{μ} , when all S's are block diagonal dominant matrices⁵ which indicate orbits partitioning, then Algorithm 1 can find all symmetries in \mathbb{G} .

In the second order matrix case, we restrict our theorem to the special case that all points lie in the same orbit. If all points does not lie in the same orbit, then the inverse Fourier transform formula will put weights



Figure 4: Searching for \mathbb{G} using the marginal-based search algorithm for the perturbed icosahedron. We use a simplified notation for each node, for example, the node of $(11) \rightarrow (11)$ denote all permutations that maps $(11, 12) \rightarrow (11, 12)$. The blue nodes are those should be kept and the red nodes are those should be dropped off in the groundtruth.

	Dodecahedron	Icosahedron	Octopus	Human
Greedy	76.7%	90.8%	81.3%	100%
Eigen	75.8%	85.8%	75.0%	50%
Morgan	72.5%	87.5%	75.0%	50%
Fourier	79.1%	91.7%	87.5%	100%
	1 0 1	C DIC		1

Table 2: Accuracy of Different Approaches.

	Dodecahedron	Icosahedron	Octopus	Human
Greedy	248.71	85.09	5.89	0.59
Eigen	220.35	82.10	5.03	0.17
Morgan	218.92	79.50	5.18	0.17
Fourier	140.04	52.31	2.03	0.41
1	<u>ар : п</u>	D' (D')(4	1

Table 3: Running Time of Different Approaches.

on different entries in S_{ij} which yield a distribution over permutations. However, such a distribution is still an ℓ_2 projection of the noised indicator distribution to the Fourier space.

5 Experiments

We test our algorithm on several examples, including regular polyhedra and 3D geometric objects. All the experiments are performed in Matlab on a regular desktop with 2.4GHz CPU and 3G RAM.

The first example is on detecting the symmetries for all 3D regular polyhedra. We first build a second order similarity matrix S using the continuous Gaussian kernel $s_{ij} = \exp(-|d_i - d_j|^2)$ where where d_i (d_j) is the distance between two points in the pair indexed by i (j). We normalize S and use it as a marginal probability matrix to estimate the symmetry group \mathbb{G} by implementing Algorithm (1). As shown in table 1, our approach can detect the symmetry group \mathbb{G} for all 3D regular polyhedra with reasonable running time. We note that brute force search for the symmetry group \mathbb{G} for dodecahedron and icosahedron would be very difficult, since the sizes of the permutation groups are $20! \approx 2.43 \times 10^{18}$ and $12! \approx 4.79 \times 10^8$.

One benefit of the proposed approach of inferring the symmetry group \mathbb{G} is that it can naturally deal with

⁴When we are done with building the k-th level nodes in the tree, we also prune the nodes in the current tree which do not have any k-th level children.

 $^{^5\}mathrm{Entries}$ off blocks are strictly less than entries within blocks that lie in the same row and column

noise. As an example, we randomly perturb the vertices of the icosahedron with certain magnitudes. After that, we repeat the previous experiment where we build the similarity matrix, normalize it to get a marginal probability matrix, and then estimate the symmetry group \mathbb{G} . It turns out that our approach can still find the symmetry group \mathbb{G} for the perturbation with a relative magnitude up to 0.05 (the edge length is 1).

To demonstrate how the marginal-based search algorithm prunes the nodes in the tree, we show part of the tree implemented during our experiments in Figure 4-(a). As we can see from the figure, on the first, second and fifth level, the nodes that intersect with \mathbb{G} have larger marginals than those that do not. Such a gap tends to be smaller at certain levels, such as level 3 and 4, thus it is very possible that we may include some nodes which do not intersect with \mathbb{G} during the implementation of our algorithm. However, it turns out that when we look several levels down, those nodes will be dropped. The labeling of vertices of the icosahedron illustrated in this tree is shown in Figure 3-(a).

As another example, we detect the symmetries for the octopus, see Figure 3-(b). Unlike the regular polyhedra examples, the symmetry of the octopus can only be defined as isometries which preserve the geodesic distance, rather than the Euclidean distance. We use the fuzzy geodesics proposed in [15] which can be interpreted as a robust distance measure to get an effective similarity matrix between pairs. The later routines for inferring the group \mathbb{G} are the same as in previous experiments. Though in this example the octopus is heavily deformed, we can still recover the dihedral group as its symmetry group.

Using the same technique of fuzzy geodesic metrics, we can get similarity matrix for many other 3D geometric models, such as the hand and human, as shown in Figure 3-(c,d). We can fully determine the 2-fold symmetries of the human model using our approach. However, for the heavily perturbed hand model, many permutations will be identified to be good ones, among which the permutations that have the highest values are still meaningful. For example, the top 2 permutations being identified are the identity and $(1, 2, 3, 4, 5, 6) \rightarrow$ (6, 3, 2, 4, 5, 1).

We finally compare our algorithm with another greedy heuristics [12] whose pruning criteria is based on the current maximum distortions. Several other algorithms such as principle eigenvector analysis (spectral analysis of the similarity matrices), the Morgan algorithm (an iterative procedure to estimate the orbit partitioning), and etc [1, 11] can be used as a pre-processing step which may reduce the size of the searching space. The comparison of accuracy (how many percentages of correct symmetries identified) and running time of different approaches are shown in Table 2 and 3. Throughout these experiments, we distort the geometric models so that it becomes difficult to recognize all the symmetries. Thus, we typically observe that eigenvector analysis and Morgan algorithm often make errors in identifying the orbit of the vertices. Whenever such an error occurs, it decreases the accuracy dramatically. The greedy heuristic algorithm typically has longer running time than our proposed Fourier approach.

6 Acknowledgement

The authors would like to thank the anonymous reviewers for valuable comments and suggestions. Leonidas Guibas and Xiaoye Jiang wish to acknowledge the support of NSF grants FODAVA 0808515, as well as ARO grant W911NF-10-1-0037.

- [1] P. J. Cameron. *Permutation Groups*. London Mathematical Society, 1999.
- [2] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar graph isomorphism is in logspace. In *IEEE Conference on Computational Complexity*, 2009.
- [3] P. Diaconis. Group Representations in Probability and Statistics. Institute of Mathematical Statistics, 1988.
- [4] T. Funkhouser and P. Shilane. Partial matching of 3D shapes with priority-driven search. In *Proceedings of* SGP, 2006.
- [5] J. Huang, C. Guestrin, and L. J. Guibas. Fourier theoretic probabilistic inference over permutations. *Journal* of Machine Learning Research, 2009.
- [6] G. D. James. The Representation Theory of the Symmetric Groups. Springer-Verlag, 1978.
- [7] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. In ACM Transactions on Graphics, 2006.
- [8] F. Murnaghan. The analysis of the kronecker product of irreducible representations of the symmetric group. *American Journal of Mathematics*, 1938.
- [9] M. Ovsjanikov, Q. Merigot, F. Memoli, and L. Guibas. One point isometric matching with the heat kernel. In *Proceedings of SGP*, 2010.
- [10] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar-reflective symmetry transform for 3D shapes. In *Proceedings of SIGGRAPH*, 2006.
- [11] I. Ponomarenko. Graph isomorphism problem and schurian algebras. *preprint*, 1994.
- [12] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Symmetries of non-rigid shapes. In *Proceedings of ICCV*, 2007.
- [13] R. Read and D. Corneil. The graph isomorphism disease. Journal of Graph Theory, 1977.
- [14] B. Sagan. The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions. Springer-Verlage, 2001.
- [15] J. Sun, X. Chen, and T. Funkhouser. Fuzzy geodesics and consistent sparse correspondences for deformable shapes. In *Proceedings of SGP*, 2010.
- [16] L. Weinberg. On the maximum order of the automorphism group of a planar triply connected graph. SIAM Journal on Applied Mathematics, 1966.

List coloring and Euclidean Ramsey Theory (Abstract)

Noga Alon *

Alexandr Kostochka[†]

Abstract

It is well known that one can color the plane by 7 colors with no monochromatic configuration consisting of the two endpoints of a unit segment, and it is not known if a smaller number of colors suffices. Many similar problems are the subject of Euclidean Ramsey Theory, introduced by Erdős et. al. in the 70s.

In sharp contrast we show that for any finite set of points K in the plane, and for any finite integer s, one can assign a list of s distinct colors to each point of the plane, so that any coloring of the plane that colors each point by a color from its list contains a monochromatic isometric copy of K. The proof follows from a general new theorem about coloring uniform simple hypergraphs with large minimum degrees from prescribed lists.

1 Euclidean Ramsey Theory

A well known problem of Hadwiger and Nelson is that of determining the minimum number of colors required to color the points of the Euclidean plane so that no two points at distance 1 have the same color. Hadwiger showed already in 1945 that 7 colors suffice, and Nelson as well as L. Moser and W. Moser noted that 3 colors do not suffice. These bounds have not been improved, despite a considerable amount of effort by various researchers.

A more general problem has been considered by Erdős, Graham, Montgomery, Rothschild, Spencer and Straus [4, 5, 6] under the name Euclidean Ramsey Theory. The main question is the investigation of finite point sets K in the Euclidean space for which any coloring of an Euclidean space of a sufficiently high dimension $d \ge d_0(K, r)$ by r colors must contain a monochromatic copy of K. The conjecture is that this holds for a set K if and only if it can be embedded in a sphere. Another conjecture considered by these authors asserts that for any set K of 3 points in the plane, there is a coloring of the plane by 3 colors with no monochromatic copy of K.

Intriguing variants of these questions arise when one places some restrictions on the set of colors available in each point. This is related to the notion of list coloring introduced by Vizing [8] and by Erdős, Rubin and Taylor [7].

2 List coloring

The list chromatic number (or choice number) $\chi_{\ell}(G)$ of a graph G = (V, E) is the minimum integer s such that for every assignment of a list L_v of s colors to each vertex v of G, there is a proper vertex coloring of G in which the color of each vertex is in its list. This notion was introduced independently by Vizing [8] and by Erdős, Rubin and Taylor [7]. In both papers the authors realized that this is a variant of usual coloring that exhibits several new properties, and that in general $\chi_{\ell}(G)$, which is always at least as large as the chromatic number of G, may be arbitrarily large even for graphs G of chromatic number 2.

It is natural to extend the notion of list coloring to hypergraphs. The list chromatic number $\chi_{\ell}(H)$ of a hypergraph H is the minimum integer s such that for every assignment of a list of s colors to each vertex of H, there is a vertex coloring of H assigning to each vertex a color from its list, with no monochromatic edges.

An interesting property of list coloring of graphs, which is not shared by ordinary vertex coloring, is the result that the list chromatic number of any (simple) graph with a large average degree is large. Indeed, it is shown in [1] that the list chromatic number of any graph with average degree d is at least $(\frac{1}{2} - o(1)) \log_2 d$, where the o(1)-term tends to zero as d tends to infinity. Our main combinatorial result is an extension of this fact to simple uniform hypergraphs.

^{*}Schools of Mathematics and Computer Science, Tel Aviv University, Tel Aviv, Israel nogaa@tau.ec.il

[†]Department of Mathematics, University of Illinois, Urbana, IL, USA kostochk@math.uiuc.edu

3 The new results

A hypergraph is called *simple* if every two of its distinct edges share at most one vertex. It is an r-graph if each of its edges contains exactly r vertices. We prove that the result of [1] can be extended to simple r-graphs. This is stated in the following theorem.

Theorem 1 For every fixed $r \ge 2$ and $s \ge 2$, there is d = d(r, s), such that the list chromatic number of any simple r-graph with n vertices and nd edges is greater than s.

It is worth noting that the theorem provides a linear time algorithm for computing, for a given input simple r-graph, a number s such that its list chromatic number is at least s and at most f(s) for some explicit function f. There is no such known result for ordinary coloring, and it is known that there cannot be one under some plausible hardness assumptions in Complexity Theory, as shown in [3].

The above result implies the following.

Theorem 2 For any finite set X in the Euclidean plane and for any positive integer s, there is an assignment of a list of size s to every point of the plane, such that whenever we color the points of the plane from their lists, there is a monochromatic isometric copy of X.

The proofs of both theorems can be found in [2].

- [1] N. Alon, Degrees and choice numbers, Random Structures & Algorithms 16 (2000), 364–368.
- [2] N. Alon and A. V. Kostochka, Hypergraph list coloring and Euclidean Ramsey Theory, Random Structures and Algorithms, to appear.
- [3] I. Dinur, E. Mossel and O. Regev, Conditional hardness for approximate coloring, SIAM J. Comput. 39 (2009), 843–873.
- [4] P. Erdős, R. L. Graham, P. Montgomery, B. L. Rothschild, J. Spencer, and E. G. Straus, Euclidean Ramsey theorems. I. J. Combinatorial Theory Ser. A 14 (1973), 341–363.
- [5] P. Erdős, R. L. Graham, P. Montgomery, B. L. Rothschild, J. Spencer, and E. G. Straus, Euclidean Ramsey theorems. II. Infinite and finite sets (Colloq., Keszthely, 1973) Vol. I, pp. 529–557. Colloq. Math. Soc. Janos Bolyai, Vol. 10, North-Holland, Amsterdam, 1975.
- [6] P. Erdős, R. L. Graham, P. Montgomery, B. L. Rothschild, J. Spencer, and E. G. Straus, Euclidean Ramsey theorems, III. Infinite and finite sets (Colloq., Keszthely, 1973) Vol. I, pp. 559–583. Colloq. Math. Soc. Janos Bolyai, Vol. 10, North-Holland, Amsterdam, 1975.
- [7] P. Erdős, A. L. Rubin and H. Taylor, *Choosability in graphs*, Proc. West Coast Conf. on Combinatorics, Graph Theory and Computing, Congressus Numerantium XXVI, 1979, 125-157.
- [8] V. G. Vizing, Coloring the vertices of a graph in prescribed colors (in Russian), Diskret. Analiz. No. 29, Metody Diskret. Anal. v. Teorii Kodov i Shem 101 (1976), 3-10.

Rigidity-Theoretic Constructions of Integral Fary Embeddings

Timothy Sun*

Abstract

Fáry [3] proved that all planar graphs can be drawn in the plane using only straight line segments. Harborth *et al.* [7] ask whether or not there exists such a drawing where all edges have integer lengths, and Geelen *et al.* [4] proved that cubic planar graphs satisfied this conjecture. We re-prove their result using rigidity theory, exhibit other natural families of planar graphs that satisfy this conjecture as immediate corollaries, and also prove a weaker result for all planar graphs in \mathbb{R}^3 .

1 Introduction

All graphs in this paper are simple and finite. Let G = (V, E) be a planar graph. A *Fary embedding* $\phi_G : V \rightarrow \mathbb{R}^2$ of G is an embedding such that the drawing induced by ϕ_G with straight-line edges has no crossing edges. Fáry [3] proved a classic theorem on these embeddings.

Theorem 1 (Fáry [3]) All planar graphs have a Fary embedding.

The main idea for the proof was by induction on the number of vertices. A vertex v of degree at most 5 in the interior is deleted from a maximal planar graph G, and v is carefully replaced in a Fary embedding of a triangulation of G - v so that it "sees" all its neighbors. An *integral Fary embedding* is a Fary embedding in which for all adjacent vertices a and b, $||\phi_G(a) - \phi_G(b)||$ is an integer. Harborth *et al.* [7] found integral Fary embeddings for the Platonic graphs, which led them to conjecture the following.

Conjecture 1 All planar graphs have an integral Fary embedding.

Previous attacks on this conjecture took the same direction as [3], inductively adding new vertices by using solutions to Diophantine equations. Kemnitz and Harborth [8] outline an idea for a possible proof and a construction for some planar graphs, but their method does not always work. Geelen *et al.* [4] give a partial solution in which they demonstrated that all cubic planar graphs satisfy Conjecture 1. Our method determines when it is possible to perturb an edge length without affecting any other edge lengths and preserving planarity, and we prove Conjecture 1 for planar graphs in which all edges can be perturbed. While that family does not contain all planar graphs, it does contain some well-known families of planar graphs, including all but one of the cubic planar graphs. Unlike the results in [4] and [8], the exact combinatorial characterization for such graphs are known.

2 Rigidity and Edge Perturbations

A (d-dimensional) framework is a pair (G, p) where $p: V(G) \to \mathbb{R}^d$, known as a *configuration*, is a mapping which takes the vertices of G to points in Euclidean dspace. We assume that the image of p does not lie on a hyperplane and that p is injective. A generic configuration is one where all vd coordinates are independent over the rationals, and a *generic framework* is a framework with a generic configuration. We say that a framework is *flexible* (in \mathbb{R}^d) if there exists a continuous motion of the vertices that preserves edge lengths and is not a Euclidean motion, and that it is *rigid* otherwise. We say that a graph is (generically) flexible/rigid if all generic frameworks of that graph are flexible/rigid. While the rigidity of a particular framework is dependent on the choice of configuration, generic rigidity is a property of only the underlying graph.

Let G be a graph. Consider the function f_G that takes a configuration to a vector of all the edge lengths squared. In other words, $f_G : \mathbb{R}^{vd} \to \mathbb{R}^e$ is a function which takes

$$p = (p_1, p_2, \dots, p_v) \mapsto (\dots, ||p_i - p_j||^2, \dots).$$

The Jacobian $df_G(p)$, called the *rigidity matrix* of (G, p), is an $e \times vd$ matrix where each row corresponds to an edge and encodes the vector between the two vertices incident with the edge. For example, the rigidity matrix of the graph K_3 with the configuration $p_1 = (1, 1), p_2 = (2, -2), p_3 = (0, 3)$ can be written as

$$\begin{array}{ccccccc} p_{1,1} & p_{1,2} & p_{2,1} & p_{2,2} & p_{3,1} & p_{3,2} \\ v_1v_2 & & & \\ 2 * & v_1v_3 & & \\ v_2v_3 & v_2v_3 & & \\ \end{array} \begin{pmatrix} -1 & 3 & 1 & -3 & 0 & 0 \\ 1 & -2 & 0 & 0 & -1 & 2 \\ 0 & 0 & 2 & -5 & -2 & 5 \end{pmatrix}.$$

By factoring out the 2, the entries of the rigidity matrix can be calculated by simply taking the difference be-

^{*}Department of Computer Science, Columbia University, ts2578@columbia.edu

tween coordinates. Since all possible edge-length functions can be obtained by some permutation of the edges and swapping the "head" and "tail" of an edge, we refer to df_G as the rigidity matrix. Most importantly, we are only interested in the rank of the rigidity matrix, which is not affected by such choices. The derivative of any flex or Euclidean motion on a framework lies in the kernel of the framework's rigidity matrix, so the rank gives an informal notion of how rigid the framework is. In particular, adding an edge which is linearly independent from the other edges reduces the dimension of the space of flexes by one. The rank of the rigidity matrix is dependent on the choice of configuration, so we restrict our attention to a specific subset of all configurations. A configuration p is a regular point of f_G if

$$\operatorname{rank} df_G(p) = \max_{q \in \mathbb{R}^{vd}} \operatorname{rank} df_G(q).$$

For v > d, let r(v, d) be defined as the quantity $vd - \binom{d+1}{2}$. $vd - \binom{d+1}{2}$ can be informally thought of as the number of degrees of freedom we have in selecting a framework. There are vd coordinates to choose from, but the *d*-dimensional space of translations and the $\binom{d}{2}$ -dimensional space of rotations limit the space of non-congruent frameworks. Asimow and Roth [1] formalized this intuitive notion and proved that at regular points, a framework on more than *d* vertices is rigid at a regular point if and only if the rank of its rigidity matrix is r(v, d). In the case where *v* is at most *d*, the only rigid graphs are the complete graphs.

The theorem by Asimow and Roth [1] roughly states that we need as many edge constraints as degrees of freedom for a framework to be rigid. For example, the graph $K_{3,3}$ is generically rigid in \mathbb{R}^2 since it has 9 = r(6, 2) linearly independent edges. In fact, the only flexible frameworks of $K_{3,3}$ are those whose vertices lie on a conic section, like in Figure 1. However, even when there are exactly r(v, d) edges, the framework may not be rigid. For example, any degree 1 vertex can pivot around its neighbor in the plane. We will present the classic combinatorial characterization of graphs with all independent edges in \mathbb{R}^2 in the next section that eliminates such problems.



Figure 1: When the configuration is not generic, sometimes the framework is infinitesimally flexible even though the graph is generically rigid.

A redundant edge is one whose removal does not de-

crease the rank of the rigidity matrix. With regards to flexes, removing a redundant edge does not increase the space of flexes. The main technique of this paper is described in the following theorem.

Theorem 2 Let (G, p) be a framework in \mathbb{R}^2 such that p is a Fary embedding and a regular point, and let ab be a non-redundant edge. Then there exists a framework (G, p') such that p' is a Fary embedding and a regular point, ||p'(a) - p'(b)|| is rational, and all other edge lengths remain fixed.

Proof. There exists an open neighborhood of configurations N_{ϵ} around p of Fary embeddings. To see this, we can examine the set of configurations where two fixed edges do not intersect. Since this set is open, the intersection of all such constraints is also open. Furthermore, there exists an open neighborhood of regular points N_{η} around p. This follows from the fact that there is a maximal rank square submatrix with non-zero determinant and that the determinant is a continuous function on the coordinates.



Figure 2: Perturbing an edge. If we remove a nonredundant edge ab of a rigid framework (G, p_1) , there will be a non-trivial one-dimensional flex C. By restricting C to $N_{\epsilon} \cap N_{\eta}$, we can find a configuration p_2 such that $||p_2(a) - p_2(b)||$ is rational. Then, the edge abin (G, p_2) has rational length.

Consider the graph formed by removing ab. Since ab is non-redundant, its removal creates a one-dimensional space of flexes. Moving along the flex, the distance between a and b changes, otherwise this flex would be a Euclidean motion. Since the rationals are dense in the reals, we can find a configuration $p' \in N_{\epsilon} \cap N_{\eta}$ such that ||p'(a) - p'(b)|| is rational. Then p' is a regular point $(p' \in N_{\eta})$ and a Fary embedding $(p' \in N_{\epsilon})$, ||p'(a) - p'(b)|| is rational, and all other edge lengths remained constant.

3 Harborth's Conjecture for (2,3)-Sparse Graphs

A graph G is (m, n)-sparse if for any subgraph G' with v' vertices and e' edges, $e' \leq \max(0, mv'-n)$. Our result in the plane relies on the following characterization.
Theorem 3 (Graver et al. [6, Lemma 4.2.1])

Every edge in (G, p) is non-redundant at a regular point p if and only if G is (2,3)-sparse.

This result perhaps takes on its most familiar form in Laman [9], who demonstrates that the generically rigid graphs in the plane are exactly the (2, 3)-sparse graphs with 2v - 3 edges, the so-called Laman graphs.



Figure 3: $K_3 \times K_2$, a planar Laman graph.

In the original paper, Laman actually showed only the existence of rigid realizations of Laman graphs, but the density of regular points guarantees that all generic frameworks of Laman graphs are rigid, as well. While the (2,3)-sparseness property does not directly play a part in the proof of the following result, it is useful in characterizing other families of planar graphs that have integral Fary embeddings as corollaries.

Theorem 4 All planar (2,3)-sparse graphs have integral Fary embeddings.

Proof. Let G be a (2,3)-sparse graph. We can find a Fary embedding p that is also a regular point by taking a Fary embedding ϕ_G and perturbing it slightly. The regular points are dense in \mathbb{R}^{vd} , so this is always possible. By repeatedly applying Theorems 2 and 3, we can perturb all edges to rational lengths by a sequence of configurations $p = p_0 \rightarrow p_1 \rightarrow \ldots \rightarrow p_e$, where each successive term in the sequence is obtained by perturbing another edge in the preceding configuration. Then, p_e is a Fary embedding of G with all rational edge lengths, and scaling appropriately yields an integral Fary embedding.

An intuitive way to interpret the above result is that if there are few edges and they are evenly spread out among the vertices, it is possible to perturb the lengths of the edges almost however we want. That is, we can choose any rational lengths within some neighborhood of a Fary embedding and obtain an integral Fary embedding by scaling. The construction by Geelen *et al.* [4] can be shown to work on (2,3)-sparse graphs, but their result does not allow for arbitrary choices of rational lengths. On the other hand, Biedl [2] gives an efficient algorithm in the case of 3-connected cubic graphs demonstrating that we can actually choose integer lengths linear in the number of vertices.

Unfortunately, (2,3)-sparseness is far from covering all the planar graphs. When the graph has more than 2v-3 edges, we can no longer use this approach since there are redundant edges. Fortunately, this approach is just enough to prove some already-known results. Let G be a *sub-cubic* graph if it has maximal degree 3. We obtain the following results from Theorem 4.

Corollary 5 (Geelen et al. [4]) All sub-cubic planar graphs have integral Fary embeddings.

Proof. Sub-cubic graphs have at most $\frac{3}{2}v$ edges, so the only sub-cubic graph with more than 2v-3 edges is K_4 . No connected sub-cubic graph can have K_4 as a proper subgraph because otherwise some vertex would have degree at least 4. Hence, all connected sub-cubic graphs with the exception of K_4 are (2,3)-sparse, so they have an integral Fary embedding by the previous theorem. There are several ways of finding an integral Fary embedding for K_4 , the smallest of which can be found using Pythagorean triples as demonstrated in [7].

Corollary 6 Triangle-free planar graphs have integral Fary embeddings.

Proof. Triangle-free graphs with $v \ge 3$ have at most 2v - 4 edges, and since a subgraph of a triangle-free graph is also triangle-free, they are (2, 3)-sparse. \Box

Corollary 7 Bipartite planar graphs have integral Fary embeddings.

G is a *series-parallel* graph if it is a subgraph of a graph that is constructed from K_2 by adding vertices and attaching them to two adjacent vertices. Wagner [10] proved that a graph is series-parallel if and only if it does not contain K_4 as a minor. Since both $K_{3,3}$ and K_5 have K_4 as a minor, series-parallel graphs are planar. Alternatively, the constructive characterization immediately yields a method of finding integral Fary embeddings.

Corollary 8 Series-parallel graphs have integral Fary embeddings.

Proof. Let G be a "maximal" series-parallel graph. As stated above, G can be constructed from adding new vertices and connecting them to adjacent vertices, so G has 2v - 3 edges. Any subgraph of G is also series-parallel, so G is (2, 3)-sparse.

Corollary 9 Outerplanar graphs have integral Fary embeddings.

4 Integral Convex Embeddings in \mathbb{R}^3

In this section, we prove a result weaker than Conjecture 1. A *convex embedding* is an embedding of a planar graph in \mathbb{R}^3 such that the set of edges can be extended

to form the skeleton of a convex polyhedron on the same set of vertices. One notable property of such an embedding is that it is also *linkless* (and furthermore flat). That is, the set of cycles are pairwise unlinked in a convex embedding. We prove that all planar graphs have integral convex embeddings by using a known sufficient condition for independence in \mathbb{R}^3 .

Theorem 10 (Gluck [5]) Let (G, p) be a framework in \mathbb{R}^3 such that G is planar and p is a regular point. Then every edge is non-redundant.

Since a convex embedding stays a convex embedding under small perturbations, we can make the following analogous statements to Theorems 2 and 4.

Theorem 11 Let (G, p) be a framework in \mathbb{R}^3 such that p is a convex embedding and a regular point, and let uv be a non-redundant edge. Then there exists a framework (G, p') such that p' is a convex embedding and a regular point, ||p'(u) - p'(v)|| is rational, and all other edge lengths remain fixed.

Theorem 12 All planar graphs have an integral convex embedding.

Ziegler [11, Problem 4.18] asks whether every 3polytope has a realization where every edge has rational length. Theorem 12 answers this in the affirmative in the case where the number of edges is maximal. In particular, the technique of perturbing each edge does not necessarily preserve the flatness of a non-triangular face, so we can only answer this problem for 3-polytopes with only triangular faces.

5 Acknowledgements

I would like to thank Dylan Thurston for a helpful discussion on the topic and the anonymous reviewer for his or her suggestions on the paper. I especially want to thank David Surowski, who in his final piece of academic advice, urged me to submit this paper.

References

- L. Asimow and B. Roth, *The rigidity of graphs*, Trans. Amer. Math. Soc. 245 (1978), 279-89.
- [2] T. Biedl, Drawing some planar graphs with integer edge-lengths, to appear in Canadian Conference on Computational Geometry (2011).
- [3] I. Fáry, On straight-line representation of planar graphs, Acta Sci. Math. 11 (1948), 229-233.
- [4] J. Geelen, A. Guo, D. McKinnon, Straight line embeddings of cubic planar graphs with integer edge lengths, J. Graph Theory 58 (2008) No. 3, 270-274.

- [5] H. Gluck, Almost all simply connected closed surfaces are rigid, Geom. Top., Lecture Notes in Math. 438 (1975), 225-239.
- [6] J. Graver, B. Servatius, H. Servatius, *Combina*torial rigidity, Grad. Stud. Math., Vol. 2, Amer. Math. Soc. (1993).
- [7] H. Harborth, A. Kemnitz, M. Moller, A. Sussenbach, Ganzzahlige planare Darstellungen der platonischen Korper, Elem. Math. 42 (1987), 118-122.
- [8] A. Kemnitz, H. Harborth, Plane Integral Drawings of Planar Graphs, Discrete Math. 236 (2001), 191-195.
- [9] G. Laman, On graphs and rigidity of plane skeletal structures, J. Eng. Math. 4 (1970), 331-340.
- [10] K. Wagner, Über eine Eigenschaft der ebenen Komplexe, Math. Ann. 144 (1937), 570-590.
- [11] G. Ziegler, Lectures on polytopes, Springer (1995), 123.

Drawing some planar graphs with integer edge-lengths

Therese Biedl *

Abstract

In this paper, we study drawings of planar graphs such that all edge lengths are integers. It was known that such drawings exist for all planar graphs with maximum degree 3. We give a different proof of this result, which is based on a simple transformation of hexagonal drawings as created by Kant. Moreover, if the graph is 3-connected then the vertices have integer coordinates that are in O(n). We then study some other classes of planar graphs, and show that planar bipartite, seriesparallel graphs, and some other graphs also have planar drawings with integer edge lengths.

1 Introduction

A planar graph is a graph that can be drawn without crossing. Fáry, Stein and Wagner [4, 13, 15] proved independently that every planar graph has a drawing such that all edges are drawn as straight-line segments. Sometimes additional constraints are imposed on the drawings. The most famous one is to have integer coordinates while keeping the area small; it was shown in 1990 that this is always possible in $O(n^2)$ area [5, 12].

In this paper, we study a different restriction that was first posed by Kemnitz and Harborth [8]: Does every planar graph admit a straight-line drawing with integer edge lengths? This question remains open in general, but was answered in the positive for planar graphs with maximum degree 3 by Geelen, Guo and McKinnon [6].

In this paper, we first give a different proof of the result for planar graphs with maximum degree 3. In particular, our proof is constructive and yields a lineartime algorithm to find the drawing. (In contrast to this, Geelen, Guo and McKinnon require a theorem about rational distances that does not lend itself to an algorithm easily.) For 3-connected 3-regular graphs, our algorithm is very easy: Use the drawings with few slopes that are known to exist, and modify them so that all edge lengths are integers. In the resulting drawing all vertices are also at (integer) grid points, and the grid has width and height O(n). For graphs that are not 3-connected, we split them into subgraphs, draw these separately, and paste them together suitably. The proof here is still algorithmic, but no bound on the grid-size of the resulting drawing is apparent.

We also study some other graphs classes, such as planar bipartite graphs and series-parallel graphs. As it turns out, the proof of Geelen et al. [6] actually works for these graphs as well, and so they also can be drawn with integer edge lengths. This was also shown independently (with a different proof) by Sun [14].

2 Drawing 3-connected 3-regular planar graphs

We first study graphs with maximum degree 3 that are also 3-connected, i.e., cannot be separated by removing at most 2 vertices. Such graphs are in fact 3-regular, i.e., every vertex has degree 3. In 1993, Kant [7] showed how to create hexagonal grid drawings of 3-connected 3-regular graphs. His results (cf. Theorem 9 and Figure 5 of [7]) imply:

Theorem 1 [7] Let G be a 3-connected 3-regular graph, and let v_o be an arbitrary vertex on the outer-face of G. Then $G - v_o$ has a straight-line drawing Γ such that

- all edges are drawn horizontally, vertically or with slope -1,
- the drawing is contained in a triangle with corners at (0,0), $(\frac{n-2}{2}-1,0)$ and $(0,\frac{n-2}{2}-1)$,
- the three neighbours of v_o are placed at the three corners of the triangle.

Such a drawing can be found in linear time.

See also Figure 1. A similar result was also proved later by Dujmovic et al. [3] using the so-called canonical ordering.



Figure 1: Kant's drawing (from [7].)

To convert Γ into a drawing with integer edge lengths, skew it suitably, and then add v_o . There are many ways

^{*}David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, e-mail biedl@uwaterloo.ca. Research partially supported by NSERC. The author would like to thank Terry Anderson, Michal Stern, Ruth Urner, and especially Elena Lesvia Ruiz Velazquez for inspiring discussions.

to do such a skew. The most intuitive one would be to convert the drawing back to the hexagonal grid (with angle 0, $\pi/3$ and $2\pi/3$.) This would make the edge lengths integers, but vertices would not be at grid points (and in fact, would have irrational coordinates.)

We hence use a different skew that maps grid points to grid points and lines to slopes that are part of a Pythagorean triplet. More precisely, define the linear mapping $\psi : (x, y) \rightarrow (7x - 3y, 24y)$ and note that it maps grid points to grid points. Consider any line segment s in Γ ; say s connects grid points (x_1, y_1) and (x_2, y_2) :

- If s is horizontal, then $y_1 = y_2$, and hence $\psi(s)$ is also horizontal. Since grid points are mapped to grid points, $\psi(s)$ hence has integer length.
- If s has slope -1 then $x_2 x_1 = y_1 y_2$. Hence $\psi(s)$ has slope

$$= \frac{24y_1 - 24y_2}{(7x_1 - 3y_1) - (7x_2 - 3y_2)}$$
$$= \frac{24(x_2 - x_1)}{7(x_1 - x_2) - 3(x_2 - x_1)}$$
$$= \frac{24}{-10} = -\frac{12}{5}$$

Say $\psi(s)$ projects to length X in x-direction and length Y in y-direction, then $Y = \frac{12}{5}X$ and both X and Y are integers (since ψ maps grid points to grid points.) Hence X = 5A for some integer A, and the length of $\psi(s)$ is $\sqrt{X^2 + Y^2} = \sqrt{(5A)^2 + (12A)^2} = \sqrt{169A^2} = 13A$, which is an integer.

• If s is vertical, then $x_1 = x_2$. Similar calculations show that $\psi(s)$ has slope -24/7, its x-projection has length 7A for some integer A, and the length of $\psi(s)$ is $\sqrt{(7A)^2 + (24A)^2} = 25A$, an integer.

Therefore $\psi(\Gamma)$ is a drawing of $G - v_o$ where all edges have integer length. Hence it only remains to add v_o suitably. In $\psi(\Gamma)$ the three neighbours of v_o are placed at (0,0), $(7\frac{n-2}{2},0)$ and $(-3\frac{n-2}{2},24\frac{n-2}{2})$. Place v_o at $(-3\frac{n-2}{2},-24\frac{n-2}{2})$, which is a grid point. The three edges to its neighbours than have slope 12/5, 24/7 and $+\infty$, respectively. Since the neighbours are also at grid points, this implies (as above) that the edge lengths are integers.

Theorem 2 Every 3-regular 3-connected planar graph has a planar straight-line drawing such that

- all edges have integer length,
- all edges are horizontal, vertical or have slope $\pm \frac{5}{12}$ or $\pm \frac{7}{24}$,
- the width is 5(n-2) and the height is 24(n-2).

Such a drawing can be found in linear time.



Figure 2: Applying ψ and adding v_o .

Without going into details, we note here that a different skew to apply would have been ψ' : $(x, y) \rightarrow$ (7x - 9y, 12y). This maps segments of slope -1 to segments of slope -3/4, and vertical segments to segments of slope -4/3; since $3^2 + 4^2 = 5^2$ one easily shows that edge lengths are then integers. The area of the final drawing then can be shown to be $8(n-2) \times 12(n-2)$, which is less than in Theorem 2 and also has a better aspect ratio. But this drawing would have a larger angle at the "top tip", which will be undesirable later.

In fact, any two Pythagorean triplets a < b < c and a' < b' < c' where $\{a, b\}$ and $\{a', b'\}$ have a term in common can provide a suitable skew. For example, if a' = b, then use the skew $(x, y) \rightarrow (b'x - ay, a'y)$ to obtain a drawing of area $(b' + a)a' \cdot ((n-2)/2)^2$. There are many such pairs of Pythagorean triples, and any of them give $O(n^2)$ area, but is ψ' the best one for the constant in the area-bound?

3 Max-degree-3 graphs

The previous section studied graphs that have maximum degree 3 and are 3-connected. In this section, we now extend this to all graphs of maximum degree 3, i.e., we explain how to deal with a *bridge* (an edge whose removal disconnects the graph) and with a *cutting edge-pair* (a pair of edges whose removal disconnected the graph.) Since the graph has maximum degree 3, it must have either a bridge or a cutting edge-pair or must be 3-connected.

Our approach is the "standard" approach in graph

drawing, also used in [7]: Cut the graph apart at such an edge/pair, draw each part separately, and paste the drawings together suitably. The idea of this is very simple, but the details are a bit more complicated since we need to add invariants to the drawing to ensure that they can be merged.

We will only explain how to obtain a drawing with rational coordinates; this implies the result after scaling.

3.1 Bridges

We will show how to draw any planar graph G with maximal degree 3 with rational edge lengths such that additionally one pre-specified vertex w of G is on the convex hull of the resulting drawing. We proceed by induction on the number of bridges. In the base case, G has no bridge, so it is 2-connected. We will show an even stronger statement for 2-connected graphs in the next subsection.

For the induction step, assume now that G has a bridge $e = (v_1, v_2)$, and let G_1 and G_2 be the two subgraphs that result from removing e, with v_i in G_i . Assume that w belongs to graph G_1 .

Draw G_1 recursively with rational edge lengths such that w is on the convex hull. In this drawing, find an open disk D that is inside the face where G_2 used to be, and such that any point inside D can be connected to v_1 without intersecting other edges of G_1 . Furthermore, if v_1 is on the outer-face of G_1 , choose D so small that w is still on the convex hull of the union of G_1 and D.

Draw G_2 with rational edge lengths such that v_2 is on the convex hull. Shrink the drawing of G_2 , if necessary, so that it can fit inside the open disk D. Then connect v_2 and v_1 , rotating G_2 so that the edge (v_1, v_2) does not intersect it, and shifting G_2 as needed to achieve rational length of the edge (v_1, v_2) . See Figure 3.



Figure 3: Merging the drawing of G_2 into the drawing of G_1 .

3.2 Cutting edge-pairs

So now assume that G has no bridge, but it may have a cutting edge-pair.

In this case, we will show how to draw G with rational edge lengths such that additionally one pre-specified edge (v, w) on the outer-face of G is drawn as the base of a strictly enclosing half-square¹. By this we mean that there exists a triangle T such that (v, w) is drawn on one edge of T, the angles of T at v and w are $\pi/4$, and the rest of the drawing is in the interior of T.

If G has no cutting edge-pair, then it is 3-connected and we can apply the construction of Section 2. Recall that Kant's algorithm allows to choose v_o . If we choose it to be the clockwise first of v and w, then these two vertices will be the leftmost vertices (connected vertically) in Figure 2. By choice of the slopes, the rays of slope -1 and +1 from these vertices will form a halfsquare that contains the whole drawing. So (v, w) is the base of a strictly enclosing half-square as desired.

Now assume that G has cutting edge-pair e_1, e_2 whose removal splits G into graphs G_1 and G_2 . Furthermore, assume that $e_i = (v_i, w_i)$ and $v_i \in G_1$ while $w_i \in G_2$. Assume first that neither e_1 nor e_2 is the edge (v, w)to be drawn as base of a strictly enclosing half-square. After possible renaming, we can then assume that (v, w)belongs to G_1 .

Let G'_1 be the graph obtained from G_1 by adding (v_1, v_2) , if it did not exist already, and draw G'_1 recursively with rational edge lengths and with (v, w) as base of a strictly enclosing half-square. Locate a triangle T' with base at (v_1, v_2) and small enough that it fits inside the face of G_1 where G_2 used to be. Furthermore, if (v_1, v_2) was on the outer-face of G_1 , choose T' so small it fits inside the half-square that strictly encloses the drawing and has (v, w) as base.

Let G'_2 be the graph obtained from G_2 by adding (w_1, w_2) , if it did not exist already, and draw G'_2 recursively with rational edge lengths and with (w_1, w_2) as base of a strictly enclosing half-square.

Shrink the drawing of G'_2 small enough so that it fits inside the interior of T with (w_1, w_2) is parallel to (v_1, v_2) , and then connect v_1 to w_1 and v_2 to w_2 . See Figure 4.

We can justify that this can be done with rational distances for all edges as follows. Let α be the smaller of the angles of T' at v_1 and v_2 . Let $0 < \beta < \alpha$ be an angle such that $\sin(\beta)$ is rational. It is easy to find such a β : Since $(2i)^2 + (i^2 - 1)^2 = (i^2 + 1)^2$, simply choose $\beta = \arctan(2i/(i^2 - 1))$ for large enough integer i; then $\sin(\beta) = 2i/(i^2 + 1)$, which is rational.

Now form an isosceles trapezoid with base (v_1, v_2) and angle β at v_1 and v_2 . Make the non-parallel sides to be of rational length. Then the *top edge* (the shorter of the parallel edges) has also rational length since $\sin(\beta)$ is rational and (v_1, v_2) has rational length. Also, choose the non-parallel sides long enough such the top edge is

 $^{^1{\}rm The}$ term "isosceles right triangle" would be more accurate than "half-square", but also more cumbersome.

so short that a half-square with the top edge as base would still be inside T'. This is possible since $\beta < \alpha$.

Scale the drawing of G_2 such that the (rationallength) edge (w_1, w_2) fits onto the (rational-length) topedge; hence all edges in G_2 are scaled by a rational as desired. The drawing of G_2 then fits entirely inside the half-square on top of the top-edge of the trapezoid, and hence is inside T' and creates no crossings. Therefore the resulting drawing is planar.



Figure 4: Merging the drawing of G_2 into the drawing of G_1 .

A special case occurs if edge (v, w) is one of the edges of the cutting pair, say $v = v_1$ and $w = w_1$. Define the graphs G_1 and G_2 as before, and draw them recursively, drawing (v_1, v_2) and (w_1, w_2) as bases of their respective strictly enclosing half-squares.

Consider the drawing of G_1 . Since it is strictly enclosed by a half-square, we can in fact enclose it in an isosceles triangle where the isosceles angles have size $\alpha_1 < \pi/4$. Similarly define $\alpha_2 < \pi/4$ as the isosceles angle of an isosceles enclosing triangle of G_2 .

Choose β such that $\max\{\alpha_1, \alpha_2\} < \beta < \pi/4$ and such that $\sin(\beta)$ is rational. Such a β exists since there are infinitely many Pythagorean triplets for which the two shorter lengths differ by one [1].

Scale the drawings of G_1 and G_2 such that the edges (v_1, v_2) and (w_1, w_2) have length 1. Now place the drawings of G_1 and G_2 in a trapezoid where the angles at the larger parallel side are β , the non-parallel sides have length 1, and the parallel sides are rational. See Figure 5. One easily verifies all conditions.



Figure 5: Merging the drawings of G_1 and G_2 if (v, w) is part of the cutting edge-pair.

This ends the proof that there exists an rational edgelength drawing in all cases. Since breaking apart a graph can be done in constant amortized time, and finding the appropriate coordinates for placing the subgraphs can be done in constant time, we hence have:

Theorem 3 Any planar graph with maximum degree 3 has a planar drawing with integer edge lengths. Moreover, such a drawing can be found in O(n) time.

We note here that no bound on the coordinates required to achieve integer edge lengths are apparent if the graph has a bridge or a cutting edge-pair. The O(n)' run-time hence only holds under the assumption that arbitrarily small rationals can be handled in constant time. Finding a bound for the coordinates remains an open problem.

4 Graphs with a 3-elimination order

The algorithm that we gave above for finding integer edge-length drawings very much relies on the graph having maximum degree 3. In contrast to this, the proof given by Geelen, Guo and McKinnon [6] only needs a much weaker property of graphs, which we paraphrase as follows:

Definition 1 Let G be a graph. We say that G has a 3-elimination order v_1, \ldots, v_n if

- G has only the 2 vertices v_1, v_2 , or
- v_n has degree at most 2, and v_1, \ldots, v_{n-1} is a 3elimination order for $G - v_n$, or
- v_n has degree 3, and v_1, \ldots, v_{n-1} is a 3-elimination order for $G' = G - v_n \cup (u, w)$, where u and w are two of the neighbours of v_n .

We note here that the neighbours u and w of a vertex v_n of degree 3 can be chosen arbitrarily. Also, this edge is added only if $G - v_n$ does not contain it already.

This 3-elimination order is a stronger concept than the 3-acyclic edge orientation (see for example [2]), where it is only required that v_n has degree at most 3, but no edge between its neighbours is added. On the other hand, it is a weaker concept than the vertex order that defines a 3-tree. Since we will need this concept later, we define it briefly here. A graph is a *k*-tree if it has a vertex order v_1, \ldots, v_n such that v_i , for i > k, has exactly k predecessors and they form a clique. A graph is a *partial k*-tree if it is a subgraph of a k-tree.

Geelen, Guo and McKinnon did not phrase their proof in terms of a 3-elimination order, but following their steps, one can see that this is in fact all they needed, and so they proved:

Theorem 4 [6] Every graph G that has a 3-elimination order has a straight-line drawing Γ with rational edge lengths. Moreover, we can create Γ such that the vertices are placed arbitrarily close to a given drawing Γ' of G. In particular, if G is planar then Γ can be made planar.

Geelen et al. then used the fact that every graph with maximum degree 3 has a 3-elimination order. They also point out that every partial 3-tree has such an order. But actually, this holds for even more graphs, as we will argue now.

Call a graph $G(k, \ell)$ -sparse if any induced subgraph H of G has $|E(H)| \leq \max\{0, k|V(H)| - \ell\}$. Independently of the research in this paper, Sun [14] showed that any (2, 3)-sparse graph has an integer edge-length drawing, using results from rigidity theory. But in fact, it can even be shown for (2, 1)-sparse graphs.

Lemma 5 Any (2,1)-sparse graph G has a 3elimination order.

Proof: We prove this by induction; the claim is trivial if G has only 1 vertex. So assume $n \ge 2$. Since G has at most 2n-1 edges, it has a vertex v of degree at most 3, and we choose this vertex as v_n . G - v is (2, 1)-sparse and so if $\deg(v) \le 2$, we can find a 3-elimination order for G - v by induction, add v_n to it and are done.

If $\deg(v) = 3$ then we add an edge between two neighbours u, w of v, so (2, 1)-sparseness of $G' = G - v \cup \{(u, w)\}$ is not immediately obviously. But we claim that it holds as follows.

Let H' be any induced subgraph of G'. If H' does not contain both u and w, then H' is also an induced subgraph of G and hence $|E(H')| \leq 2|V(H')| - 1$. If H' does contain both u and w, then consider the graph H that is induced by the vertex $V(H') \cup \{v\}$ in graph G. Since G is (2, 1)-sparse, $|E(H)| \leq 2|V(H)| - 1$. Therefore, $|E(H')| = |E(H)| + 1 - 3 \leq 2|V(H)| - 1 + 1 - 3 = 2|V(H)| - 3 = 2|V(H')| - 1$ as desired.

So G' is also (2,1)-sparse and we can find a 3elimination order of it by induction. Adding $v = v_n$ to it gives the desired order. \Box

There are numerous graphs that are known to be (2,1)-sparse, and we list here just a few:

Theorem 6 Any (2, 1)-sparse graph G has a straightline drawing with rational edge lengths that is planar if G is planar. This includes the following graph classes:

- 1. Connected graphs with maximum degree 4 that are not 4-regular.
- 2. Graphs with arboricity 2.
- 3. Planar bipartite graphs.
- 4. Series-parallel graphs, which are the same as graphs of treewidth 2, which are the same as partial 2-trees.
- 5. Outer-planar graphs.

Proof: The main claim follows immediately by combining Theorem 4 with Lemma 5 and scaling to make edge lengths into integers. It remains to argue that the given graph classes are actually (2, 1)-sparse.

- 1. Any connected graph G with maximum degree 4 is (2, 1)-sparse unless it is 4-regular. For G itself has at most 2n 1 edges, and any strict subgraph H of G has at least one vertex with an edge into G H and hence is also not 4-regular.
- 2. A graph of arboricity 2 is a graph whose edges can be split into two forests. It is well-known that this is the same as the set of (2,2)-sparse graphs [9], which are hence (2, 1)-sparse.
- 3. Any planar bipartite graph has arboricity 2 [10].
- 4. A k-tree can easily be shown to have arboricity k: for each vertex v_i , assign the (at most) k edges to predecessors to different forests. Therefore a partial 2-tree has arboricity 2.
- 5. Every outer-planar graph is a series-parallel graph.

5 Conclusion and open problems

In this paper we studied planar straight-line drawings that have integer edge lengths for all edges. It was already known that this exists for all graphs with maximum degree 3; we provided a different proof of this, which is simpler and constructive, especially for 3connected 3-regular graphs. Then we proved the same result for some other classes of planar graphs.

The most pressing open problem concerns whether such drawing exists for all planar graphs. Two subclasses of planar graphs where we strongly believe this to be true are the 4-regular graphs and graphs that are acyclic 3-orientable [2]. How can this be proved for them?

For graphs that have an integer edge-length drawing, can we assume that vertices are additionally at integer coordinates? (This holds for the drawings of Theorem 4, and hence for all graphs classes studied thus far.) If so, are the coordinates bounded as a function of n, and how small can they be made? (We proved linear bounds for 3-connected 3-regular graphs only.)

Other modification of this problem are possible. For example, we could define graphs on a given set of points by adding only those edges that are integers, or perhaps even only in a given set of integers (see also the work by Schnabel [11].) What kind of graphs define these, and do they include all planar graphs? In other words, can we draw any planar graph such that the distance between two points is an integer if and only if the edge between them exists?

References

- C.C. Chen and T.A. Peng. Classroom note: Almostisosceles right-angled triangles. Australasian Journal of Combinatorics, 11:263–267, 1995.
- [2] H. de Fraysseix and P. Ossona de Mendez. Regular orientations, arboricity and augmentation. In *Graph Drawing (GD'94)*, volume 894 of *Lecture Notes in Computer Science*, pages 111–118, 1994.
- [3] V. Dujmovic, D. Eppstein, M. Suderman, and D. Wood. Drawings of planar graphs with few slopes and segments. *Computational Geometry: Theory and Applications*, 38:194–212, 2007.
- [4] I. Fáry. On straight line representation of planar graphs. Acta Scientiarum Mathematicarum (Szeged), 11:229– 233, 1948.
- [5] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
- [6] J. Geelen, A. Guo, and D. McKinnon. Straight line embeddings of cubic planar graphs with integer edge lengths. *Journal of Graph Theory*, 58(3):270–274, 2008.
- [7] G. Kant. Hexagonal grid drawings. In Graph-theoretic concepts in computer science (WG'93), volume 657 of Lecture Notes in Comput. Sci., pages 263–276, 1993.
- [8] A. Kemnitz and H. Harborth. Plane integral drawings of planar graphs. *Discrete Mathematics*, 236:191–195, 2001.
- [9] C. St. J. Nash-Williams. Decomposition of finite graphs into forests. J. London Math. Soc., 39:12, 1964.
- [10] G. Ringel. Two trees in maximal planar bipartite graphs. J. Graph Theory, 17:755–758, 1993.
- [11] K. Schnabel. Representation of graphs by integers. In *Topics in Combinatorics and Graph Theory*, pages 635– 640. Physica-Verlag, 1990.
- [12] W. Schnyder. Embedding planar graphs on the grid. In ACM-SIAM Symposium on Discrete Algorithms (SODA '90), pages 138–148, 1990.
- [13] S. Stein. Convex maps. In Proceedings of the Amercian Mathematical Society, volume 2, pages 464–466, 1951.
- [14] T. Sun. Rigidity-theoretic construction of integral Fary embeddings. In *Canadian Conference on Computational Geometry (CCCG '11)*, 2011. In these proceedings.
- [15] K. Wagner. Bemerkungen zum Vierfarbenproblem. Jahresbericht der Deutschen Mathematiker-Vereinigung, 46:26–32, 1936.

Approximating the Obstacle Number for a Graph Drawing Efficiently^{*}

Deniz Sarıöz[†]

Abstract

An obstacle representation for a (straight-line) graph drawing consists of the positions of the graph vertices together with a set of polygonal obstacles such that every line segment between a pair of non-adjacent vertices intersects some obstacle, while the vertices and edges of the drawing avoid all the obstacles. The obstacle number obs(D) for a graph drawing D is the least number of obstacles in an obstacle representation for it. We present an efficient algorithm for computing the obstacle number for a given graph drawing D with approximation ratio $O(\log obs(D))$. This is achieved by showing that the V-C dimension is bounded for the family of hypergraphs of the underlying transversal problem, and using results from epsilon net theory.

1 Introduction

Let D be a straight-line drawing of a (not necessarily planar) graph G on n vertices in the Euclidean plane with no three graph vertices on the same line. We refer to the open line segment between a pair of non-adjacent graph vertices as a *non-edge* of D. We define an *obstacle representation for* D as the set of vertices of G identified with their positions in D together with a collection of polygons (not necessarily convex) called *obstacles*, such that:

- 1. no edge of D meets any obstacle, and
- 2. every non-edge of D meets at least one obstacle.

Without loss of generality, the vertices of polygonal obstacles taken together with the graph vertices are in *general position* (no three on a line) and no graph vertices are inside any obstacle. Notice that the positions of the graph vertices and the obstacles in such an obstacle representation are sufficient to determine the abstract structure of the graph, based on whether or not a pair of graph vertices can "see" each other. Graphs obtained in this manner are called *visibility graphs*, and they are extensively studied and used in computational geometry, robot motion planning, wireless sensor networks, and mobile ad-hoc networks; see [5, 23, 18, 17, 11, 9].

We define the *obstacle number for* D as the least number of obstacles over all obstacle representations for D. We denote this parameter by obs(D).

Our main contribution is a polynomial-time approximation algorithm that, given a graph drawing D, computes an obstacle representation for D with $O(obs(D) \log obs(D))$ obstacles.

Related notions of (an) obstacle representation of a graph and (the) obstacle number of a graph were first defined by Alpert, Koch, and Laison [1]. An obstacle representation of a graph G is equivalent to an obstacle representation for some (straight-line) drawing D of it. The obstacle number of a graph G is the minimum value of obs(D) attained over all drawings D of G.

The obstacle numbers of certain graphs have been determined exactly; upper bounds have been established for some graph families, and proofs of unboundedness have been offered for others [1, 21, 16, 20, 10]. However, the question of devising a computational procedure to determine or approximate the obstacle number of a graph has to our knowledge not yet been addressed, even in part. The results presented here can be considered a first step in that direction.

In Section 2, we will explicate the connection to hypergraphs defined by intersection, and present some background about hypergraph transversals including notions of V-C dimension and epsilon net. In Section 3, we prove our main result having to do with bounding the V-C dimensions of various hypergraphs. In Section 4 we present a concrete algorithm and discuss its efficacy.

2 Preliminaries

2.1 Intersection Hypergraphs and their Transversals

A hypergraph is a pair (X, \mathcal{F}) in which X is a set of ground elements, and \mathcal{F} is a collection of subsets of X. We introduce the following notation and terminology for intersection hypergraphs. Let X and Y be collections of sets. For each $y \in Y$, let N(y) = $\{x \in X \mid x \cap y \neq \emptyset\}$, and say that y generates N(y). Let $\mathcal{F} = \{N(y) \mid y \in Y\}$. Then (X, \mathcal{F}) is an intersection hypergraph, which we shall denote by H(X, Y)whenever convenient.¹ Similarly, for each $x \in X$, let

^{*}Research supported by NSA grant 47149-0001 and PSC-CUNY grant 63427-0041.

[†]Ph. D. Program in Computer Science, The Graduate Center of The City University of New York (CUNY), sarioz@acm.org

¹In many well-studied geometric hypergraphs H(X, Y), each set in X is a singleton. The intersection of a member of X with a member of Y thus corresponds to the inclusion of the former in

 $N(x) = \{y \in Y \mid x \cap y \neq \emptyset\}$, and say that x generates N(x). Let $\mathcal{F}' = \{N(x) \mid x \in X\}$. The hypergraph (Y, \mathcal{F}') , which we shall denote by H(Y, X) when convenient, is known as the *dual* of the hypergraph H(X, Y).

A transversal of an intersection hypergraph H(X, Y)is a subset $T \subseteq X$ such that every member of Y—that meets some member of X—meets some member of T. Let τ denote the minimum cardinality of a transversal of H(X, Y). The (optimization version of) the hypergraph transversal problem is to compute τ exactly, and this has an equivalent formulation as the set cover problem. The decision version of the hypergraph transversal problem is NP-complete; indeed, the restriction to the case in which every member of Y meets exactly two members of X corresponds to a canonical NP-complete problem, "Vertex Cover."

2.2 Computing the Obstacle Number for a Graph Drawing as a Transversal Problem

For a given graph drawing D, we refer to a connected component of the complement of D as a *face* of the drawing. To rephrase an observation in [1] in our context, in an obstacle representation for D with cardinality obs(D), there can be at most one obstacle per face, for otherwise obstacles in the same face could be merged, contradicting the minimality of obs(D). Hence for any given graph drawing, each polygonal obstacle to be included in a minimal obstacle representation can be considered to be a face of the drawing. If need be, one can compute for each face a representative simple polygon that lies inside the face and meets every non-edge that the face meets. (This is not always a simple matter of perturbing the boundary of a face to lie inside the face, since a face may have holes and so its boundary may be a disconnected set.)

Since an *n*-vertex graph has less than n^2 edges (with $\Omega(n^2)$ edges attainable), its drawing must have less than n^4 faces (with $\Omega(n^4)$ faces attainable). Computing obs(D) is therefore a matter of computing a transversal for the finite intersection hypergraph H(X, Y) where X is the face set of D and Y is the non-edge set of D. Observe that $|X| < n^4$ and $Y < n^2$, with $|X| = \Omega(n^4)$ attainable simultaneously with $|Y| = \Omega(n^2)$. Using a representation of D with integer coordinates represented as signed integers, an incidence matrix representation of H(X, Y) with fewer than n^8 bits (and possibly $\Omega(n^8)$) can be computed using standard techniques [5] in time polynomial in the number of input bits, and in time

poly(n) in the RAM model with unit-cost arithmetic operations. It is important to make this distinction, since coordinates may need to be represented using exponentially many bits in n, see [12]; or more, as discussed in Section 5.

It is well-known that the greedy algorithm for the hypergraph transversal problem, which iteratively adds to an initially empty set T a member $x \in X$ that meets the largest number of sets $y \in Y$ that do not already meet some $x' \in T$, provides a $O(\log |Y|)$ -factor approximation [25]. Thus we have a natural $O(\log n)$ -factor approximation algorithm for our problem of computing the obstacle number for a given drawing.

2.3 Improving the Approximation Ratio

How about doing better? Not only is the approximation ratio tight for this greedy algorithm, but the general hypergraph transversal problem is known to be $o(\log |Y|)$ inapproximable [25]. But it is also well-known [25, 19] that if every member of X meets at most Δ members of Y, then the greedy algorithm attains an approximation ratio of $O(\log \Delta)$. Unfortunately, this does not make our task easier, since it is seen that a face could meet $\Omega(n^2)$ non-edges by considering any drawing of the null graph on n vertices. Nonetheless, many families of hypergraphs arising in geometric settings lend themselves to algorithms with approximation ratios that do not depend on either |X| or |Y| using the following ideas.

In the context of an intersection hypergraph H(X, Y), a set $S \subseteq X$ is said to be *shattered* if for every $A \subseteq S$ there is some $y \in Y$ such that $S \cap N(y) = A$. The size of a largest shattered set is called the *V-C dimension* of H(X, Y), after Vapnik and Chervonenkis who first defined it in [24]. For some hypergraphs in which X and Y are both infinite, the V-C dimension is undefined and said to be infinite. Furthermore, for a family of hypergraphs of the form H(X, Y), even if each hypergraph has finite V-C dimension, there may exist no absolute constant upper bound for the V-C dimension. If there an integer d such that every hypergraph in that family has V-C dimension at most d, we say that the family has bounded V-C dimension.

Let $w : 2^X \to [0,1]$ be an additive weight function with w(X) = 1. For a given $\epsilon > 0$, an ϵ -net (with respect to w) is a set $S \subseteq X$ that is a transversal of $H(X, Y_{\epsilon})$, where $Y_{\epsilon} \subseteq Y$ consists exactly of those members of Yeach of which generates a subset of X with weight at least ϵ . Haussler and Welzl have shown [13] that if the V-C dimension of H(X, Y) is some integer d, then for every $\epsilon > 0$ there is an ϵ -net of size at most $cd\epsilon^{-1} \ln \epsilon^{-1}$, where c is a small constant. This is remarkable because the size of an ϵ -net bears no relation to the sizes of Xor Y. See also [19, 15].

Based on this observation, various—deterministic as well as randomized—efficient algorithms have been pre-

the latter. The members of Y are commonly referred to as *ranges*, especially in hypergraphs in which Y is a natural feature of the geometric space that the "points" of X belong to, e.g., the set of all half-spaces, all balls, or all axis-parallel boxes. We eschew the use of the term *range* since this is not the case for problems we are interested in, and also because our hypergraphs are defined by intersection not limited to inclusion: A set in X can meet two disjoint sets in Y and vice versa.

sented [3, 7, 8, 6] to compute a transversal of size within a tiny constant factor of $d\tau \ln \tau$. In Section 4, we state and analyze a specific algorithm for computing the obstacle number for a graph drawing. For now, we suffice it to say that bounding the V-C dimension for the corresponding hypergraph implies an efficient algorithm with approximation ratio independent of |X| or |Y| (and hence n).

Before we proceed, we state an important fact that we make immediate use of. It is well-known [15] that if the V-C dimension of H(Y, X) is d, then the V-C dimension of H(X, Y) is at most 2^{d+1} . The V-C dimension of a family of hypergraphs is therefore bounded if and only if the V-C dimension of the family of dual hypergraphs is bounded.

In the next section, we show that the V-C dimension is bounded for the family of hypergraphs of the form H(Y, X) where Y is the set of non-edges of a graph drawing and X is the set of faces of that drawing. This implies that the V-C dimension of H(X, Y) (the hypergraph for the transversal problem at hand) is also bounded.

3 Main Results

Theorem 1 The V-C dimension is bounded for the family of hypergraphs of the form H(Y, X) in which Y is the set of non-edges in a straight-line drawing D of a graph, and X is the set of faces of D.

We can replace each face $x \in X$ by a simple path x'inside x that meets every non-edge that x meets and does not meet any non-edges that x does not. This substitution will not alter the hypergraph structure, and the resulting paths x' will be disjoint from one another. Hence Theorem 1 is implied by the following result.

Theorem 2 The V-C dimension is bounded for the family of hypergraphs of the form H(Y, X) in which Y is a set of line segments (with every pair meeting at a single point or not at all) and X is a set of simple paths disjoint from one another.

Proof. Assume for contradiction that for every m, there is a hypergraph H(Y, X) such that Y is a set of m line segments (with every pair meeting at a single point or not at all), X is a set of paths disjoint from one another, and Y is shattered.

Given m, and a pair (Y, X) such that |Y| = m and X shatters Y, let $X_3 \subseteq X$ be a minimal set of paths that generate all the $\binom{m}{3}$ triples in Y. That is, every path in X_3 meets exactly three segments in Y, and for every three segments $i, j, k \in Y$ exactly one path $\pi_{ijk} \in X_3$ meets all three. To keep the following argument simple, without loss of generality, no $\pi \in X_3$ meets any intersection points among the segments, of which there

are at most $\binom{m}{2} = O(m^2)$. If there were such paths in X_3 , we could remove them from X_3 and still be left with at least $\binom{m}{3} - \binom{m}{2} = \Omega(m^3)$ paths. We will now charge



Figure 1: Example of an original path π_{ijk} meeting segments i, j, and k.



Figure 2: The interim path (after erasing from tail).



Figure 3: The final path (after erasing from head too), which will be charged to segment j.

each path $\pi \in X_3$ to a line segment in Y, intuitively, "the middle segment" that it meets. Nothing prevents such a path from going back and forth between three segments, so we need to define this more carefully. For a given path $\pi = \pi_{ijk}$ that meets segments $i, j, k \in Y$

(see Fig. 1), arbitrarily label one end of the path as the tail and the other as the head. Starting from the tail, erase π as long as it still meets all three segments, and stop erasing when erasing any longer would cause the remaining path to intersect fewer segments. The tail is now on one of the three segments, without loss of generality, i (see Fig. 2). Note that the path does not intersect *i* anywhere else but the tail. Now start erasing π from the head in a similar fashion, and stop erasing when erasing any more would cause the remaining path to intersect fewer than three segments. Again, the head must be on some segment when we stop, but it could not be on i by the above observation (see Fig. 3). Without loss of generality, the head is on the segment k. Now notice that the path does not meet k anywhere else but the head. Without changing the shatter property, let us replace π_{ijk} with this shorter version of its former self: starting at i, meeting j but no other segment one or more times, before it ends at k. We charge π_{ijk} to j.

Let \hat{s} be a segment that accumulated the greatest charge at the end of this process. Since at least $\Omega(m^3)$ paths were charged to at most m segments, \hat{s} was charged by at least $\Omega(m^2)$ paths. Let X' denote the set of paths in X_3 that were charged to \hat{s} .



Figure 4: Example of a cell of \mathcal{A} and the segments from the original X' that are inside the cell.

Let \mathcal{A} denote the arrangement of the line segments in $Y \setminus \{\hat{s}\}$. A segment endpoint or an intersection point of two segments is called a *vertex of the arrangement*. An open interval on a segment of the arrangement between two vertices of the arrangement that contains no vertex of the arrangement is called an *edge of the arrangement*. A connected component in the complement of the union of the segments is called a *cell of the arrangement*.

Note that every path in X' starts at an edge of the arrangement \mathcal{A} , ends at another edge of \mathcal{A} , and its interior is fully contained in a cell of \mathcal{A} (see Fig. 4). Let G(X') be the graph with the endpoints of the paths as

the vertex set, and the interiors of the paths as edges. Since the paths are disjoint, clearly, G(X') is planar. Now for each edge of the arrangement, merge the path endpoints on that edge at the midpoint of the edge while making sure that the paths remain interior-disjoint (see Fig. 5). Recall that by Euler's formula a planar graph on n vertices has at most 3n - 6 edges. It is not clear that we have a contradiction yet, since \mathcal{A} may have up to m^2 edges (attained when every pair among the msegments cross). Hence it seems that G(X') may have $\Theta(m^2)$ vertices, so it is plausible that G(X') has $\Theta(m^2)$ edges.



Figure 5: The same cell after merging path endpoints.

However, each edge of G(X') must be contained in a single cell that meets \hat{s} . Might this mean that G(X')has $o(m^2)$ vertices? We know that every vertex of G(X')corresponds to a distinct edge of \mathcal{A} in a cell of \mathcal{A} that meets \hat{s} . Hence, the complexity of the zone² of \hat{s} is an upper bound on |V(G(X'))|. We present two lemmas regarding the complexity of the zone of a line segment in an arrangement of line segments, each of which is sufficient by itself.

Lemma 3 Let \mathcal{A} be an arrangement of n line segments, and let s be another line segment. The zone of s has complexity $O(n^{4/3})$.

Proof. Even if *s* meets all *n* segments of \mathcal{A} , its zone will consist of at most *n* cells including the unbounded cell. The bound then follows from the main result in [2]: That the maximum number of edges bounding *m* cells in an arrangement of *n* line segments in the plane is $O(m^{2/3}n^{2/3} + n\alpha(n) + n\log m)$.

 $^{^{2}}$ The *complexity* of a cell of an arrangement is the number of edges of the arrangement that are incident to it. The *zone* of a segment is the set of cells that it meets, and the complexity of the zone of a segment is the number of edges incident to all the cells that it meets.

Lemma 4 (B. Aronov, personal communication) Let \mathcal{A} be an arrangement of n line segments, and let sbe another line segment. The zone of s has complexity $O(n\alpha(n))$ where α denotes the very slow growing inverse of Ackerman's function.

Proof. Let the shape s' be obtained by enlarging s (e.g. taking the Minkowski sum of *s* with a small enough disk) such that s' meets no vertex of \mathcal{A} that s does not. Obtain a new arrangement \mathcal{A}' of line segments by erasing s' from \mathcal{A} . Doing this will possibly disconnect some original line segments that define \mathcal{A} into two, ending up with an arrangement \mathcal{A}' of at most 2n line segments. Every point of s' is in the same cell of this new arrangement. Every cell in an arrangement of m line segments has complexity $O(m\alpha(m))$ [22]. Hence, every cell of \mathcal{A}' has complexity at most $O(2n\alpha(2n))$, i.e., $O(n\alpha(n))$, including the cell that s is in. Since every edge bordering a cell of \mathcal{A} that s meets corresponds to one or two edges bordering the cell of \mathcal{A}' that s is in, the complexity of the zone of s in \mathcal{A} is at most $O(n\alpha(n))$.

Either lemma implies that the zone of \hat{s} has complexity $o(m^2)$, hence the number of vertices of the planar graph G(X') is $o(m^2)$. But since G(X') has $\Omega(m^2)$ edges, we have a contradiction due to Euler's Formula. Therefore, the V-C dimension is bounded for the family of hypergraphs H(Y, X) in which Y is a set of line segments and X is a set of paths disjoint from one another. \Box

4 Algorithm

We continue using the terminology of having been given a drawing D of an n-vertex graph, with X as the set of faces in D and Y as the set of non-edges in D. For the rest of this section, we assume that D has been processed into the incidence matrix of the hypergraph H(X,Y) with $O(n^8)$ entries in memory, as discussed in Section 2.

First, we present a randomized algorithm modeled around an algorithm presented by Efrat and Har-Peled [6] and relies on the analysis in [4], for a totally unrelated family of hypergraphs, in which an upper bound on the V-C dimension was in fact known. Here we use dovetailing to not only vary k (essentially a guess for τ) but also vary d (essentially a guess for the V-C dimension).

According to the analysis therein, for a given graph drawing D that admits an obstacle representation with τ obstacles, our algorithm will return a traversal of size at most $O(\tau \log \tau)$, and it is clear that its running time is polynomial in n even in the worst case.

In the following C-style pseudocode, a = b means a takes on the value of b, the macro sampleSize(d, k) expands to $\lceil 2dk \lg k \rceil$, and the macro numRounds(k, |X|) expands to $\lceil 8k \lg |X| \rceil$.

ComputeObstacleRepresentation(set_of_faces X,

set_of_nonedges Y) { bestSoln = X; bestSize = |X|; for(deekay = 2; 2 deekay < bestSize; deekay = 2 deekay) { for(k = 2; k \le deekay; k = 2k) { d = deekay / k; if (sampleSize(d, k) \ge bestSize) break; Assign weight 1 to each face in X; for (i = 1; i \le numRounds(k, |X|); i = i + 1) {

- Pick randomly a set S of sampleSize(d, k) obstacles, choosing each obstacle randomly and independently from the face set X according to their weights.
- Check if the obstacles in S together meet all of the non-edges in Y; if so, set bestSoln = S, set bestSize = |S|, and break the innermost loop.
- Else, find a non-edge y that does not meet any obstacle in S, and let N(y) be the set of faces in X that the non-edge y meets.
- Compute ω , the sum of weights of faces in N(y). If $2k\omega \leq$ the sum of weights of all faces in X, double the weight of every face in N(y).

// end for i// end for k// end for deekay return bestSoln;

} // end ComputeObstacleRepresentation

5 Remarks

The problem of computing the obstacle number for a graph drawing D exactly is in NP, since it can be transformed into a hypergraph transversal problem in polynomial time. Hence, a naive deterministic algorithm can compute obs(D) in time $2^{O(n)}$, or if we allow ourselves to be output-sensitive, in time merely $n^{O(obs(D))}$, by trying every k-face combination for every value of k from 0 up to obs(D). Since the submission of the accepted version of this paper, an NP-completeness proof has been scheduled to appear on arXiv [14].

What about the original problem of determining the obstacle number of a given abstract graph on n vertices? If all drawings of a graph could be enumerated up to the incidence matrix of faces versus non-edges, then by using our approximation algorithm in the "inner loop," we could obtain a $O(\log OPT)$ -approximation to the original problem. While this may be viable for small instances (perhaps in conjunction with a distributed approach), we conjecture that this problem lies outside of NP and believe it to be intractable in a centralized model of computation. Our rationale follows.

For some simple order types of *n*-point configurations on the plane, a coordinate representation on an integer lattice needs exponentially many bits in n in order to allow the order type to be inferred [12]. Further, we know that a particular labeled graph has two drawings with different obstacle numbers but vertex sets of the same simple labeled order type. (The dual labeled order type of the $\binom{n}{2}$ connecting lines of n vertices in a drawing appears to be sufficient to determine the obstacle number for the drawing.) Hence, coordinate representations of some drawings for the present purpose seem to require at least exponential storage in n. Some drawing-based certificates will in turn have sizes super-polynomial in the number of bits that represent the abstract graph. It may be tempting to think that a certificate could instead be based on the poly(n) sized incidence matrix of faces versus non-edges, but it seems unlikely that one can decide in polynomial time whether or not the given graph has *some* drawing corresponding to a given incidence matrix.

Acknowledgments

The author is indebted to Boris Aronov, Alon Efrat, Nabil Mustafa, and János Pach for fruitful discussions and feedback, and the anonymous CCCG reviewers for keen comments. Alon Efrat's encouragement in the early stages of this work and pointers to recent publications were especially helpful. The author assumes full responsibility for all errors and omissions.

References

- H. Alpert, C. Koch, and J. Laison. Obstacle numbers of graphs. Discrete Comput. Geom., 44:223–244, 2010.
- [2] B. Aronov, H. Edelsbrunner, L. J. Guibas, and M. Sharir. The number of edges of many faces in a line segment arrangement. *Combinatorica*, 12:261–274, 1992.
- [3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14(4):463–479, 1995.
- [4] K. L. Clarkson. Algorithms for polytope covering and approximation. In Proc. Algorithms and Data Structures, 3rd Workshop, WADS '93, Montréal, Canada, August 11–13, 1993, LNCS 709, pages 246– 252. Springer, 1993.
- [5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry. Algorithms and Applications (2nd ed.). Springer-Verlag, 2000.
- [6] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Info. Proc. Letters*, 100:238–245, December 2006.
- [7] A. Efrat, F. Hoffmann, C. Knauer, K. Kriegel, G. Rote, and C. Wenk. Covering with ellipses. *Algorithmica*, 38:145–160, 2003.

- [8] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Info. Proc. Letters*, 95(2):358–362, 2005.
- [9] A. Frieze, J. Kleinberg, R. Ravi, and W. Debany. Lineof-sight networks. *Combinatorics, Probability and Computing*, 18:145–163, 2009.
- [10] R. Fulek, N. Saeedi, and D. Sariöz. Convex obstacle numbers of outerplanar graphs and bipartite permutation graphs, 2011. arXiv:1104.4656v2 [cs.DM].
- [11] S. K. Ghosh. Visibility Algorithms in the Plane. Cambridge University Press, Cambridge, 2007.
- [12] J. E. Goodman, R. Pollack, and B. Sturmfels. Coordinate representation of order types requires exponential storage. In *Proc. 21st annual ACM Symposium on Theory of Computing*, STOC '89, pages 405–410, New York, NY, USA, 1989. ACM.
- [13] D. Haussler and E. Welzl. ε-nets and simplex range queries. Discrete Comput. Geom., 2:127–151, 1987.
- [14] M. P. Johnson and D. Sariöz. Computing the obstacle number of a plane graph, July 2011. http://arXiv.org.
- [15] J. Matoušek. Lectures on Discrete Geometry. Graduate Texts in Mathematics. Springer, 2002.
- [16] P. Mukkamala, J. Pach, and D. Sariöz. Graphs with large obstacle numbers. In *Graph Theoretic Con*cepts in Computer Science, LNCS 6410, pages 292–303. Springer, 2010.
- [17] J. O'Rourke. Visibility. In Handbook of Discrete and Computational Geometry, CRC Press Ser. Discrete Math. Appl., pages 467–479. CRC, 1997.
- [18] J. O'Rourke. Open problems in the combinatorics of visibility and illumination. In Advances in Discrete and Computational Geometry, volume 223 of Contemp. Math., pages 237–243. AMS, Providence, RI, 1999.
- [19] J. Pach and P. K. Agarwal. Combinatorial Geometry. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., 1995.
- [20] J. Pach and D. Sariöz. Small (2, s)-colorable graphs without 1-obstacle representations, 2010. arXiv:1012.5907v2 [cs.DM].
- [21] J. Pach and D. Sariöz. On the structure of graphs with low obstacle number. *Graphs and Combinatorics*, 27:465–473, 2011.
- [22] R. Pollack, M. Sharir, and S. Sifrony. Separating two simple polygons by a sequence of translations. *Discrete Comput. Geom.*, 3:123–136, January 1988.
- [23] J. Urrutia. Art gallery and illumination problems. In Handbook of Computational Geometry, pages 973–1027. North-Holland, Amsterdam, 2000.
- [24] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies to their probabilities. *Theory Probab. Appl.*, 16(2):264–280, 1971.
- [25] V. V. Vazirani. Approximation Algorithms. Springer, 2004.

A Note on Minimum-Segment Drawings of Planar Graphs

Stephane Durocher *[†]

Debajyoti Mondal *

Rahnuma Islam Nishat[‡]

Sue Whitesides ^{‡§}

Abstract

A straight-line drawing of a planar graph G is a planar drawing of G, where each vertex is mapped to a point on the Euclidean plane and each edge is drawn as a straight line segment. A segment in a straight-line drawing is a maximal set of edges that form a straight line segment. A minimum-segment drawing of G is a straightline drawing of G, where the number of segments is the minimum among all possible straight-line drawings of G. In this paper we prove that it is NP-complete to determine whether a plane graph G has a straight-line drawing with at most k segments, where $k \geq 3$. We also prove that the problem of deciding whether a given partial drawing of G can be extended to a straight-line drawing with at most k segments is NP-complete, even when G is an outerplanar graph. Finally, we investigate a worst-case lower bound on the number of segments required by straight-line drawings of arbitrary spanning trees of a given planar graph.

1 Introduction

A planar graph is a graph that admits a plane embedding. A plane graph is a fixed planar embedding of a planar graph. A straight-line drawing Γ of a planar graph G is an embedding of G in the Euclidean plane, in which each vertex of G is mapped to a distinct point, each edge of G is a straight line segment, and no two edges intersect except possibly at a common endpoint. A segment of Γ is a maximal set of edges in Γ that form a straight line segment. Γ is called a minimum-segment drawing of G if the number of segments in Γ is the minimum possible. Figure 1(a) depicts a plane graph G, Figure 1(b) depicts its straight-line drawing with 13 segments, and Figure 1(c) shows a minimum-segment drawing of G with 7 segments.

Dujmović *et al.* [4] showed that $\eta/2$ segments are necessary and sufficient for a straight-line drawing of a tree, where η is the number of odd degree vertices in the tree.



Figure 1: (a) A plane graph G. (b) A straight-line drawing of G. (c) A minimum-segment drawing of G.

They gave optimal bounds on the number of segments in straight-line drawings of outerplanar graphs, plane 2-trees and plane 3-trees, as well as algorithms for constructing straight-line drawings of planar 3-connected graphs with at most 5n/2 segments, where n is the number of vertices. Later, Samee *et al.* [13] gave a lineartime algorithm for computing minimum-segment drawings of series-parallel graphs, where all the vertices have degree at most three. Recently, Biswas *et al.* [2] gave a linear-time algorithm to obtain minimum-segment convex drawings of 3-connected cubic plane graphs.

A natural question is: what is the time complexity of computing a minimum-segment drawing of a planar graph [2]? Dujmović *et al.* [4] posed the following related questions: (a) Is there a polynomial-time algorithm to draw a given outerplanar graph with the minimum number of segments? (b) What is the minimum c such that every *n*-vertex planar graph has a plane drawing with at most cn + O(1) segments?

In many applications a graph is drawn emphasizing the drawing of one of its spanning trees, and the other edges are displayed on demand [5, 8, 11]. Given an arbitrary spanning tree, one may want to draw it with the minimum number of segments, where the edges that are not in the spanning tree are to be drawn with polylines or curves. Given a planar graph G, we investigate a worst-case lower bound on the number of segments required by straight-line drawings of arbitrary spanning trees of G. For this purpose, we introduce a new graph parameter for planar graphs, which we define as follows: the spanning-tree segment complexity of a planar graph G is the minimum positive integer C such that every spanning tree of G admits a drawing with at most C segments. Observe that any lower bound on C is a lower bound on the number of segments required by straight-line drawings of those spanning trees of G that

^{*}Department of Computer Science, University of Manitoba. {durocher, jyoti}@cs.umanitoba.ca

[†]Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[‡]Department of Computer Science, University of Victoria. {rnishat, sue}@cs.uvic.ca

[§]Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Victoria.

determine the spanning-tree segment complexity of G. For simplicity, in the rest of the paper we use the term *segment complexity* instead of the term spanning-tree segment complexity.

Main results: Our main results are given below.

- (1) Given an arbitrary integer $k \ge 3$, it is NP-complete to decide if a given plane graph has a straight-line drawing with at most k segments (see Section 3).
- (2) It is NP-complete to determine whether a given partial drawing of an outerplanar graph G can be extended to a straight-line drawing of G with at most k segments, even when the partial drawing can be extended to a straight-line drawing of G (see Section 4).
- (3) In Section 5, we derive lower bounds on segment complexities of different classes of planar graphs (see Table 1).

Graph Class	Lower Bound on C
Maximal outerplanar	n/6
Plane 2-tree	n/6
Plane 3-tree	(2n-5)/6
Plane 3-connected	n/8
Plane 4-connected	n/5

Table 1: Lower Bound on Segment Complexity. Here n denotes the number of vertices.

2 Preliminaries

Here we introduce some preliminary definitions.

Let G = (V, E) be a connected simple graph with vertex set V and edge set E. Let v be a vertex in G. We denote the degree of v by deg(v). G is called k-connected, $k \ge 1$, if the minimum number of vertices, whose removal results in a disconnected graph or a single-vertex graph, is k. An *independent set* S is a subset of V, such that no two vertices of S are adjacent.

A plane graph partitions the plane into connected regions, called *faces*. The unbounded face is called the *outer face* and all other faces are called the *inner faces*. The vertices on the boundary of the outer face are called the *outer vertices* and all other vertices are called the *inner vertices*. A *maximal planar graph* is a planar graph, where addition of any edge results in a nonplanar graph.

An outerplanar graph is a planar graph that admits a plane embedding, where all its vertices are on the outer face. We call such an embedding an outerplanar embedding. An outerplanar graph G is called a maximal outerplanar graph if addition of any edge to G results in a graph that does not admit an outerplanar embedding. An arrangement of a set L of n lines is the subdivision of the plane induced by L, where the vertices are the intersection points of the lines. An arrangement A(L) of L is called *simple* if no three lines intersect at the same point and no two lines are parallel. In this paper we consider simple arrangements only. An arrangement graph G(L) is the graph obtained from A(L) by removing the infinite half edges (see Figure 2). The following lemma



Figure 2: (a) An arrangement of 5 lines. (b) Arrangement graph.

gives some properties of arrangement graphs.

Lemma 1 [Bose et al. [3]] Let G be a 2-connected graph, where each vertex has degree at most four. Then G is an arrangement graph of a set of l lines if and only if G admits a straight-line drawing Γ such that:

- 1. Each segment contains l-2 edges.
- 2. All the vertices of degree two and degree three in G are on the outer face of Γ .
- 3. Each vertex of degree two is the endpoint of exactly two segments and each vertex of degree three is the endpoint of exactly one segment. No segment has an endpoint that is a vertex of degree four.
- 4. The number of segments is $l = n_2 + (n_3/2)$, where n_2 and n_3 are the number of vertices of degree two and degree three, respectively.

We call Γ an arrangement drawing of G.

3 Minimum-Segment Drawing

In this section we prove that it is NP-complete to decide whether a plane graph has a straight-line drawing with a given number of segments. We first need the following two lemmas.

Lemma 2 Let G be a graph with l(l-1)/2 vertices and l(l-2) edges, where $l \ge 3$. Let the number of degree two and degree three vertices be n_2 and n_3 , respectively. Then G is an arrangement graph if and only if G admits a straight-line drawing Γ with l segments, where $l = n_2 + (n_3/2)$.

Proof. By Lemma 1, if G is an arrangement graph, then G admits a drawing with $l = n_2 + (n_3/2)$ segments.

We thus assume that Γ is a straight-line drawing of G with l segments and then prove that Γ is an arrangement drawing of G. By Lemma 1, this will imply that G is an arrangement graph.

We first prove that Γ satisfies Property 1 of Lemma 1. Suppose for a contradiction that there exists a segment l' that contains at least l-1 edges. Therefore, l' is intersected by at least l other straight lines. Thus the number of segments in Γ is at least l+1, a contradiction. Thus each segment contains at most l-2 edges. Since the number of edges in Γ is l(l-2) and there are l segments, therefore each segment contains exactly l-2 edges, which proves the property.

We next prove that Γ satisfies Property 2 of Lemma 1. Since each segment of Γ contains l-2 edges, it is intersected by all the other l-1 segments in Γ . Thus Γ contains all pairwise intersections of the l segments and any extension of the segments beyond their endpoints will not create any new crossings. We now claim that the vertices of degree two and three must lie on the outer face of Γ . Suppose for a contradiction that there exists an inner vertex v such that $\deg(v) < 4$. Then extension of the segments at v towards infinity will intersect the outer face of Γ , which contradicts the fact that Γ contains all pairwise intersections of the l segments.

Finally, we prove that Γ satisfies Property 3 of Lemma 1. The number of vertices in G is l(l-1)/2and the number of segments in Γ is l. Thus, Γ contains all pairwise intersections of the l segments and each vertex v in Γ must be an intersection point of two different segments. Consequently, if $\deg(v) = 4$, then v cannot be an endpoint of any of those two different segments. Similarly, if $\deg(v) = 3$, then v is the endpoint of one of those two different segments. If $\deg(v)$ is two, then vmust be the endpoints those two different segments. \Box

Lemma 3 An arrangement drawing of an arrangement graph G is a minimum-segment drawing of G.

Proof. Let G be an arrangement graph of l lines. By Lemma 2, G admits a drawing with at most l segments. We now prove that any straight-line drawing of G contains at least l segments.

Let w be any vertex of G. Observe that if deg(w) = 2, then the two neighbors x and y of w are adjacent. Since wxy form a triangle, in any minimum-segment drawing of G, xw and yw must lie on different segments. Therefore, each vertex of degree two will be the endpoint of at least two segments in any minimum-segment drawing.

If deg(w) = 3, then let x, y and z be the three neighbors of w. Observe that at most two of the edges wx, wy and wz can lie on the same segment, which implies that w must be an endpoint of the segment that contains the remaining edge. Therefore, each vertex of degree

three will be the endpoint of at least one segment in any minimum-segment drawing.

Let the number of vertices of degree two and degree three be n_2 and n_3 , respectively. Then any minimumsegment drawing must contain at least $(2n_2+n_3)/2$ segments. By Lemma 2, $(2n_2+n_3)/2 = l$.

We are now ready to prove that it is NP-complete to decide whether a plane graph admits a straight-line drawing with a given number of segments. We define the MIN-SEG-DRAW problem as follows:

INSTANCE : A plane graph G, where the vertices are uniquely labeled, and an integer $k \geq 3$.

QUESTION : Is there a straight-line drawing Γ of G with at most k segments?

We reduce an NP-hard problem, ARRANGEMENT-GRAPH-RECOGNITION [3], to MIN-SEG-DRAW.

INSTANCE : A plane 2-connected graph G with k(k-1)/2 vertices and k(k-2) edges, where the degree of each vertex of G is at most four and all the vertices of degree two and degree three are on the outer face of G.

QUESTION : Is G an arrangement graph? We now have the following theorem.

Theorem 4 MIN-SEG-DRAW is NP-Complete.

Proof. Given a drawing Γ , we can certify whether Γ is a straight-line drawing with at most k segments in polynomial time. We can also verify in polynomial time whether Γ is a drawing of G or not as follows: We first compare the outer face of G with outer face of Γ . If they are different then Γ is not a drawing of G. Otherwise, for each vertex v, we compare the clockwise ordering of the neighbors of v in Γ with the corresponding ordering of neighbors of v in G. If for any vertex the two orderings are different, then Γ is not a drawing of G. In all other cases Γ is a drawing of G. Thus the problem is in NP.

To prove the problem is NP-hard we reduce ARRANGEMENT-GRAPH-RECOGNITION to MIN-SEG-DRAW. Let G be an instance of ARRANGEMENT-GRAPH-RECOGNITION. We assign a unique label to each vertex of G. The resulting labeled graph G' is an instance of MIN-SEG-DRAW.

By Lemma 2 and Lemma 3, G' is an arrangement graph if and only if G' admits a straight-line drawing with at most k segments. Therefore, the answer to the instance of MIN-SEG-DRAW is the answer to the instance of ARRANGEMENT-GRAPH-RECOGNITION. \Box

4 Minimum-Segment Drawing with Given Partial Drawing

Drawing a graph extending a given partial drawing is a well-studied problem [1, 7]. The problem of deciding whether a given partial drawing can be extended to a straight-line drawing of a given planar graph has been shown to be NP-complete by Patrignani [12]. We show that given a planar graph G and the drawing of a subgraph of G, determining whether the drawing can be extended to a straight-line drawing of G with at most k segments is NP-complete, even when G is outerplanar and the partial drawing can be extended to some straight-line drawing of G. A formal definition of the decision problem is as follows:

INSTANCE : An outerplanar graph G, a straight-line drawing Γ' of a subgraph G' of G such that Γ' can be extended to some straight-line drawing of G, and an integer $k \geq 1$.

QUESTION : Is there a straight-line drawing of G, which includes Γ' , with at most k segments?

We call this problem PARTIAL-MIN-SEG. We prove NP-hardness by reduction from a strongly NP-complete problem 3-PARTITION [6] which is defined as follows.

INSTANCE : A set of 3m nonzero positive integers $S = \{a_1, a_2, \ldots, a_{3m}\}$ and an integer B > 0, where $a_1 + a_2 + \ldots + a_{3m} = mB$ and $B/4 < a_i < B/2, 1 \le i \le 3m$. QUESTION : Can S be partitioned into m subsets S_1, S_2, \ldots, S_m such that $|S_1| = |S_2| = \ldots = |S_m| = 3$ and the sum of the integers in each subset is equal to B?

Observe that the NP-completeness of 3-PARTITION holds even when each integer of S is greater than one.

A fan f is a maximal outerplanar graph with n vertices, where a vertex v has degree n - 1. We call v the *apex* of f and all the other vertices the *path vertices*. We call the edges that are incident to v the *ribs* of f.

We now have the following theorem.

Theorem 5 PARTIAL-MIN-SEG is NP-Complete.

Proof. We can prove that the problem is in NP as in the proof of Theorem 4. We now create an instance of PARTIAL-MIN-SEG from an instance $B, S = \{a_1, \ldots, a_{3m}\}$, of 3-PARTITION, where each integer of S is greater than one.

We construct in polynomial time an outerplanar graph G as in Figure 3(a), where 3m + 2 fans have a common apex v. Each fan f_i , $1 \le i \le 3m$, contains exactly a_i path vertices. There are two more fans f' and f'' which contain m + 1 path vertices and mB + m + 1path vertices, respectively. The size of G is polynomial since 3-PARTITION is strongly NP-complete. We denote by G' the subgraph of G induced by the vertices of f'and f''. We construct a straight-line outerplanar drawing Γ' of G' that satisfies the following (a)–(c).

(a) Let w_1, \ldots, w_{m+1} be the path vertices of f' ordered clockwise around v and let $u_1, u_2, \ldots, u_{mB+m+1}$ be the path vertices of f'' ordered clockwise around v. For each $j, 1 \leq j \leq m+1$, rib (w_j, v) of f' and rib (v, u_i) of f'' form a segment, i=B(j-1)+j. These segments are shown in bold lines in Figure 3(a).

(b) The edges between path vertices of f' and f'' are drawn on two different segments. All the other edges of f'' are drawn as separate segments, which are shown as thin lines in Figure 3(a).

The gray region in Figure 3(a) shows Γ' . By construction, the number of segments in Γ' is k' = mB + m + 3. We can observe that G admits some straight-line drawing that includes Γ' . We now ask whether G admits a straight-line drawing, including Γ' , with at most k = mB + m + 3 + 3m segments. In the following we prove that such a drawing exists if and only if the given instance of 3-PARTITION has a positive answer.

We first assume that the 3-PARTITION we considered has a positive answer. In other words, S can be partitioned into m subsets S_1, S_2, \ldots, S_m such that each S_i , $1 \leq i \leq m$, contains exactly three integers and the sum of the integers in S_i is equal to B. Since each fan f_i , $1 \leq i \leq 3m$, requires at least one new segment to draw the edges between path vertices, any extension of Γ' requires at least k' + 3m = k segments. Let E' be the set of ribs of f'' that are not drawn on the same segment as any rib of f'. To obtain a straight-line drawing of Gwith exactly k segments, we need to draw each rib of each f_i on the same segment as one of the ribs in E'. Let e_1 and e_2 be any two consecutive ribs of f' in Γ' and let e'_1 and e'_2 be the ribs of f'' that are on the same segments as e_1 and e_2 , respectively. Then the number of ribs between e'_1 and e'_2 is B. Let the integers in any $S_i, 1 \leq i \leq m$, be a, b and c, where a + b + c = B. We place the fans that have a, b and c path vertices inside the face bounded by the ribs e_1 and e_2 in Γ' in such a way that each rib of a, b and c shares a segment with one of the ribs of f'' between e'_1 and e'_2 . In this way, we place the three fans with path vertices corresponding to the integers in S_i in the face bounded by the pair of ribs e_i and e_{i+1} , where $1 \leq i \leq m$. The final drawing Γ of G that includes Γ' has exactly k segments.

We now assume that the given instance of 3-PARTITION has a negative answer and hence the set S cannot be partitioned into m subsets as described above. We prove that in that case G does not have a drawing with k or fewer segments including Γ' . Recall that any extension of Γ' to some straight-line drawing of G requires at least k segments. Suppose for a contradiction that G has a drawing Γ including Γ' with exactly k segments. Then each rib of each f_i , $1 \leq i \leq 3m$, must be drawn on the same segment as one of the ribs of E'. Since Γ is a planar drawing of G, each f_i must be placed inside a face bounded by two consecutive ribs of f'. Therefore, the fans f_1, \ldots, f_{3m} are partitioned into m subsets and the total number of ribs for each set of fans must be B. Since $a_i < B/2$, no two fans can cumulatively have *B* ribs. Similarly, since $B/4 < a_i$, four or more fans cumulatively have more than *B* ribs. Therefore, each subset must contain exactly three fans. Hence each subset of fans corresponds to a subset S_i of *S* that contains three integers whose sum is *B*. This gives a solution to the given instance of 3-PARTITION, a contradiction. Therefore, *G* cannot have a drawing with at most *k* segments including Γ' .



Figure 3: Illustration for the proof of Theorem 5.

5 Segment Complexity of Planar Graphs

In this section we give lower bounds on the segment complexities of different classes of planar graphs. Recall that segment complexity of a planar graph G is the minimum positive integer C, such that any spanning tree of G admits a drawing with at most C segments. Dujmović *et al.* [4] proved that if the number of odd degree vertices in a tree is η , then any straight-line drawing of the tree requires at least $\eta/2$ segments. If a spanning tree T of G has x leaves, then $x \leq \eta$ and any straightline drawing of the tree requires at least x/2 segments. Thus we have the following observation.

Observation 1 Let G be a planar graph with a spanning tree T, where the number of leaves is x. Then x/2 is a lower bound on the segment complexity of G.

By Observation 1, we obtain a lower bound on the segment complexity of a planar graph by finding a spanning tree with many leaves. A maximum-leaf spanning tree of a graph G is a spanning tree of G, where the number of leaves is the maximum possible. It is NP-hard to find a maximum-leaf spanning tree in a graph G, even when G is a planar bipartite graph with maximum degree four [10]. In the following we obtain lower bounds on segment complexities for maximal outerplanar graphs, plane 2-trees, plane 3-trees, plane 3-connected graphs and plane 4-connected graphs.

A graph G with n vertices is a k-tree if G satisfies the following properties:

(a) If n = k, then G is the complete graph K_n .

(b) If n > k, then G can be constructed from a ktree G' with n-1 vertices by adding a vertex adjacent to exactly k vertices of G', where the induced graph of these k-vertices is a complete graph.

Every k-tree G = (V, E) admits an ordered partition $\pi = (V_1, V_2, ..., V_m)$ of V that satisfies the following:

(a) V_1 contains k vertices inducing a complete graph and every other partition contains only one vertex.

(b) Let G_k , $1 \le k \le m$, be the subgraph of G induced by $V_1 \cup V_2 \cup \ldots \cup V_k$. Then G_k , k > 1, is a k-tree obtained by adding V_k to G_{k-1} .

Every 2-tree is 3-colorable. The following lemma finds a spanning tree of a plane 2-tree using graph coloring.

Lemma 6 Let G be a plane 2-tree with $n \ge 3$ vertices. Let S be a set of vertices that are assigned the same color c in a 3-coloring of G. Then G - S is a tree.

Proof. Let $\pi = (V_1, V_2, ..., V_m)$ be an ordered partition of V. We use induction on m. The case when m = 1is straightforward since G_1 is K_2 . We thus assume that for each G_i , $1 \leq i \leq m-1$, $G_i - S_i$ is a tree. Now consider $G_m = G$. Let z be the vertex in V_m and let xand y be its neighbors. By the definition of plane 2-tree, x and y are adjacent. We assume that G is colored with colors c_1, c_2, c_3 such that $color(x)=c_1, color(y)=c_2$ and $color(z)=c_3$. If $c=c_3$, then $G-S=G_{m-1}-S_{m-1}$ is a tree by induction. If $c=c_1$ or $c=c_2$, then G-S is formed by connecting vertex z to $G_{m-1}-S_{m-1}$ with exactly one edge. Since $G_{m-1}-S_{m-1}$ is a tree, G-S is a tree. \Box

We use Lemma 6 to prove the following theorem.

Theorem 7 Let G be a maximal outerplanar graph with $n \ge 3$ vertices. Then the segment complexity of G is at least n/6.

Proof. We show that every maximal outerplanar graph G with $n \ge 3$ vertices has a spanning tree T, where the number of leaves in T is at least n/3. By Observation 1, this will prove that the segment complexity of G is at least n/6.

Every maximal outerplanar graph admits a 3coloring. Let S_i , $1 \leq i \leq 3$, be a set of vertices that are assigned color *i* in a 3-coloring of *G*. The set with the maximum cardinality among S_1 , S_2 and S_3 will have at least n/3 vertices. Without loss of generality assume that the set with the maximum cardinality is S_1 , that is, $|S_1| \geq n/3$. Every outerplanar graph is a plane 2tree. Therefore, by Lemma 6, $G - S_1$ is a tree, which we denote by T'.

Let v be a vertex in S_1 . Since S_1 is an independent set and G is connected, there exists an edge (x, v), where xis a vertex of T'. Therefore, we can connect v to x to obtain another tree that contains v as one of its leaves. By making the vertices of S_1 into leaves in T', we can obtain a spanning tree T with at least n/3 leaves. \Box In a similar technique as we used in the proof of Theorem 7 we can prove the following theorem.

Theorem 8 Let G be a plane 2-tree with $n \ge 3$ vertices. Then the segment complexity of G is at least n/6.

Every plane 3-tree G has a spanning tree with at least (2n-5)/3 leaves [14]. Furthermore, Kleitman and West [9] proved that every plane 4-connected graph has a spanning tree with at least 2n/5 leaves, and every plane 3-connected graph has a spanning tree with at least n/4 leaves. We combine these results with Observation 1 to obtain the following theorem.

Theorem 9 The segment complexities of plane 3-trees, plane 4-connected graphs and plane 3-connected graphs are at least (2n-5)/6, n/8 and n/5, respectively.

6 Conclusion

Among other results, we have proved that it is NPcomplete to decide whether a plane graph G has a straight-line drawing with k segments. This motivates finding approximation algorithms for minimum-segment drawings of different classes of planar graphs.

A minimum-segment drawing becomes more visually coherent if we minimize the number of distinct lines that contain the segments of the drawing. We call such a drawing a *minimum-line drawing*. Figures 4(a) and (b) depict two different minimum-segment drawings of a tree, where the number of lines are 7 and 6, respectively. Since the number of distinct slopes used in both figures is two, the problem of computing a minimum-line drawing is different from the problem of minimizing the number of distinct slopes.

Open Problem: Compute non-trivial upper bounds on the number of lines required for minimum-line drawings of different classes of planar graphs.



Figure 4: (a) A minimum-segment drawing. (b) A minimum-segment drawing, which is also a minimum-line drawing. Lines are shown in dotted lines.

Acknowledgment. The authors wish to thank an anonymous reviewer for bringing to our attention an error in our initial submission.

References

[1] P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. In *Proc. of* ACM-SIAM SODA, pages 202–221, 2010.

- [2] S. Biswas, D. Mondal, R. I. Nishat, and M. S. Rahman. Minimum-segment convex drawings of 3-connected cubic plane graphs. In *Proc. of CO-COON*, pages 182–191, 2010.
- [3] P. Bose, H. Everett, and S. K. Wismath. Properties of arrangement graphs. *International Jour*nal of Computational Geometry and Applications, 13(6):447-462, 2003.
- [4] V. Dujmović, D. Eppstein, M. Suderman, and D. R. Wood. Drawings of planar graphs with few slopes and segments. *Computational Geometry: Theory* & Application, 38(3):194–212, 2007.
- [5] The Cooperative Association for Internet Data Analysis. Walrus. http://www.caida.org/ tools/visualization/walrus/.
- [6] M. R. Garey and D. S. Johnson. Computers and intractability. Freeman, San Francisco, 1979.
- [7] E. Di Giacomo, W. Didimo, G. Liotta, H. Meijer, and S. K. Wismath. Point-set embeddings of trees with given partial drawings. *Computational Geom*etry: Theory & Application, 42(6-7):664–676, 2009.
- [8] C. Homan, A. Pavlo, and J. Schull. Smoother transitions between breadth-first-spanning-tree-based drawings. In *Proc. of GD*, pages 442–445, 2006.
- [9] D. J. Kleitman and D. B. West. Spanning trees with many leaves. SIAM Journal on Discrete Mathematics, 4(1):99–106, 1991.
- [10] P. C. Li and M. Toulouse. Variations of the maximum leaf spanning tree problem for bipartite graphs. *Information Processing Letters*, 97:129– 132, 2006.
- [11] T. Munzner. Drawing large graphs with H3Viewer and site manager (system demonstration). In *Proc.* of GD, pages 384–393, 1998.
- [12] M. Patrignani. On extending a partial straight-line drawing. International Journal of Foundations of Computer Science, 17(5):1061–1070, 2006.
- [13] M. A. H. Samee, M. J. Alam, M. A. Adnan, and M. S. Rahman. Minimum segment drawings of series-parallel graphs with the maximum degree three. In *Proc. of GD*, pages 408–419, 2008.
- [14] F. Zickfeld. Geometric and combinatorial structures on graphs. PhD dissertation, Technische Universität Berlin, 2007.

Characterization of Shortest Paths on Directional Frictional Polyhedral Surfaces

Gutemberg Guerra-Filho*

Pedro J. de Rezende[†]

Abstract

In this paper, we address a shortest path problem where an autonomous vehicle moves on a polyhedral surface according to a distance function that depends on the direction of the movement (directional) and on the friction of the space (frictional). This shortest path problem generalizes a hierarchy of problems and finds geometric structure to solve several proximity problems. We perform the characterization of shortest paths for a directional frictional geodesic (DFG) distance function on polyhedral surfaces. We derive the local optimality criterion necessary to solve the corresponding shortest path problem using the continuous Dijkstra algorithm [11]. The derivation of this optimality criterion essentially involves demonstrating the strict convexity of the DFG distance function. This contribution is the most fundamental result that enables all constructions of the continuous Dijkstra algorithm to solve the corresponding DFG shortest path problem.

1 Introduction

Path planning still remains an active field with modern applications such as autonomous vehicles and surveillance systems [1, 5, 7]. Although some effort has been made on path planning in unknown and dynamic environments [4, 19], the shortest path planning problems in known static environments are a fundamental step towards these spaces.

In this paper, we address a shortest path problem where an autonomous vehicle moves on a polyhedral surface according to a distance function that depends on the direction of the movement (directional) and on the friction of the space (frictional). More specifically, the distance function considers the total work done by an external force applied to the vehicle to move it from a point to another. Therefore, we generalize the shortest path problem on polyhedral surfaces [11, 12] to consider the moving direction, friction, and slope.

The continuous Dijkstra paradigm [11] is an algorithm that solves several shortest path problems by simulating the propagation of a wave from a source point to all points in the space. The structure of the wave is updated at discrete events when the wave reaches vertices and edges. This structure consists of *intervals of optimality* subdividing each edge according to the sequences of vertices and edges that uniquely define a path from the source to any point in the interval.

The continuous Dijkstra algorithm finds shortest paths by exploring characteristic properties related to the local behavior of the shortest paths. The characterization of shortest paths consists of the determination of a local optimality criterion. In this paper, we perform the characterization of shortest paths for a directional frictional geodesic (DFG) distance function on polyhedral surfaces. We derive the local optimality criterion necessary to solve the corresponding shortest path problem using the continuous Dijkstra algorithm. The derivation of this optimality criterion essentially involves demonstrating the strict convexity of the DFG distance function.

Section 2 defines the shortest path problem addressed in this paper and the corresponding distance function considered in this problem. Section 3 reduces a hierarchy of shortest path problems to the DFG shortest path problem. In section 4, we present the characterization of geodesics and shortest paths on polyhedral surfaces according to this distance function. Section 5 has our concluding remarks.

2 Directional Frictional Shortest Path Problem

Let S be a polyhedral surface, possibly non-convex, specified by a set of faces, edges, and vertices. We assume that all faces of S are triangles since simple polygons may be triangulated in linear time on the number of vertices [2]. We consider *bounded polyhedral surfaces*, that is, a surface with a finite number of bounded faces.

Based on Newtonian mechanics, we address a shortest path problem related to a point particle moving on the surface S according to a path of minimum resistance. As a *distance function*, we consider the total amount of mechanical work that must be done to move the particle through the path. More specifically, the distance function considers the work done by an *external force* F to move the particle between points on the same face of S. The forces acting on the particle, besides the external

^{*}Department of Computer Science and Engineering, University of Texas at Arlington, guerra@cse.uta.edu.

[†]Institute of Computing, State University of Campinas, rezende@ic.unicamp.br. Partially supported by CNPq grants 483177/2009-1, 473867/2010-9, FAPESP grant 07/52015-0.

force F, are the weight P(|P| = m.g), the normal $N(|N| = |P| \cos \alpha_f)$, and the friction $A(|A| = \mu_f |N|)$, where m is the mass of the particle, g is the acceleration of gravity, α_f is the acute angle corresponding to the slope between a face f and the horizontal plane, and μ_f is the kinetic friction coefficient of face f. We assume a constant kinetic friction coefficient μ_f and a constant slope α_f for all points on each face f of S. The static friction force is ignored. We also assume that μ_f is finite and positive. Thus, we do not consider obstacles or free regions.

The vector sum of the forces weight and normal $(P \oplus N)$ results in a force R_{PN} . This force is contained on the face with direction of the gradient, that is, perpendicular to the intersection between the face and the horizontal plane (see Fig. 1). The forces weightnormal R_{PN} and friction A implies in a resulting force $R_{PNA} = R_{PN} \oplus A$ according to angle β (see Fig. 1). We assume the acceleration of the particle is zero (*i.e.*, a constant positive speed). Therefore, since the resulting force of all forces is null, we conclude that F has the same direction and size of R_{PNA} , but opposite orientation. Thus, the size of force F is:

$$|F| = mg\sqrt{\sin^2 \alpha_f} + 2\mu_f \sin \alpha_f \cos \alpha_f \cos \beta + \mu_f^2 \cos^2 \alpha_f.$$

Figure 1: Resulting force acting on the body.

We define the directional frictional geodesic (DFG) distance function such that each face f is associated with a kinetic friction coefficient $\mu_f > 0$ that specifies the resistance to move on the interior of face f. Similarly, each edge e is associated with a kinetic friction coefficient $\mu_e > 0$. According to the DFG distance, the length of a line segment from a point s to a point t in the edge e is the size of the external force F for $\beta \in \{0, \pi\}$ times the euclidean distance between s and t:

$$mg|\mu_e \cos \alpha_e + \sin \alpha_e \cos \beta||st|,$$

where α_e is the slope angle between e and its orthogonal projection into the horizontal plane. The length of a line segment from a point s to a point t on the interior of face f is the size of force F times the euclidean distance between s and t:

$$mg\sqrt{\sin^2\alpha_f + 2\mu_f\sin\alpha_f\cos\alpha_f\cos\beta + \mu_f^2\cos^2\alpha_f|st|},$$

where β is the angle between the friction force and the weight component projected into the plane of the face.

Formally, the DFG shortest path problem is stated as follows: Given one source point s on a triangulated polyhedral surface S, an assignment of kinetic friction coefficients to edges and faces, and an error tolerance $\varepsilon > 0$; build a structure that allows the computation of an ε -optimal path (according to the DFG distance function) from s to any query point t, such that the path stays on the surface S.

3 A Hierarchy of Shortest Path Problems

Several shortest path problems are reduced to the directional frictional shortest path problem addressed in this paper. The most specific shortest path problem considers the interior of a simple polygon (ISP) [17]. The Euclidean shortest path problem with polygonal obstacles (EPO) [6] consists of finding shortest paths in a plane avoiding a set of disjoint simple polygonal obstacles. This problem generalizes the ISP problem when the complement of the polygon is considered an obstacle. A special case of the EPO problem considers parallel straight line segments (PLS) as obstacles [9]. In a simpler version of the PLS problem, the obstacles are parallel half-lines (PHL) [18]. Another special case of the EPO problem consider only polygonal obstacles with disjoint convex hulls (DCH) [16].

A generalization of the EPO problem consists of finding shortest paths according to the Euclidean metric on the surface of a (possibly non-convex) polyhedron [11]. This problem is called the discrete geodesic problem (DGP). To reduce the EPO problem to the DGP problem, we construct a surface where each obstacle becomes an infinite orthogonal prism whose base is on the plane. A shortest path on this surface between points in the plane is fully contained in the plane and avoids all prisms which corresponds to a shortest path avoiding obstacles. Another special case of the DGP problem, is the shortest path problem on the surface of a convex polyhedron (SCP) [13].

The weighted region problem (WRP) [12] consists of finding a path in a planar subdivision that minimizes the total cost according to a weighted Euclidean metric. The WRP problem generalizes the EPO problem. In this case, the weights associated with the free space and obstacles are 1 and $+\infty$, respectively. A special case of the WRP problem arises when the weights of regions are 0, 1, or $+\infty$ which has applications to the maximum concealment problem (MCP).

The DFG problem addressed in this paper generalizes the DGP problem when there is only one constant force applied to the particle for the whole polyhedral surface. In this case, the total work done to move the particle considers only the weight force. On the other hand, if the polyhedral surface is embedded into a single plane, the only force applied to the particle is the constant friction on each face, hence, the WRP problem is a special case of the DFG problem.

4 Geodesic and Optimal Paths

Assuming that each face has a constant kinetic friction coefficient and a constant slope, the following lemma states that geodesic paths are piecewise linear. A *piecewise linear* path is a path whose intersection with any face is the union of disjoint line segments.

Lemma 1 Let f be a face with $\mu_f > 0$. Let s and t be points on the interior of f. A subpath from s to t fully contained on the interior of f is geodesic if and only if it is a straight line segment.

Proof. If the subpath from s to t is geodesic, then it must be locally optimal with regards to the DFG distance function. We assume that the subpath from s to t is the arc of a differential parametric curve $\Phi : I \to \mathbb{R}^2$ from an open range $I \subset \mathbb{R}$ into \mathbb{R}^2 that represents the plane of face f, where Φ is a function that leads $i \in I$ to a point $\Phi(i) = (x(i), y(i)) \in \mathbb{R}^2$. The tangent vector $\Phi'(i)$ is the vector (x'(i), y'(i)), where x'(i) is the first derivative of x(i) in $i \in I$.

The length d_{i_0,i_1} of the arc of the parametric curve Φ according to the DFG distance function, from $\Phi(i_0) = s$ to $\Phi(i_1) = t$, where $i_0, i_1 \in I$ is [3]:

$$d_{i_0,i_1} = \left| \int_{i_0}^{i_1} F_{x_i} V_x \left| \Phi'(i) \right| di \oplus \int_{i_0}^{i_1} F_{y_i} V_y \left| \Phi'(i) \right| di \right|.$$

The forces $F_{x_i} V_x$ and $F_{y_i} V_y$ are horizontal and vertical components of the external force F_i , where V_x and V_y are the unit vectors in the direction of the axes. We denote by |st| the Euclidean length of the straight line segment \overline{st} . Therefore, we have the following:

$$\left| \int_{i_0}^{i_1} F_{x_i} V_x \left| \Phi'(i) \right| di \right| = -|A| \cos \beta |st|,$$
$$\left| \int_{i_0}^{i_1} F_{y_i} V_y \left| \Phi'(i) \right| di \right| = -|R_{PN}| \int_{i_0}^{i_1} \left| \Phi'(i) \right| di - |A| \sin \beta |st|,$$

where $|\Phi'(i)| = \sqrt{(x'(i))^2 + (y'(i))^2}$ denotes the size of vector $\Phi'(i)$, A is the friction force, and R_{PN} is the weight and normal resulting force.

The length of the line segment \overline{st} according to the DFG distance function is $d_{s,t} = |F_x V_x |st| \oplus F_y V_y |st||$, where

$$\begin{split} |F_x \, V_x \, |st|| &= - |A| \cos \beta \, |st| \,, \\ |F_y \, V_y \, |st|| &= - |R_{PN}| \, |st| - |A| \sin \beta \, |st| \,. \end{split}$$

Thus, we must show that $d_{s,t} \leq d_{i_0,i_1}$. However, since

$$|F_x V_x |st|| = \left| \int_{i_0}^{i_1} F_{x_i} V_x |\Phi'(i)| di \right|,$$

we need only to demonstrate that $|F_y V_y |st|| \leq \left| \int_{i_0}^{i_1} F_{y_i} V_y |\Phi'(i)| \, di \right|$. Since $|st| \leq \int_{i_0}^{i_1} |\Phi'(i)| \, di$, we have $|(-|R_{PN}| \, |st|) V_y| \leq \left| \left(-|R_{PN}| \int_{i_0}^{i_1} |\Phi'(i)| \, di \right) V_y \right|.$

Therefore,

$$\left| \left(-|R_{PN}| |st| - |A| \sin \beta |st| \right) V_y \right| \le \left| \left(-|R_{PN}| \int_{i_0}^{i_1} |\Phi'(i)| \, di - |A| \sin \beta |st| \right) V_y \right|,$$

that is, the length of the line segment \overline{st} is less than or equal to the length of the arc of any parametric curve according to the DFG distance function. Thus, the subpath represented by this arc is geodesic if and only if it is a straight line segment. Furthermore, since the geodesic subpath is a line segment, we can drop our assumption that the curve Φ is differential. \Box

Corollary 4.1 Geodesic paths on polyhedral surfaces according to the DFG distance function are piecewise linear.

The locus of points t on a face f with constant distance δ from a source point s, according to the DFG distance function, consists of a curve defined by the following polar equation: |st| =

$$\frac{\delta}{mg\sqrt{\mu_f^2\cos^2\alpha_f + 2\mu_f\cos\alpha_f\sin\alpha_f\cos\beta_t + \sin^2\alpha_f}},$$

where β_t is the angle between the vector of the force R_{PN} in the direction of the gradient and the vector of the friction force A in the direction of the line segment \overline{st} . This curve has an oval shape. The curve is a circle when $\alpha_f = 0$, but it has a degenerated shape when $mg\sqrt{\mu_f^2\cos^2\alpha_f + 2\mu_f\cos\alpha_f\sin\alpha_f\cos\beta_t + \sin^2\alpha_f} = 0$. This only occurs when $\cos\beta_t = -1$. In this case, we have $(\mu_f\cos\alpha_f - \sin\alpha_f)^2 = 0$, that is, $\mu_f = \tan\alpha_f$. We assume that $\mu_f \neq \tan\alpha_f$ to avoid this degenerated case and to guarantee the strict convexity of the DFG distance function. This locus is a strong evidence of the convexity of the DFG function. However, the algebraic proof ¹ of this fact is necessary to guarantee that there exists a local optimality criterion.

The angle of incidence θ is the acute angle between a segment of a geodesic path that crosses (incoming ray) the boundary of face f and a vector perpendicular to the boundary of f. The angle of refraction θ' is the acute angle between a segment of a geodesic path that crosses (outgoing ray) the boundary of face f' and a vector perpendicular to the boundary of f'.

A geodesic path must pass through the interior of an edge e according to a local optimality criterion for the directional frictional geodesic shortest path problem.

¹This proof is presented in appendix B.

Lemma 2 Let f and f' be two faces that share an edge e. Let s be a point on the interior of f and let t be a point on the interior of f'. Let p be a geodesic path between s and t that passes through only one point x^* in the interior of e, then x^* is uniquely defined.

Proof. The proof consists of solving a minimization problem on the length of a path from a point $s(0, -y_0, z_0)$ on face f to a point $t(x_1, y_1, z_1)$ on face f' passing through only one point x^* in edge $e = f \cap f'$ according to the DFG distance function (see Fig. 2). The faces f and f' are defined by points s, t, and by the edge e. The points in the edge e are projected into the y axis and they have height equal to (ax + b), where a and b are constants.



Figure 2: Local optimality criterion.

We must find the minimum point x^* in the following function ² with a single real variable x:

$$\sqrt{\mu_f^2 \cos^2 \alpha_f + 2\mu_f \cos \alpha_f \sin \alpha_f \cos \beta_{f_x} + \sin^2 \alpha_f} |sx^*| + \sqrt{\mu_{f'}^2 \cos^2 \alpha_{f'} + 2\mu_{f'} \cos \alpha_{f'} \sin \alpha_{f'} \cos \beta_{f'_x} + \sin^2 \alpha_{f'}} |x^*t|,$$

where $\mu_f, \mu_{f'}$ are friction coefficients; $\alpha_f, \alpha_{f'}$ are slope angles between faces and the horizontal plane; and $\beta_{f_x}, \beta_{f'_x}$ are the angles between the friction force A in the direction of the movement and the resulting force R_{PN} in the direction of the gradient for each face, respectively. Note that the angles ³ β_{f_x} and $\beta_{f'_x}$ change according to x.

The convexity of the DFG distance function between points on the same face guarantees that there exists a single point $x_s(x_t)$ in e whose distance from s (to t) is minimum. Therefore, the point x^* must be in the range $[x_s, x_t]$. The determination of the minimum point x_s is achieved using the first derivative of the DFG distance function from s to x^* . Analogously, we find the point x_t . In appendix C, we get the following expression that specifies x_s in function of the angle of incidence θ_s in edge e, where c_{s_0}, \ldots, c_{s_4} are constants:

$$c_{s_0} + c_{s_1} \sin \theta_s + c_{s_2} \sin^2 \theta_s + c_{s_3} \sin^3 \theta_s + c_{s_4} \sin^4 \theta_s = 0.$$

The point x^* is uniquely specified by the first derivative of the DFG function from s to x^* added to the DFG function from x^* to t (see Exp. 5):

$$\frac{c_{5}(\cos\beta_{f_{x^{*}}})^{'}|sx^{*}| + (2c_{5}\cos\beta_{f_{x^{*}}} + c_{6})(|sx^{*}|)^{'}}{\sqrt{2c_{5}\cos\beta_{f_{x^{*}}} + c_{6}}} + c_{7}(\cos\beta_{f_{x^{*}}})^{'}|x^{*}t| + (2c_{7}\cos\beta_{f_{x^{*}}} + c_{8})(|x^{*}t|)^{'}}{\sqrt{2c_{7}\cos\beta_{f_{x^{*}}} + c_{8}}} = 0,$$

where $c_5 = \mu_f \cos \alpha_f \sin \alpha_f$, $c_6 = \mu_f^2 \cos^2 \alpha_f + \sin^2 \alpha_f$, $c_7 = \mu_{f'} \cos \alpha_{f'} \sin \alpha_{f'}$, and $c_8 = \mu_{f'}^2 \cos^2 \alpha_{f'} + \sin^2 \alpha_{f'}$.

We simplify the equation above, seeking an equivalent expression in function of angles θ and θ' (see Exp. 6):

$$\frac{\sin\theta(c_5\cos\beta_{f_{x^*}}+c_6)+c_9}{\sqrt{2c_5\cos\beta_{f_{x^*}}+c_6}} + \frac{\sin\theta'(c_7\cos\beta_{f_{x^*}}+c_8)+c_{10}}{\sqrt{2c_7\cos\beta_{f_{x^*}}+c_8}} = 0,$$
(1)

where $c_9 = \frac{c_5 c_4}{\sqrt{1+a^2}|sq'|}$, q' is the intersection of the straight line passing by s in the same direction of the vector of the force R_{PN} with the edge e, c_{10} is a constant analogous to c_9 related to face f'. Note that $\cos \beta_{f_{x^*}} = \pm \cos \psi \cos \theta \pm \sin \psi \sin \theta$ and $\cos \beta_{f'_{x^*}} = \pm \cos \psi' \cos \theta' \pm \sin \psi' \sin \theta'$, where ψ and ψ' are constants. Therefore, the algebraic expression above uniquely specifies x^* . \Box

The critical angle $\theta_{f,f'}$ for *e* consists of the angle of incidence when the angle of refraction $\theta' = \frac{\pi}{2}$. In this case, the local optimality criterion (see Eq. 1) has the following form:

$$\frac{\sin \theta_{f,f'}(c_5(\pm \cos \psi \cos \theta_{f,f'} \pm \sin \psi \sin \theta_{f,f'}) + c_6) + c_9}{\sqrt{2c_5(\pm \cos \psi \cos \theta_{f,f'} \pm \sin \psi \sin \theta_{f,f'}) + c_6}} + \frac{\pm c_7 \sin \psi' + c_8 + c_{10}}{\sqrt{2c_7 \sin \psi' + c_8}} = 0.$$

Then, analogously to θ_s and $\theta_t,$ the critical angle $\theta_{f,f'}$ is given by the expression

$$c_{f,f'_{0}} + c_{f,f'_{1}} \sin \theta_{f,f'} + c_{f,f'_{2}} \sin^{2} \theta_{f,f'} + c_{f,f'_{2}} \sin^{3} \theta_{f,f'} + c_{f,f'_{4}} \sin^{4} \theta_{f,f'} = 0,$$

where c_{f,f'_i} are constants for $i = 0, \ldots, 4$ (see Eq. 7).

A geodesic path *critically uses* part of an edge e when it reaches the edge e at the critical angle $\theta_{f,e}$ at a point q interior to e, travels along edge e for some distance, and leaves edge e into the interior of f' at the critical angle $\theta_{e,f'}$ at a point r interior to e (see Fig. 3(a)).

A geodesic path is *critically reflected* by e from face f when it is incident to edge e at critical angle $\theta_{f,e}$ at a point q interior to e, travels along edge e for some distance, and exits edge e back into face f at a point r interior to e, leaving the edge at angle $\theta_{e,f}$ (see Fig. 3(b)).

²We prove the strict convexity of this function in appendix B. ³Details about the angles β_{f_x} and $\beta_{f'_x}$ are found in appendix A.



Figure 3: Geodesic paths on an edge.

We generalize Lemma 2 to consider critical angles, that is, paths that can either critically use part of an edge or be critically reflected. In this case, we have a two-variable minimization problem.

Lemma 3 A geodesic path crosses edge $e = f \cap f'$ in one of two ways: either it intersects edge e at one crossing point and satisfies the local optimality criterion at that point, or it hits edge e at a critical angle $\theta_{f,e}$, travels along the edge for some distance, and exits the edge into the other face (into the same face) at a critical angle $\theta_{e,f'}$ ($\theta_{e,f}$).

Proof. The proof consists of solving a convex (nonlinear) programming problem [10] in two real variables xand x', where x and x' are the coordinates of the points q and r at the x axis, respectively (see Fig. 3). Our goal is to minimize the function d(x, x') =

$$\sqrt{\mu_f^2 \cos^2 \alpha_f + 2\mu_f \cos \alpha_f \sin \alpha_f \cos \beta_{fx} + \sin^2 \alpha_f} |sx^*| + (\mu_e \cos \alpha_e - \sin \alpha_e) |x^*x'^*| + \sqrt{\mu_{f'}^2 \cos^2 \alpha_{f'} + 2\mu_{f'} \cos \alpha_{f'} \sin \alpha_{f'} \cos \beta_{fx'} + \sin^2 \alpha_{f'}} |x'^*t|$$

subject to $g(x, x') = x - x' \leq 0$. The Karush-Kuhn-Tucker conditions [8] imply the three relations $\nabla d(x, x') + l \nabla g(x, x') = 0, l g(x, x') = 0, l \geq 0$, where $\nabla d(x, x')$ and $\nabla g(x, x')$ are gradient vectors and l is the Lagrange multiplier. Therefore, if l = 0 we have $\nabla d(x, x') = \left(\frac{\partial d(x, x')}{\partial x}, \frac{\partial d(x, x')}{\partial x'}\right) = 0$, otherwise,

$$\frac{c_5(\cos\beta_{f_x})'|sx|}{\sqrt{c_6 + 2c_5\cos\beta_{f_x}}} + \sqrt{c_6 + 2c_5\cos\beta_{f_x}} (|sx|)' + \sqrt{1 + a^2}(\mu_e\cos\alpha_e - \sin\alpha_e) = 0 \text{ and}$$
$$\frac{c_7(\cos\beta_{f'_x})'|x't|}{\sqrt{c_8 + 2c_7\cos\beta_{f'_x'}}} + \sqrt{c_8 + 2c_7\cos\beta_{f'_{x'}}} (|x't|)' - \sqrt{1 + a^2}(\mu_e\cos\alpha_e - \sin\alpha_e) = 0.$$

Simplifying in terms of critical angles $\theta_{f,e}$ and $\theta_{e,f'}$,

$$\frac{\sin \theta_{f,e} (c_6 + c_5 \cos \beta_{f_x}) + c_9}{\sqrt{c_6 + 2c_5 \cos \beta_{f_x}}} + (\mu_e \cos \alpha_e - \sin \alpha_e) = 0 \text{ and}$$
$$\frac{\sin \theta_{e,f'} (c_8 + c_7 \cos \beta_{f'_{x'}}) + c_{10}}{\sqrt{c_8 + 2c_7 \cos \beta_{f'_{x'}}}} - (\mu_e \cos \alpha_e - \sin \alpha_e) = 0.$$

However, the Lagrange multiplier is not zero if and only if g(x, x') = 0, that is, x = x'. In this case, the path crosses the edge at the single crossing point and satisfies the following local optimality criterion:

$$\frac{(c_6 + c_5 \cos\beta_{f_x})\sin\theta + c_9}{\sqrt{c_6 + 2c_5 \cos\beta_{f_x}}} + \frac{(c_8 + c_7 \cos\beta_{f'_{x'}})\sin\theta' + c_{10}}{\sqrt{c_8 + 2c_7 \cos\beta_{f'_{x'}}}} = 0.$$

Depending on the Lagrange multiplier, a geodesic path either intercepts the edge at a single point $(l \neq 0)$, or travels along the edge and exits at critical angle $\theta_{e,f'}$ (l = 0) according to the local optimality criterion. There is a similar proof for critically reflected paths. \Box

The intersection of a geodesic path p with an edge e is a set, probably empty, of points and segments. These points are called *crossing points* of the edge e for path p and the segments, *shared segments* for e and p.

The convexity of the DFG function uniquely specifies a geodesic path that intercepts an edge sequence.

Lemma 4 If p is a geodesic path from a point s to a point t that intercepts the edge sequence $E = (e_1, \ldots, e_k)$ with $e_i \neq e_{i+1}$ (so that there are no shared segments), then p is the unique geodesic path connecting s to t.

Proof. We show that the function that gives the DFG length of the path from point s to point t intercepting the edge sequence E is a strictly convex function of the crossing points at each edge (see Fig. 4).



Figure 4: A path intercepting edge sequence E.

The DFG function of the length of the path from point s to point t intercepting the edge sequence E is given by $d(q_1, \ldots, q_k) = d_1 + \sum_{i=1}^{k-1} d_{i+1} + d_{k+1} =$

$$\begin{split} & \sqrt{\mu_1^2 \cos^2 \alpha_1 + 2\mu_1 \cos \alpha_1 \sin \alpha_1 \cos \beta_1 + \sin^2 \alpha_1} \, |sq_1| + \\ & \sum_{i=1}^{k-1} \sqrt{\mu_{i+1}^2 \cos^2 \alpha_{i+1} + 2\mu_{i+1} \cos \alpha_{i+1} \sin \alpha_{i+1} \cos \beta_{i+1} + \sin^2 \alpha_{i+1}} \, |q_i q_{i+1}| \\ & + \sqrt{\mu_{k+1}^2 \cos^2 \alpha_{k+1} + 2\mu_{k+1} \cos \alpha_{k+1} \sin \alpha_{k+1} \cos \beta_{k+1} + \sin^2 \alpha_{k+1}} \, |q_k t|, \end{split}$$

where q_i is the crossing point at edge e_i . Our goal is to show that $d(q_1, \ldots, q_k)$ is a strictly convex function of the crossing points at each edge. According to theorems in appendix B, the functions d_1 and d_{k+1} are strictly convex. The function d_{i+1} is strictly convex in two scalar variables that specify the points q_i and q_{i+1} . This follows from the strict convexity of this function when one of the points q_i or q_{i+1} is fixed. Thus, the intersection of an orthogonal plane to the horizontal plane (parallel to x or y axis) with this function implies a strictly convex curve. Generalizing the direction of the orthogonal plane, then the function in two variables is strictly convex.

Since function $d(q_1, \ldots, q_k)$ is a summation of strictly convex functions, then $d(q_1, \ldots, q_k)$ is strictly convex. Therefore, function $d(q_1, \ldots, q_k)$ has a unique global minimum, and any local minimum must be global. Since p is a local minimum, it is also the unique global minimum intercepting the edge sequence E.

A critical point of entry of a geodesic path p into face f consists of a point q (the closer endpoint of a shared segment to the source s) interior to an edge $e = f \cap f'$ when p hits q from the side of f. Similarly, a critical point of exit of path p into face f is a point r interior to $(f \cap f')$ (the further endpoint of a shared segment from the source s) when p goes from r into face f.

Let v and v' be consecutive vertices encountered in the list of points describing a geodesic path p. The characterization of geodesic paths implies that the structure of the subpath of p between v and v' is an alternate list of crossing points and shared segments. A geodesic path p may be uniquely specified by a list of vertices, edges, and faces whose interiors contain a portion of p. Edges and faces may be repeated in this list.

Finally, we have the following characterization of geodesic and shortest paths on polyhedral surfaces according to the DFG function:

Theorem 5 The general form of either a geodesic or a shortest path is a piecewise linear path that goes through an alternating sequence of vertices, (possibly empty) edge sequences, and shared segments, such that the path satisfies the local optimality criterion at each edge along any edge sequence and at the endpoints of each shared segment.

Proof. Follows from Lemmas 2 and 3. \Box

5 Conclusions

We performed the characterization of shortest paths on polyhedral surfaces according to the DFG distance function. We derived the local optimality criterion by showing the strict convexity of this distance function. The DFG shortest path problem generalizes a hierarchy of shortest path problem [11, 12]. This implies in a single framework to address several shortest path problems and in the versatility necessary to consider several applications. Furthermore, this framework finds geometric structures embedded on polyhedral surfaces (i.e., a shortest path Voronoi diagram [14])that allows the solution of proximity problems (closest pair of points, all nearest neighbors, minimum expanding tree) on polyhedral surfaces according to the DFG distance.

References

- P. Bhattacharya and M. Gavrilova. Roadmap-based path planning - Using the Voronoi diagram for a clearance-based shortest path. *IEEE Robotics and Automation Magazine*, 15(2):58–66, 2008.
- [2] B. Chazelle. Triangulating a simple polygon in linear time. Discrete Comput. Geom., 6:485–524, 1991.
- [3] M. P. do Carmo. Elementos de Geometria Diferencial. Ao Livro Tecnico S.A., Rio de Janeiro, 1971.
- [4] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous driving in unknown environments. *Experimental Robotics*, 54:55–64, 2009.
- [5] R. Geraerts and M. Overmars. The corridor map method: Real-time high-quality path planning. In *IEEE Int. Conf. on Robotics and Automation*, pages 1023–1028, 2007.
- [6] J. Hershberger and S. Suri. Efficient computation of Euclidean shortest paths in the plane. In Proc. 34th Annual IEEE Sympos. Found. Comput. Sci. (FOCS 93), pages 508-517, 1993.
- [7] G. Jan, K. Chang, and I. Parberry. Optimal path planning for mobile robot navigation. *IEEE/ASME Trans*actions on Mechatronics, 13(4):451–460, 2008.
- [8] H. W. Kuhn and A. W. Tucker. *Linear inequalities and related systems*. Princeton University Press, Princeton, 1956.
- [9] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14:393– 410, 1984.
- [10] D. G. Lvenberger. Linear and Nonlinear Programming. Addison-Wesley Publishing Company, 1937.
- [11] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal* on Computing, 16:647–668, 1987.
- [12] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal ACM*, 38:18–73, 1991.
- [13] D. M. Mount. On finding shortest paths on convex polyhedra. Technical Report 1495, Department of Computer Science, University of Maryland, 1985.
- [14] D. M. Mount. Voronoi diagrams on the surface of a polyhedron. Technical Report 1496, Department of Computer Science, University of Maryland, 1985.
- [15] A. W. Roberts and D. E. Varberg. Convex Functions. Academic Press, Inc., 1973.
- [16] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters*, 23:71–76, 1986.
- [17] M. I. Shamos. Computational Geometry. Ph.D. thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978.
- [18] M. Tompa. An optimal solution to a wire-routing problem. Journal Comput. Syst. Sci., 23:127–150, 1981.
- [19] J. van den Berg and M. Overmars. Planning the shortest safe path amidst unpredictably moving obstacles. *Tracts in Advanced Robotics*, 47:103–118, 2008.

Memory-Constrained Algorithms for Shortest Path Problems

Tetsuo Asano *

Benjamin Doerr[†]

Abstract

We present an algorithm computing a shortest path between to vertices in a square grid graph with edge weights that uses memory less than linear in the number of vertices (apart from that for storing in the input). For any $\varepsilon > 0$, our algorithm uses a work space of $O(n^{(1/2)+\varepsilon})$ words and runs in $O(n^{O(1/\varepsilon)})$ time.

1 Introduction

It is well known that given a weighted graph of n vertices, the shortest path between any two vertices can be computed in $O(n^2)$ time using work space of O(n) words in addition to arrays keeping graph information, whose total size is O(m+n), where m is the number of edges and $m = O(n^2)$ in general. This is achieved by the original version of Dijstra's algorithms [1].

It is known that the shortest path problem is NLcomplete [2]. In other words, it seems hopeless to have an algorithm for the problem using work space of O(1)words of $O(\log n)$ bits.

What happens if we allow a larger, but sublinear work space? Surprisingly, nothing is known for this question as far as the authors know. In this first work on this problem we prove that there is a sublinear-space algorithm for computing the shortest path in a grid graph of size $\sqrt{n} \times \sqrt{n}$. Our algorithm uses a work space of $O(n^{(1/2)+\varepsilon})$ words and runs in $O(n^{O(1/\varepsilon)})$ time for any fixed $\varepsilon > 0$.

2 Computing the Shortest Path Distance

We first present a space-efficient algorithm for computing the length of the shortest path in a grid graph of size $\sqrt{n} \times \sqrt{n}$ where the source and target vertices are located at the lower left corner and upper right corner of the grid, respectively. Once we know an algorithm for computing the shortest path distance, we can report the shortest path by repeatedly applying the algorithm. Throughout the paper we assume that the length of a word is long enough to keep the shortest path distance for any vertex in a given graph. Let G = (V|E) be a square grid graph of size $\sqrt{n} \times \sqrt{n}$. For simplicity, we assume that n is a square number and thus \sqrt{n} is an integer. Two vertices are neighbors if their L_1 -distance is one. All edges have positive weights.

First we decompose the grid graph G into $k \times k$ small square grid graphs called "block graphs" $S_1(V_1, E_1), S_2(V_2, E_2), \ldots, S_{k^2}(V_{k^2}, E_{k^2})$ of the same size $(\sqrt{n}/k) \times (\sqrt{n}/k)$. These block graphs are ordered arbitrarily as far as every block graph appears exactly once in the order. The edge sets of these graphs form a partition of E, i.e., we have

$$E_i \cap E_j = \emptyset$$
, for any $i \neq j$, and

$$E_1 \cup E_2 \cup \dots \cup E_{k^2} = E.$$

Each vertex set V_i has $O(\sqrt{n}/k)$ boundary vertices which may be common to some other vertex sets, and $O(n/k^2)$ inner vertices which are contained only in the vertex set V_i . Since there are k^2 squares, the total number of boundary vertices is $O(k^2 \times \sqrt{n}/k) = O(k\sqrt{n})$.

Figure 1 shows an example of a grid graph and its decomposition into 6×6 squares together with a shortest path from the lower left corner to the upper right corner. As is seen in the figure, a shortest path may visit a square many times.



Figure 1: An example of a grid graph of size $k \times k$ and its decomposition into subgraphs called squares. The shortest path from the lower left corner to the upper right corner is also shown.

We are now ready to describe a basic algorithm for computing the shortest path distance between two arbitrarily specified vertices in a given grid graph. We first

^{*}School of Information Science, JAIST, Japan, t-asano@jaist.ac.jp

[†]Max-Planck-Institut für Informatik, Germany, doerr@mpi-inf.mpg.de

assume that a source vertex is located at the lower left corner of the grid and a target vertex at the upper right corner. This constraint is removed later.

We execute a simple implementation of Dijkstra's algorithm [1] for small squares again and again. For the implementation we need an array for storing temporary distances from the source vertex. We use two different arrays for this purpose. One is an array C to keep a distance from the source vertex to each boundary vertex in the entire graph G. Its size is $O(k\sqrt{n})$. The array is maintained during the entire algorithm.

The other is an array T to keep a temporary distance from the source vertex (of the whole grid) to each vertex in a square including boundary and inner vertices of the square. It is used for a one-shot implementation of Dijkstra's algorithm for a small square (plus the source vertex of the whole grid). Since each square has the same shape, we can use the array again and again for different squares.

Now, we begin with the lower left square S_1 with the source vertex s in it. After initializing the two arrays C and T with infinity, we set C[s] = 0 for the source vertex s. Then, using the array T for all vertices in S_1 , we apply a simple implementation of Dijkstra's shortest path finding algorithm to the square S_1 . It runs in time quadratic in the number of vertices, that is, in $O(n^2/k^4)$ time. As post-process, we transfer distances of all the boundary vertices of S_1 to the common array C.

Then, we move to the next square S_2 . The initialization step is to transfer distances of boundary vertices of S_2 stored in the common array C to the temporary array T. Distance values for all inner vertices of S_2 are initialized to infinity. Then, we implement Dijkstra's algorithm to the square. As is well known, this algorithm computes a shortest path tree which includes every shortest path from a single source vertex to all other vertices. If you needed a source vertex, you could define an imaginary source vertex and imaginary edges from it to all the boundary vertices of the square whose weights are given by the current distance values of those vertices.

After the square S_2 , we move to S_3 , S_4 , and so on until the last square S_m . We call the entire process stated above "a scan over the grid graph."

What can we expect after a scan over the graph? Let P be the shortest path. The path P passes through a number of squares. Suppose P passes through $S_1 = S_{\sigma_1}, S_{\sigma_2}, \ldots, S_{\sigma_L}$ in this order. For example, the shortest path in Figure 1 is characterized by a sequence $(S_1, S_2, S_5, S_4, S_5, S_4, S_7, S_4, S_7, S_8, S_9, S_6, S_9, S_8, S_5, S_6, S_3, S_6, S_3, S_6, S_9).$

How long is the sequence? There are $O(k^2)$ squares and each square is visited by the sequence at most $O(\sqrt{n}/k)$ times (because this is the number of boundary vertices of a square). Since the shortest path must be simple, we can conclude that the length L of the path Pis bounded by $O(k^2) \times O(\sqrt{n}/k) = O(k\sqrt{n})$. By induction, we also see that after the *i*-th scan over the graph, we have computed the shortest path distance up to all boundary vertices of $S_{\sigma i}$. Thus, the shortest path distance must have been computed after $O(k\sqrt{n})$ -th scan. Each scan is done in $O((n/k^2)^2 \times k^2) = O(n^2/k^2)$ time using work space of $O(k\sqrt{n} + n/k^2)$ words.

Algorithm 1 is a formal description of this basic procedure.

Theorem 1 Given a grid graph of size $O(\sqrt{n}) \times O(\sqrt{n})$ with positive edge weights, we can compute the shortest path distance from the lower left corner to the upper right corner of the grid in $O(n^{2+1/2}/k)$ time using work space of $O(k\sqrt{n} + n/k^2)$ words.

Of course, in an actual implementation of the algorithm we would stop after a whole scan over the grid graph did not result in improving any shortest path distances to boundary vertices.

It is not so hard to extend the algorithm so that it finds the shortest path distance for any two vertices as far as both of them are boundary vertices. We could also extend it to allow inner vertices as source and target vertices.

The work space is minimized to $O(n^{2/3})$ when $k\sqrt{n} = n/k^2$, that is, when $k = n^{1/6}$.

3 Reporting the Shortest Path

Once we have computed the shortest path distance d(s,t) from s to t, we can report the shortest path. When we have computed d(s,t), we have also computed the shortest path distance $d(s,v_i)$ for every boundary vertex v_i not only in the last square S_m , but also in all other squares. Keeping the distances in yet another array D of size $O(n^{2/3})$, we execute Dijkstra's algorithm to the square S_m with the upper right corner vertex t as a new source vertex. Now, for each boundary vertex v_i we have two distances, the global shortest path distance $D(s, v_i)$ from s to v_i and the shortest path distance $d(t, v_i)$ from t to v_i within the square S_m . We choose the boundary vertex v_i of the largest value of $d(v_i, t)$ such that

 $D(s, v_i) + d(v_i, t) = d(s, t).$

Since Dijkstra's algorithm finds all shortest paths from t within the square, we can report the shortest path from t to v_i as the last part of the entire shortest path from s to t.

Our next target is the boundary vertex v_i . To report the shortest path from s to v_i , we repeat the same process again with v_i as a new target vertex instead of t within the square containing v_i which is adjacent to the previous square. Forgetting everything we apply the same algorithm from the scratch again to compute the

Algorithm 1: Basic Algorithm for Computing the Shortest Path Distance between Two Vertices in a Grid Graph.

- **Input**: A grid graph G defined by a $\sqrt{n} \times \sqrt{n}$ grid with weighted edges, assuming \sqrt{n} is an integer.
- Output: The shortest path distance from a source vertex s located at the lower left corner to a target vertex t at the upper right corner.

Decompose the grid into $k\times k$ squares

 $S_1, S_2, \ldots, S_{k^2}$ of the same size.;

Let V_i be a set of vertices in a square S_i for each $i = 1, \ldots, k^2$.;

// Assume each V_i contains exactly

 $\sqrt{n}/k \times \sqrt{n}/k = n/k^2$ vertices.;

Let B_i be a set of vertices of V_i that lie on the boundary of S_i (those vertices shared with other squares), called *boundary vertices* of S_i .;

// The number of boundary vertices of each square is $O(\sqrt{n}/k)$.;

Let $B = B_1 \cup \cdots B_{k^2}$ be the set of all boundary vertices.;

// The total number of boundary vertices is $O(k\sqrt{n})$.;

Define an array C[] for distances to boundary vertices.;

Define an array T[] for distances to vertices in a square.;

for each boundary vertex v_i do $C[i] = \infty$.

 $C[0] = 0. // v_0$ is the source vertex s.; for round = 1 to $k\sqrt{n}$ do

for each Square S_i do for each boundary vertex $v_{i,j} = v_p$ in S_i do T[j] = C[p].for each inner vertex $v_{i,j}$ in S_i do $\[\] T[j] = \infty.$ // Dijkstra's algorithm; while there is an unselected vertex in S_i do Choose a vertex $v_{i,p}$ such that T[p] > 0and T[p] is smallest.; // The source vertex s should be treated exceptionally. for each vertex $v_{i,q}$ in S_i adjacent to $v_{i,p}$ do | if $T[q] > T[p] + w(v_{i,p}, v_{i,q})$ then $T[q] = T[p] + w(v_{i,p}, v_{i,q}).;$ T[p] = -T[p]. // Mark the vertex // Transfer the results into the common array C.; for each boundary vertex $v_{i,j} = v_p$ in S_i do C[p] = -T[j].

return the shortest path distance C[t] of the target vertex v_t at the upper right corner.

shortest path distance from s to v_i . In a similar way, we can report the shortest path from v_i to the next intermediate vertex v_i on the shortest path from v_i to s.

Theorem 2 Given a grid graph of size $O(\sqrt{n}) \times O(\sqrt{n})$ with positive edge weights, we can output the shortest path between any two vertices on the grid in $O(n^3)$ time using work space of $O(k\sqrt{n}+n/k^2)$ words for any value of k with $2 \le k \le n/2$.

Proof. We only show the time complexity of the algorithm described above. Again, the number of iterations is bounded by $O(k\sqrt{n})$. Since each iteration is done in $O((n^{2+1/2}/k)$ time, the total time complexity is $O(n^3)$.

3.1 Some Generalizations

We have assumed that source and target vertices are fixed to two corner vertices of the grid. It is rather easy to remove this constraint. Suppose a source vertex s is an inner vertex of a square S_i . Then, we do nothing for the squares S_1, \ldots, S_{i-1} in the first scan over the given grid graph. Then, we execute Dijkstra's algorithm to the square S_i after initializing the distance for the inner vertex s as 0. Then, we can correctly compute distances from s to all the boundary vertices of S_i within the square. It is just the same for a target vertex. Thus, just small modifications are enough to adapt the previous algorithm to apply for general cases where source and target vertices are arbitrarily specified.

4 Reducing the work space

We have shown that the shortest path distance between any two vertices in a grid graph of size $\sqrt{n} \times \sqrt{n}$ can be computed in $O(n^{2+1/2}/k)$ time using work space of $O(k\sqrt{n} + n/k^2)$ words after decomposing the grid into $k \times k$ equal small squares. We have also shown that the work space is minimized to $O(n^{2/3})$ when $k = n^{1/6}$. Is it possible to reduce the work space? Our answer is Yes.

A basic idea is to introduce recursion. Given a grid graph of size $\sqrt{n} \times \sqrt{n}$, we decompose it into $k \times k$ squares of equal dimensions. We further decompose each square into $k \times k$ small squares of equal dimensions.

At the top level, we have k^2 squares (called level-1 squares) of dimensions $\sqrt{n}/k \times \sqrt{n}/k$. The number of boundary vertices of each level-1 square is $O(\sqrt{n}/k)$. Thus, the total number of boundary vertices at level 1 is $O(k\sqrt{n})$.

Each level-1 square is decomposed into k^2 small squares (level-2 squares) of dimensions $\sqrt{n}/k^2 \times \sqrt{n}/k^2$. There are $O(\sqrt{n}/k^2)$ boundary vertices in each level-2 square, and thus the total number of boundary vertices at level 2 in a level-1 square is $O(\sqrt{n})$. On the other hand, the number of inner vertice in each level-2 square is $O(n/k^4)$.



Figure 2: Hierarchical Decomposition of a grid graph. An example of a two-level decomposition.

As before, we apply Dijkstra's algorithm to all level-1 squares in order, but we do not use inner vertices of each square. We recursively apply the previous algorithm to each level-1 square to compute distances for all boundary vertices in the square using inner vertices of level-2 smaller squares.

Now, the work space we use consists of all boundary vertices at level 1, all boundary vertices at level 2 in the currently active level-1 square and all inner vertices in the currently active level-2 small square and thus it amounts to $O(k\sqrt{n} + \sqrt{n} + n/k^4)$. It is minimized to $O(n^{1/10}\sqrt{n})$ when $k\sqrt{n} = n/k^4$, that is, when $k = n^{1/10}$. The time complexity increases. If we denote by T_2 the time for each level-2 square, then we have

 $T_2 = O((n/k^4)^2) = O(n^2/k^8)$

since we apply quadratic-time Dijkstra's algorithm for a square of size $\sqrt{n}/k^2 \times \sqrt{n}/k^2$.

The time for each level-1 square, denoted by T_1 , is given by

 $T_1 = O(k^2 \times T_2 \times \sqrt{n}) = O(n^{5/2}/k^6)$

since we do scan the square just as before. In the similar way, the time for the entire grid, denoted by T_0 , is given by

 $T_0 = O(k^2 \times T_1 \times k\sqrt{n}) = O(n^3/k^3).$

Therefore, if we set $k = O(n^{1/10})$, then the work space is given by $O(n^{1/10}\sqrt{n})$ and the time complexity by $O(n^3/n^{1/30})$.

We can extend the recursion into level $\ell > 1$. The smallest square at level ℓ has dimensions $\sqrt{n}/k^{\ell} \times \sqrt{n}/k^{\ell}$ and contains $(\sqrt{n}/k^{\ell})^2 = n/k^{2\ell}$ inner vertices. When we apply Dijkstra's algorithm to the smallest square, it takes $O(n/k^{2\ell})$ work space for inner vertices and $O(n^2/k^{4\ell})$ time. For the level $i = 0, 1, \ldots, \ell - 1$, the number of boundary vertices in the level is given by
$$\begin{split} &O(\sqrt{n}/k^{i+1}) \text{ and the time complexity } T_i \text{ for the level } i \\ &\text{ is given by } \\ &T_i = O(k^2 \cdot T_{i+1} \cdot (\sqrt{n}/k^{i+1}) \times k^2) = O(\sqrt{n}/k^{i-3}T_{i+1}). \\ &\text{ Thus, we have } \\ &T_0/T_{\ell-1} = (T_0/T_1) \cdot (T_1/T_2) \cdots (T_{\ell-2}/T_{\ell-1}) = \\ &n^{(\ell-1)/2}/k^{\ell(\ell+1)/2}. \\ &\text{ We also have } \\ &T_\ell = O((n/k^{2\ell})^2) = O(n^2/k^{4\ell}), \\ &\text{ and } \\ &T_{\ell-1} = O(k^2 \cdot T_\ell \cdot (\sqrt{n}/k^\ell) \cdot k^2) = O(n^{5/2}/k^{5\ell-4}). \\ &\text{ Combining the above results, we have } \end{split}$$

$$T_0 = O(\frac{n^{\ell/2+2}}{k^{\ell(\ell+1)/2}}).$$

On the other hand, the total work space S is given by

$$S = O(k\sqrt{n} + \sqrt{n} + \frac{\sqrt{n}}{k} + \dots + \frac{\sqrt{n}}{k^{\ell-2}} + \frac{n}{k^{\ell}})$$

This total work space S is minimized to

$$S = O(n^{\frac{1}{2(\ell+1)}}\sqrt{n})$$

when $k = O(n^{\frac{1}{2(\ell+1)}}).$

Substituting it into T_0 , we have

$$T_0 = O(n^{2 + \frac{\ell^2 + \ell}{4(2\ell + 1)}})$$

Theorem 3 Given a grid graph of size $O(\sqrt{n}) \times O(\sqrt{n})$ with positive edge weights and an integer $\ell > 0$, we can output the shortest path between any two vertices on the grid in $O(n^{2+\frac{\ell^2+\ell}{4(2\ell+1)}})$ time using work space of $O(n^{\frac{1}{2(\ell+1)}}\sqrt{n})$ words.

5 Future Works

One of our future works is to extend the result in this paper to a more general class of graphs. One target class is one of maximal planar graphs. How to use a famous planar separator theorem is important. Another interesting problem is to design a sublinear-space algorithm for computing the shortest path on the Delaunay triangulation of a point set in the plane.

References

- E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik 1, 269-271, 1959.
- [2] O. Goldreich, "Computational Complexity: A Conceptual Perspective," Cambridge University Press, p. 182, 2008.

Finding Optimal Geodesic Bridges Between Two Simple Polygons

Amit M. Bhosle^{*}

Teofilo F. Gonzalez[†]

Abstract

Given two simple polygons P and Q we study the problem of finding an optimal geodesic bridge. The objective is to find a bridge that minimizes the largest distance from any point in P to any point in Q. We present an algorithm that finds an optimal geodesic bridge (of minimum weight) in $O(n^2 \log n)$ time. Our algorithm uses as a subalgorithm a simpler $O(n^2 \log n)$ time algorithm that constructs an optimal geodesic bridge from a point to a polygon.

1 Introduction

We study the problem of finding optimal connections between two disjoint simple polygons. The two polygons may represent islands to be connected by a bridge, the goal is to identify a point on each of the two polygons as the end points of the bridge, such that the longest distance from any point on one island to any point on the other is minimized. In cases where it is possible to have flyover-like bridges, the bridge is a straight line between its two end points (immaterial of whether the two points are mutually visible or not). In other cases, the bridge may need to stay outside the interiors of the two polygons. E.g. if instead of a physical bridge, we intend to find a route for a ferry that connects the two islands, the route would need to stay within the water region that separates the two islands.

In this paper we present an algorithm for the geodesicbridge problem that takes $O(n^2 \log n)$ time. We also present a simpler algorithm for the case when one of the polygons is a single point.

Let P and Q be two disjoint polygons. We use $\rho(X)$ to denote the compact region defined by polygon X, and use $\delta(X)$ to denote the boundary of X. Note that $\rho(X) \cap \delta(X) = \delta(X)$. Formally, for points $p \in \rho(P)$ and $q \in \rho(Q)$ we define the *weight* of the bridge (p,q) as $gd(p, P) + gd_e(p,q) + gd(q,Q)$ which is equal to

 $\max_{p' \in \rho(P)} \{ gd(p', p) \} + gd_e(p, q) + \max_{q' \in \rho(Q)} \{ gd(q, q') \},$ (1)

where $gd_e(p,q)$ denotes the length of the shortest geodesic path from $p \in \rho(P)$ to $q \in \rho(Q)$ that lies completely on the boundary and outside the polygons Pand Q, and gd(x, X) is the shortest geodesic distance between x and the geodesic furthest neighbor of x in the polygon X (i.e., $gd(x, X) = max_{x' \in \rho(X)} \{gd(x, x')\}$, where gd(x, x') is the shortest geodesic distance between x and x' without leaving polygon X). A pair (p,q)that minimizes the above expression is called an *optimal geodesic bridge*. An *optimal Euclidean bridge* is defined similarly, but replacing in Eq. 1 $gd_e(p,q)$ by the Euclidean distance between points p and q, and was studied in Ref. [4]. We take the liberty of sometimes using gd(p,q) to denote the actual path (instead of just its length) from p to q that defines the bridge.

2 Related Work

When the two polygons are convex, an optimal Euclidean bridge is also an optimal geodesic bridge. The problem was first studied by Cai, Xu and Zhu [5] who developed an $O(n^2 \log n)$ time algorithm. They proved that for this case the optimal bridge is between points on the boundary of the (convex) polygons which are visible from each other. Different linear time algorithms have been presented in Refs. [3, 9, 8]. The high-dimensional version of the problem has been studied [9, 11].

A 2-approximation algorithm, which finds a bridge with objective function value at most twice that of the optimal one, for convex polygons is given in Ref. [5]. Note that this approximation algorithm always generates a bridge whose endpoints are *mutually visible*. Ahn, Cheong, and Shin [1] present a $\sqrt{2}$ -approximation algorithm for convex polygons and show that their technique generalizes to multidimensional space as long as P and Q are are convex regions.

Bhosle and Gonzalez [4] showed that the end points of an optimal Euclidean bridge might not be mutually visible when the polygons are not convex. They establish that an optimal Euclidean bridge always exists such that its endpoints lie on the boundaries of the two polygons. Using this critical property, they developed an algorithm to find an optimal Euclidean bridge in $O(n^2 \log n)$ time. For the case when one of the polygons degenerates to a point an optimal Euclidean bridge can be constructed in $O(n \log n)$ time [4].

Kim and Shin [8] developed an algorithm for the vbridge problem where the bridges are restricted to have

^{*}Currently at Amazon.com, 705 5^{th} Ave. S., Seattle, WA -98144, bhosle@alumni.cs.ucsb.edu

[†]Department of Computer Science University of California Santa Barbara, CA 93106, teo@cs.ucsb.edu

their endpoints visible from each other. Note that the optimal v-bridge problem and the optimal bridge problem are identical when the polygons are convex; however, Ref. [4] shows that these problems defined over simple (non-convex) polygons have different solutions. This inequivalence holds even for rectilinear polygons [4]. Currently the fastest algorithm is by Tan [10], which runs in $O(n \log^3 n)$ time. This algorithm is quite complex and it makes substantial use of a hierarchical structure that consists of segment trees, range trees and persistent search trees, and a structure that supports dynamic ray shooting and shortest path queries. A restricted version, where the input polygons are simple, but *rectilinear* and the distance between points is measured by the Manhattan distance or L_1 distance, can be solved in linear time [12].

Kim and Shin [8] show that the approximation strategy given in Ref. [5] also applies to the v-bridge problem when the polygons are not convex. Kim and Shin [8] raise the question as to whether or not a better approximation algorithm exists for the Euclidean bridge and v-bridge problems. An optimal v-bridge has total weight within a factor of two times the weight of an optimal bridge between the two polygons. Furthermore, the bound of two is asymptotically best possible [4].

Bhosle and Gonzalez [4] developed approximation schemes that given any positive integer k construct a bridge with objective function value within a factor of $1 + \frac{2}{k+1}$ times that of the optimal one. The approximation algorithms apply to both the versions of the problem (Euclidean/geodesic bridges). It takes $O(kn \log kn)$ time for the Euclidean bridge problem and $O(k^2n^2)$ for the geodesic bride problem. These approximation algorithms introduce k artificial vertices on each line segment and then find an optimal vertex bridge (both end points must be vertices of the polygons).

3 Point To Polygon Geodesic Bridges

First we identify a set of points on the boundary of the polygon which we call *anchors* and *pseudo anchors*. Then we show that an optimal geodesic bridge must have an endpoint that is a vertex, anchor or pseudo anchor of the polygon. Our algorithm uses this fact to narrow down the search for finding an optimal bridge. A similar approach has been used for the Euclidean bridge problem in [4], but the pseudo anchors for the geodesic bridge problem are different from the ones for the Euclidean bridge algorithm [4].

In Figure 1 the furthest neighbors of point q_1 inside Q_1 are points r and r'. The thick dashed line segments indicate the furthest geodesic paths from q_1 to r and the one from q_1 to r'. These paths are said to consist of a sequence of maximal line segments. As we traverse each of these paths starting at point q_1 the first vertex of the

polygon that we visit (after q_1) is called the *first-vertex* of the corresponding furthest point geodesic path. The line segment from q_1 to the first-vertex is called the *first* link. In the polygon Q_1 in Figure 1, the vertices a and a' are the first-vertices, and the line segments (q_1, a) and (q_1, a') are called the corresponding *first-links*.

A point q on the boundary of Q is called an *anchor* if it is not a vertex of polygon Q, and there are at least two different vertices that are the first-vertex of geodesic furthest paths for q. In Figure 1 both $q_1 \in Q_1$ and $q_2 \in Q_2$ are anchors. A point p located immediately to the left of point q_1 (q_2) has line segment (p, q_1) ((p, q_2)) as its geodesic bridge. Note that point $q_3 \in Q_3$ is not an anchor point because all the geodesic furthest paths for point q have the same first-vertex, which is vertex a. A point p located immediately to the left of point q_3 will not have its geodesic bridge being the line segment (p, q_3) . Figure 2 illustrates an instance of the problem of finding an optimal geodesic bridge from a point p to a polygon Q (defined by the solid lines). The geodesic furthest point of both q_5 and q_6 is q_4 . The first-vertex of the point q_5 is q_2 , and that of q_6 is q_7 .



Figure 1: Points q_1 in Q_1 and q_2 in Q_2 are anchors, but point q_3 in Q_3 is not an anchor.



Figure 2: Geodesic Bridge from a Point to a Simple Polygon

In addition to any of the vertices and anchors of Q,

points like q_5 and q_6 (see Figure 2) can also be the end point at Q of an optimal geodesic bridge. Points of the type q_5 are similar to the *pseudo-anchors* defined for the Euclidean bridge problem in Ref. [4], and are defined as: the first point of intersection of the line connecting point p to a vertex or anchor y of Q with the boundary of Q. Note that though *similar* to the definition of the pseudo-anchors for the Euclidean bridge problem, such pseudo-anchors for the geodesic bridge problem differ in that when traversing the line (p, y) from p to y, in the geodesic bridge cases, the *first* point of intersection with Q is a pseudo-anchor of Q where as in the Euclidean bridge version, the last intersection point with Q was selected as a pseudo-anchor. Actually, if the line (p, y) intersects the polygon Q multiple times, then we can show that none of the intersection points can support an optimal geodesic bridge. This is because a pseudo-anchor of Q can support an optimal bridge *only* if the vertex (y) of Q that induces the pseudo-anchor is its *first vertex* (the first vertex of Q on the path to its geodesic furthest vertex of Q). Otherwise, a better bridge is possible by moving the pseudo-anchor by a small distance along its edge. E.g. In Figure 2, the line (p, q_7) would intersect Q thrice. But q_7 cannot be the *first vertex* of the first intersection point (because they are not *mutually visible*). Among the other intersection points, q_7 can be the first vertex of only the last intersection point. However, by arguments similar to those in the proof of Theorem 2 in Ref. [4], one can show that a geodesic bridge ending at such points can be improved by moving the point by a slight distance in an appropriate direction along the edge. Although we do not need to include the pseudo-anchors generated by these cases (multiple points of intersection with the polygon), we keep them. This avoids the overhead of deleting them, while keeping the asymptotic size of the set of pseudo-anchors the same $(O(n^2))$.

Points of the type q_6 had no significance in the Euclidean bride problem, but they are important for the geodesic bridge problem. Also, these points are independent of the point p, and are defined solely by the polygon Q. Below we formally define these two types of pseudo-anchors.

Definition 1. External Pseudo Anchors: A point q on the boundary of Q that is not a vertex of the polygon nor an anchor point is called an *external pseudo-anchor* (induced by the point p) of Q if there is a vertex or anchor y in Q such that q lies on the line (p, y), and it is the first point on the boundary of Q hit by a ray originating at p in the direction (p, y). In other words, q is the point closest to p among all intersection points between (p, y) and Q.

Definition 2. Internal Pseudo Anchors: A point q on the boundary of Q that is not a vertex of the polygon nor an anchor point is called an *internal pseudo-anchor* of Q if there is a vertex x in Q and a vertex or anchor y, also in Q, such that q lies on the line (x, y), and it is the first point on the boundary of Q hit by a ray originating at x in the direction (x, y). In other words, q is the point closest to x among all intersection points between (x, y) and Q.

For the geodesic bridge problem, the set of pseudoanchors includes external and internal pseudo anchors. In Figure 2, point q_5 is an external pseudo anchor, while point q_6 is an internal pseudo anchor. It is easy to see that there are $O(n^2)$ internal pseudo-anchors that can be computed in $O(n^2 \log n)$ using ray-shooting techniques as discussed in Ref. [4]. Similarly, the O(n) external pseudo-anchors can be computed in $O(n \log n)$ time.

Theorem 1 There is an optimal geodesic bridge from point p to polygon Q whose end point on Q is either a vertex, anchor, or pseudo-anchor from Q.

Proof. Essentially, one can show that if the end point q of an optimal geodesic bridge on polygon Q is not a vertex, anchor or pseudo-anchor of Q, then the bridge obtained by moving q slightly along the edge (in an appropriate direction) has smaller weight than the assumed optimal bridge (a contradiction). The same proof technique is used in the proof for Theorem 2 in Ref. [4] for the Euclidean bridge problem, but the characterization of the set of candidate points for an optimal bridge differs significantly.

Corollary 2 An optimal geodesic bridge from a point p to a simple polygon Q can be computed in $O(n^2 \log n)$ time.

Proof. The proof follows from the fact that the anchors and pseudo-anchors of Q can be found in $O(n^2 \log n)$ time. Furthermore, using the geodesic furthest site Voronoi diagram reported in Ref. [2], the geodesic furthest point for each vertex, anchor or pseudo-anchor can be found in $O(\log n)$ query time per point. The algorithm first constructs the shortest paths tree of the point p to the set of $O(n^2)$ vertices, anchors and pseudoanchors. The algorithm reported in Ref. [6] can be used to build this shortest paths tree in $O(n^2 \log n)$ time. For each candidate bridge end point q that is either a vertex, anchor or pseudo-anchor of Q, the algorithm computes the weight of the bridge as $gd_e(p,q,Q) + gd(q,Q)$, where $gd_e(p,q,Q)$ denotes the weight of the geodesic shortest path from point p to point q in the presence of polygon Q as an obstacle, and gd(q, Q) denotes the distance from q to its geodesic furthest vertex in Q. A candidate end point which minimizes the bridge weight is selected as the final solution. \square

Algorithm Geodesic-Bridge(p,Q) outlines in detail our procedure.

Procedure Geodesic-Bridge(p,Q): point p and

simple polygon ${\tt Q}$

Find all the anchors and pseudo anchors in Q;

- Compute gd(q, Q), the length of a geodesic furthest path in Q for each point q that is a vertex, anchor or pseudo-anchor in Q using the algorithm in [2];
- Construct the shortest path tree rooted at p to the set of vertices, anchors and pseudo anchors of Q using the algorithm in Ref. [6];
- From the tree of shortest paths rooted at p compute the geodesic distance $gd_e(p, u, Q)$ from p to each point u that is a vertex, anchor or pseudo anchor in Q in the presence of obstacle Q;
- for every vertex u that is a vertex, anchor or pseudo anchor of Q do
- Compute the length of the best geodesic bridge with an endpoint at u (endpoint u has the minimum value for $gd_e(p, u, Q) + gd(q, Q)$);

endfor

Return an optimal geodesic bridge; End Procedure Geodesic-Bridge

3.1 Point To Polygon: Additional Bridge Properties

We now discuss some additional properties of the pointpolygon version of the geodesic bridge problem. Though these properties do not result in any improvement to the point-polygon version, they are important for the twopolygon version of the problem.

Let (p, q) define an optimal geodesic bridge from point p to the polygon Q. On the geodesic path from p to q, $gd_e(p,q)$, let the second last vertex of $gd_e(p,q)$ be q^* (we say that the bridge starts at p). I.e., (q^*, q) is the *last edge* of the geodesic path $gd_e(p,q)$. Note that in some cases, the point q^* may be the same as the point p. In others, q^* will be a vertex of the polygon P or the polygon Q. By subpath optimality, we know that (q^*, q) defines an optimal geodesic bridge from the point q^* to the polygon Q. Also, the points q^* and q are mutually visible (otherwise, since $gd_e(p,q)$ is a geodesic shortest path between p and q, q^* could not have been the second last vertex on the bridge). Bridges whose end points are mutually visible are called *visible* bridges. E.g., (q^*, q) is a visible bridge. Finally, note that q^* has to either be a vertex of Q or be the same as the point p. This follows from the fact that a geodesic shortest path *bends* only at vertices of Q, and not at arbitrary points in the plane.

Visible bridges between two simple polygons can be computed in $O(n \log^3 n)$ time using the algorithm reported by Tan in Ref. [10]. The same algorithm can be used for the point to polygon problem when one of the polygons degenerates to a point. However, we use a simpler algorithm for computing optimal visible bridges from a point to a polygon in $O(n \log n)$ time. Let us now outline the algorithm.

The algorithm proceeds by computing the O(n) anchors of the polygon Q. As discussed in Ref. [4], the anchor points of Q can be identified from the geodesic furthest site Voronoi diagram which can be computed in $O(n \log n)$ time using the algorithm in Ref. [2]. For the visible bridge problem, we need to consider only the vertices and anchors of Q that are visible from p and the external pseudo-anchors induced by lines connecting the point p to the vertices and anchors of Q, all of which can be found in $O(n \log n)$ total time. Now our set of candidate bridge end points on the polygon Q contains only O(n) points, and using the techniques described in the proof of Corollary 2, the optimal visible bridge can be found in $O(n \log n)$ time. Note that an optimal visible bridge may not be an optimal geodesic bridge.

The algorithm for finding an optimal visible bridge from a point to a polygon can be combined with the fact that (q^*, q) (here, q^* is the second last vertex of the optimal geodesic bridge (p,q) is an optimal visible bridge to provide a new algorithm for finding an optimal geodesic bridge from a point to a simple polygon. The algorithm begins by precomputing a set of n+1 optimal visible bridges. Every visible bridge to polygon Q originate at the point p, and ends at a vertex of Q. Note that these n+1 points form our set of candidate second last vertices of the bridges. Next, the shortest paths tree rooted at point p to all the vertices of Q is constructed using the algorithm of Ref. [7] or Ref. [6]. This shortest paths tree provides us with the geodesic distance from pto each candidate second last vertex of the bridge. Using the precomputed value of the optimal visible bridge from a candidate second last vertex q^* to the polygon Q, and the geodesic distance from p to q^* , we can determine the weight of the geodesic bridge which has q^* as its second last vertex. Finally, the point q^* which supports the cheapest geodesic bridge defines the optimal geodesic bridge. Note that the time complexity of this algorithm is $O(n^2 \log n)$, which is the same as the previous one (Corollary 2). The time complexity is dominated by the time required to precompute the optimal visible bridges from all the candidate second last vertices to Q.

4 Geodesic Bridge Between Two Simple Polygons

We now discuss the more general version of the problem which asks to find an optimal geodesic bridge between two simple polygons.

Figure 3 illustrates an instance of the problem of finding a geodesic bridge between two simple polygons. The figure shows two geodesic bridges, (p_1, q_5) and (p_4, q_6) , between the polygons P and Q. In this figure, p_4 and q_6 are internal pseudo-anchors of P and Q respectively, while q_5 is an external pseudo-anchor of Q.



Figure 3: Geodesic Bridge Between Two Simple Polygons

We define the anchors and internal pseudo-anchors in the same way as for the point to polygon version of the problem. As in the Euclidean bridge case, the set of external psuedo anchors now has $O(n^2)$ points as follows.

Definition 3. External Pseudo Anchors: A point p on the boundary of P that is not a vertex of the polygon nor an anchor point is called an *external pseudo-anchor* of P if there is a vertex x in P and a vertex or anchor yin Q such that p lies on the line (x, y), and it is the first point on the boundary of P hit by a ray originating at y in the direction (y, x). In other words, p is the point closest to y among all intersection points between (x, y)and P.

We define the external pseudo anchors for Q in a similar way, and use the term *pseudo-anchors* to refer to the *union* of internal and external pseudo anchors.

Note that as in the case of the point to polygon geodesic bridge problem, if the line (x, y) intersects P multiple times, none of the intersection points can support an optimal geodesic bridge. However, to keep the algorithm simple, we include such points as well (moreover, the asymptotic size of the set of pseudo-anchors remains $O(n^2)$ even after eliminating them).

We now state the following theorem that limits the set of points on the boundaries of the two simple polygons P and Q that can possibly support an optimal geodesic bridge.

Theorem 3 There is an optimal geodesic bridge whose end points are vertices, anchors, or pseudo-anchors from P and Q.

Proof. The proof is a generalization of the proof of Theorem 1. $\hfill \Box$

The above theorem directly implies an $\tilde{O}(n^4)$ time algorithm for the optimal geodesic bridge¹. The algorithm begins by building the geodesic furthest-site Voronoi diagram for the polygons P and Q. For each vertex, anchor or pseudo-anchor of P and Q, find their geodesic furthest neighbors in their respective polygons. Next, for each vertex, anchor and pseudo-anchor of P, build the shortest paths tree to the $O(n^2)$ vertices, anchors and pseudo-anchors of Q. Finally, for a candidate pair (p,q), find the weight of the bridge with p and q as the end points as $gd(p, P) + gd_e(p, q, P, Q) + gd(q, Q)$, where $gd_e(p, q, P, Q)$ denotes the geodesic distance between p and q in presence of polygonal obstacles P and Q, and select the pair with the minimum weight. It is easy to verify that this algorithm runs in $\tilde{O}(n^4)$ time.

4.1 Algorithm for Finding an Optimal Geodesic Bridge between Two Simple Polygons

Before we discuss our efficient algorithm for finding an optimal geodesic bridge connecting two simple polygons P and Q in $O(n^2 \log n)$ time, we establish an important property of optimal geodesic bridges.

Lets go back to the properties discussed in Section 3.1 for the second last vertices of the bridge. Let p^* and q^* be the second and second last vertices on the geodesic bridge defined by points $p \in P$ and $q \in Q$, when traversing the path $gd_e(p,q)$ from the point p to the point q. By previous arguments, (p^*, p) is an optimal visible bridge from the point p^* to the polygon P, and (q^*, q) is an optimal visible bridge from the point q^* to Q. Note that in some cases, the points p^* and q^* may overlap with the points p and/or q. Also, it may be possible for the the point p^* (resp. q^*) to lie on the polygon Q (resp. P). In general, at least one of the two points p^* and q^* is a vertex. In the only case when neither of these is a vertex, the optimal geodesic bridge (p,q) is in fact a *visible* bridge. The algorithm by Tan [10] finds an optimal visible bridge in $O(n \log^3 n)$ time.

The algorithm first precomputes an optimal visible bridge from each vertex in P and Q to both the polygons P and Q. If r is a vertex of P or Q, let $\nu(r, X)$ denote an optimal visible bridge from r to the polygon X, for $X \in \{P, Q\}$. When computing a visible bridge from r to the polygon X, we consider only the pseudoanchors *induced* by r and the vertices and anchors of X. Furthermore, if a vertex or anchor of X is not visible from r (i.e. the line segment connecting r to the vertex or anchor of the polygon X intersects the other polygon before intersecting X), we ignore the pseudo-anchor for the simple reason that r cannot have a *visible* bridge in conjunction with this pseudo-anchor. E.g. In Figure 3, the line segment (p_6, q_2) intersects P before intersecting Q, and an optimal visible bridge from p_6 to Q cannot have this pseudo-anchor as the other end point of q_6 's visible bridge. If none of the vertices and anchors of the polygon X are visible from r, $\nu(r, X)$ is considered to have a weight of ∞ .

Every optimal geodesic bridge falls into one of the

 $^{{}^1\}bar{O}(f(n)),$ where f(n) is a polynomial function in n, is used to denote $O(f(n)\cdot polylog(n))$

following three categories.

- 1. One-link bridges: Such bridges have a single edge in the geodesic shortest path between its end points.
- 2. Two-link bridges: Such bridges have two edges in the geodesic shortest path between its end points.
- 3. Multi-link bridges: Such bridges have more than two edges in the geodesic shortest path between its end points.

Clearly, one-link bridges are visible bridges, and an optimal visible bridge can be found in $O(n \log^3 n)$ time using Tan's algorithm [10].

In the case of two-link bridges, let v be the vertex of P or Q where the two edges of $gd_e(p,q)$ meet. Note that in such a case, $v = p^* = q^*$. By previous arguments, (v, p) and (v, q) define the optimal visible bridges from the point v to the polygons P and Q respectively. There can be at most 2n such bridges - one for each vertex of Pand Q. Once we have precomputed the optimal visible bridges from each vertex of P and Q to both the polygons P and Q, the weights of these 2n bridges can be computed in constant time per vertex v. As discussed earlier, an optimal visible bridge from a point to a polygon can be computed in $O(n \log n)$ time. Consequently, the weights of all these O(n) bridges can be computed in $O(n^2 \log n)$ time.

Finally, the multi-link bridges have two distinct p^* and q^* points which are respectively the second and second last vertices on $gd_e(p,q)$ when traversing the path $gd_e(p,q)$ from p to q. Also, in such bridges, both p^* and q^* are vertices of P or Q. Given the set of 2n vertices, we can compute the shortest geodesic paths between each of the $O(n^2)$ pairs of points in $O(n^2 \log n)$ time using the algorithm reported in Ref. [6]. For each pair of candidate points p^* and q^* , the optimal bridge with these two points as the second and second last vertices is the better of $\nu(p^*, P) + gd(p^*, q^*, P, Q) + \nu(q^*, Q)$ and $\nu(p^*, Q) + gd(p^*, q^*, P, Q) + \nu(q^*, P)$.

We state below the main theorem of this section.

Theorem 4 Given two simple polygons, P and Q, an optimal geodesic bridge connecting the two polygons can be found in $O(n^2 \log n)$ time.

Proof. Using the above arguments one can establish that there are $O(n^2)$ candidate bridges of the types one-link, two-link and multi-link. Furthermore, as we show above these candidate brides can be computed in $O(n^2 \log n)$ total time. Therefore, an optimal geodesic bridge connecting the two simple polygons P and Q is one of these candidate bridges which has the minimal total weight.

Algorithm Geodesic-Bridge(P,Q), is omitted as it follows from the above discussion.

5 Concluding Remarks

We have presented the first polynomial time algorithm for finding an optimal geodesic bridge connecting two simple polygons. The time bound of our algorithm, $O(n^2 \log n)$, matches that for the Euclidean bridge problem [4], though the algorithms are structurally different.

We conjecture that it would be relatively easier to improve the time bound for the Euclidean version of the problem than the geodesic version. We leave open the challenging question of improving the $O(n^2 \log n)$ time bound. Another interesting problem would be to design $o(n^2)$ -time approximation algorithms for the geodesic bridge problem. Our algorithms also apply when there are a constant number of obstacles between the two polygons.

References

- H.-K. Ahn, O. Cheong, and C.-S. Shin. Building bridges between convex regions. *Computational Geom*etry: Theory and Applications, 25(1/2):161–170, 2003.
- [2] B. Aronov, S. Fortune, and G. Wilfong. The furthestsite geodesic Voronoi diagram. *Discrete Comput. Geom.*, 9:217–255, 1993.
- [3] B. Bhattacharya and R. Benkoczi. On computing the optimal bridge between two convex polygons. *Inform. Proc. Lett.*, 79(5):215–221, 2001.
- [4] A. M. Bhosle and T. F. Gonzalez. Exact and approximation algorithms for finding an optimal bridge connecting two simple polygons. *IJCGA*, 15(6):609–630, 2005.
- [5] L. Cai, Y. Xu, and B. Zhu. Computing the optimal bridge between two convex polygons. *Inform. Proc. Lett.*, 69:127–130, 1999.
- [6] J. Hershberger and S. Suri. An optimal algorithm for euclidean shortest paths in the plane. SIAM J. Comput., 28(6):2215–2256, 1999.
- [7] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry*, 18(4):377–383, 1997.
- [8] S. K. Kim and C. S. Shin. Computing the optimal bridge between two polygons. *Theory of Computing* Systems, 34(4):337–352, 2001.
- [9] X. Tan. On optimal bridges between two convex regions. *Inform. Proc. Lett.*, 76:163–168, 2000.
- [10] X. Tan. Finding an optimal bridge between two polygons. *IJCGA*, 12(3):249–262, 2002.
- [11] T. Tokuyama. Efficient algorithm for the minimum diameter bridge problem. LNCS, 2098:362–369, 2001.
- [12] D. P. Wang. An optimal algorithm for constructing an optimal bridge between two simple rectilinear polygons. *Inform. Proc. Lett.*, 79:229–236, 2001.
Approximating Geodesic Distances on 2-Manifolds in \mathbb{R}^3

Christian Scheffer*

Jan Vahrenhold[†]

Abstract

We present an algorithm for approximating geodesic distances on 2-manifolds in \mathbb{R}^3 . Our algorithm works on an ε -sample of the underlying manifold and computes approximate geodesic distances between all pairs of points in this sample. The approximation error is *multiplicative* and depends on the density of the sample. For an ε -sample *S*, the algorithm has a near-optimal running time of $\mathcal{O}(|S|^2 \log |S|)$, an optimal space requirement of $\mathcal{O}(|S|^2)$, and approximates the geodesic distances up to a factor of $1 - \mathcal{O}(\sqrt{\varepsilon})$ and $(1 - \mathcal{O}(\varepsilon))^{-1}$.

1 Introduction

The study of geodesic (paths and) distances on threedimensional objects has a long history both in Differential and Discrete Geometry, and a recent survey [4] summarizes the main results and the variety of applications, e.g., in GIS and Robotics. If the underlying object is differentiable, methods from Differential Geometry, e.g. special classes of partial differential, can be applied. If, on the other hand, the underlying object is non-differentiable, e.g. polyhedral objects, discretized versions of algorithms from Differential Geometry or discrete shortest-paths algorithm have to be used.

The problem setting we focus on can be seen as a hybrid between these two extremes: We study the problem of computing geodesic distances on 2-manifolds in \mathbb{R}^3 but assume that the input is a set of points sampled from the considered surface. The task then is to compute geodesic distances on the manifold between all pairs of points in the sample. Since the set of sample points can only approximate the manifold, the algorithm can only be expected to compute approximate geodesic distances. Our main contribution is to show that an approximation with an multiplicative approximation error that only depends on the quality of the point sample with respect to the manifold can be computed in near-optimal time.

1.1 Shortest Paths on Manifolds

Kimmel and Sethian [7] present the so-called *fast marching method* to compute geodesics on a discretized (explicitly given) manifold. For the case of computing distance functions and geodesics on an implicitly given manifold, we can resort to the theoretical framework of Mémoli and Sapiro [8]. Their theory is based upon continuous differential geometry methods, and a discrete version can be implemented using the fast marching method which results in a running time of $\mathcal{O}(n^2 \log n)$ for a discretized manifold consisting of n points. The algorithm approximates geodesic distances up to an additive error term that depends on the granularity of the discretized manifold (and thus on the number n of points), also, an upper bound on the local curvature of the manifold has to be known to the algorithm. While this latter restriction can be removed using methods we developed in a companion paper [10], the additive error seems to be inherent to any approach using the fast marching method.

1.2 Shortest Paths on Polyhedral Objects

In contrast to the algorithms described in the previous section, our approach is to first to approximate the manifold by a polyhedral object and then to compute (exact) geodesics on this approximation. As for the case of explicitly given manifolds, the fast marching method of Kimmel and Sethian can be used, see, e.g., the work by Mémoli and Sapiro [9], but the analysis shows that it still results in an additive approximation error. Efficient exact shortest path computations on general polyhedra are considered complex and challenging, and recent surveys [1, 4] conclude that the general problem is still wide open. The currently best known results related to shortest path computations on polyhedra are due to Chen and Han [5] and Schreiber [11]. Chen and Han [5] present an algorithm based on a *con*tinuous Dijkstra technique that, after $\mathcal{O}(n^2)$ preprocessing time, can answer distance queries to a fixed source in $\mathcal{O}(q \cdot \log n / \log q)$ time where $1 \le q \le n$ is a trade-off parameter. Since this source has to be known during preprocessing, using their algorithm as a building block in an all-pairs geodesics problem results in $\mathcal{O}(n^3)$ running time. The algorithm by Schreiber [11] solves the single-source shortest path problem in optimal $\mathcal{O}(n \log n)$ time but assumes that the underlying polyhedron belongs to one of three classes of polyhedra where the edge length of adjacent faces do not differ by more than a constant factor. In the general case we are considering, this assumption cannot be made.

^{*}Department of Computer Science, Technische Universität Dortmund, christian.scheffer@cs.tu-dortmund.de

[†]Department of Computer Science, Technische Universität Dortmund, jan.vahrenhold@cs.tu-dortmund.de.

Thus, one of the main contributions of this paper is to demonstrate that we can transform the polyhedral approximations of the manifolds we are working with such that we can both apply Schreiber's algorithm and at the same time maintain the asymptotic approximation quality of the resulting shortest paths. We embed this algorithm in an approximation context and show that computing approximate geodesic distances between all pairs of points can be done in $\mathcal{O}(n^2 \log n)$ time.

Due to the apparent difficulty of the exact shortest path problem on general polyhedra (possibly of genus $g \geq 1$), Aleksandrov *et al.* [1] focus on answering approximate shortest path queries. They present an $(1 \pm \delta)$ -approximation algorithm that, after $\mathcal{O}\left(\frac{(g+1)n^2}{\delta^{3/2}q}\log\frac{n}{\delta}\log^4\frac{1}{\delta}\right)$ preprocessing time, can answer a shortest-path query between an arbitrary pair of points in time $\mathcal{O}(q)$ (again, q is a trade-off parameter, this time chosen such that $\frac{1}{\sqrt{\delta}}\log^2\frac{1}{\delta} < q < \frac{(g+1)^{2/3}n^{1/2}}{\sqrt{\delta}}$). We come back to this algorithm in Section 4.

2 Description of the Algorithm

As discussed in the introduction, our approach is to work on a point set sampled from the considered surface. In a nutshell, we (re-)construct a polyhedral object having the sample points as its vertices that approximates the manifold and then use Schreiber's exact algorithm to compute geodesic distances between all pairs of points in the sample. As a consequence, the approximation quality of our algorithm only depends on the approximation quality of the point sample S with respect to the manifold Γ from which the points have been sampled. This quality can be characterized using a central concept introduced by Amenta and Bern [2] in the context of reconstructing smooth surfaces from point samples: For any point x on the manifold Γ , the *local feature size* lfs(x) is defined as the distance of x to the medial axis of Γ . Thus, the local feature size captures the curvature and the folding of Γ . It is known that the (topological) correctness of a reconstruction algorithm depends on the density of the set $S \subset \Gamma$ of sample points used for the reconstruction relative to the local feature size.

Definition 1 A discrete subset S a smooth 2-manifold $\Gamma \subset \mathbb{R}^3$ is an ε -sample of Γ if and only if for every point $x \in \Gamma$ there is a sample point $s \in S$ with $|xs| \leq \varepsilon \cdot lfs(x)$.

In the light of the above definition, we present an algorithm that takes an ε -sample S of a manifold Γ and computes geodesic distances on a specific polyhedron Π whose vertex set is derived from S. For any two points $s_1, s_2 \in S$, the geodesic distance on Π between s_1 and s_2 is within a multiplicative factor of the geodesic distance between these points when measured on (the unknown) manifold Γ ; the approximation error depends only on ε .

2.1 Outline of Our Approach

We start out by giving an algorithm for converting the input ε -sample S into what we call a *self-conforming* sample S^{conf} (see Definition 5). We then compute the *restricted Delaunay tetrahedrization* $\text{Del}_{|\Gamma}(S^{\text{conf}})$ (see Definition 3) and prove that $\text{Del}_{|\Gamma}(S^{\text{conf}})$ is a *self-conforming* polyhedron (see Definition 2), i.e., can be handled by Schreiber's algorithm in optimal time.

Definition 2 (Schreiber [11], p. 40) A polyhedron $P \subset \mathbb{R}^3$ (possibly non-convex) is self-conforming if for each edge e of ∂P , there is a connected region R(e), which is the union of $\mathcal{O}(1)$ facets of ∂P and whose interior contains e, so that the shortest path distance from e to any edge e' of $\partial R(e)$ is at least $2c \cdot \max\{|e|, |e'|\}$, where c is some positive constant.

We emphasize that it is sufficient for the algorithm (and its correctness proof) to know that for each edge e, some region R(e) with the desired properties exists; it is not necessary to have an explicit description of this region available. Thus, to prove that $\text{Del}_{|\Gamma}(S^{\text{conf}})$ is selfconforming (see Lemma 16), we only need to show that $\text{Del}_{|\Gamma}(S^{\text{conf}})$ has the following properties:

- The degree of each vertex of $\text{Del}_{|\Gamma}(S^{\text{conf}})$ is upperbounded by a constant.
- The minimum inner angle of each facet of $\mathrm{Del}_{|\Gamma}(S^{\mathrm{conf}})$ is lower-bounded by a constant.

The above is summarized in the following lemma:

Lemma 1 Let Γ be a smooth 2-manifold in \mathbb{R}^3 and let S be an ε -sample of Γ . We can decimate S such that the restricted Delaunay tetrahedrization $\text{Del}_{|\Gamma}(S^{\text{conf}})$ of the decimated set S^{conf} is a self-conforming polyhedron.

Finally, we show that the (exact) geodesic distances computed on $\operatorname{Del}_{|\Gamma}(S^{\operatorname{conf}})$ are within a multiplicative factor (depending on ε) of the geodesic distances on Γ .

2.2 Construction of a Self-Conforming Sample

Our algorithm computes a self-conforming subset of the input ε -sample and then constructs a restricted Delaunay tetrahedrization. To make the definition of a self-conforming sample more transparent, we first present the definition of a restricted Delaunay tetrahedrization:

Definition 3 (Funke and Ramos [6], p. 782) Let S be a set of points sampled from a manifold $\Gamma \in \mathbb{R}^3$.

- 1. The restricted Voronoi diagram $\operatorname{Vor}_{|\Gamma}(S)$ consists of cells $\operatorname{Vor}(s) \cap \Gamma$, $s \in S$.
- 2. The restricted Delaunay tetrahedrization $\text{Del}_{|\Gamma}(S)$ is the dual to $\text{Vor}_{|\Gamma}(S)$.

Since the manifold Γ is unknown to the algorithm, it is impossible to exactly compute the intersection of a Voronoi cell with the manifold. Amenta and Bern [2] and (building upon their work) Funke and Ramos [6] presented the following approach to approximating this intersection: For each point $s \in S$, compute its restricted Voronoi cell by intersecting the full-dimensional cell with an approximation of the plane tangent to Γ in s. Computing of all approximate tangent planes takes $\mathcal{O}(|S| \log |S|)$ time [6], but the approximation quality has been analyzed in an asymptotic sense only; this is mainly due to the fact that – to achieve a near-linear running time – several trade-offs need to be made.

In the situation of our algorithm, we need an exact bound on the approximation quality (to derive the bounds for vertex degrees and inner angles mentioned above). What we do *not* need, however, is a running time better than $\Theta(|S|^2 \log |S|)$, and thus we can use exact textbook algorithms for computing the Voronoi diagram and computing the intersection of the faces of a cell with the respective (approximation of the) tangent planes. With some technical effort, but using elementary trigonometry only, we then can prove the following non-asymptotic bound on the approximation quality:

Lemma 2 Let $\varepsilon \leq \frac{1}{22}$ be a constant and let s be an arbitrary sample point in an ε -sample of a 2-manifold Γ . Let x' be the furthest point (from s) in $Vor_{|\Gamma}(s)$, and let v be to the furthest vertex (from s) in the intersection of Vor(s) and the approximate tangent plane computed by the above algorithm. Then $|sx'| \leq 1.0005 \cdot |sv|$ holds.

Similarly, since neither Γ nor its medial axis are known, an exact computation of the local feature size is impossible. Instead, Funke and Ramos [6] discuss how to compute a (pointwise) lower bound for $\varepsilon \cdot lfs(\cdot)$. This lower bound is derived from the distance of a point $s \in S$ to the furthest vertex of its restricted Voronoi cell. Again, this bound is given in an asymptotic sense only. To be able to prove the above-mentioned bounds on the degree of each vertex in $\text{Del}_{|\Gamma}(S^{\text{conf}})$ and on the minimum inner angle of each facet of $\text{Del}_{|\Gamma}(S^{\text{conf}})$, we need to know the constants hidden in the Big-Oh notation. In a companion paper [10], we prove the following result:

Theorem 3 For an ε -sample S of a 2-manifold Γ in \mathbb{R}^3 we can compute, in quadratic time, a function $\phi(s)$ for each $s \in S$ as the distance of s to the furthest Voronoi vertex v of the intersection of the approximation of the plane tangent to Γ in s and Vor(s). For this function holds that $\phi(s) \leq 1.135 \cdot \frac{\varepsilon}{1-\varepsilon} \cdot lfs(s)$.

If we define $\phi'(s) := 1.0005 \cdot \phi(s)$ and assume that $\varepsilon \leq \frac{1}{22}$ holds, we have the following result:

Corollary 4 Let s be a point in an ε -sample S of a 2manifold Γ in \mathbb{R}^3 . Then, $\phi'(s) \leq 1.0005 \cdot 1.135 \cdot \frac{\varepsilon}{1-\varepsilon}$. $lfs(s) < 1.19 \cdot \varepsilon \cdot lfs(s)$ holds. Furthermore, $Vor_{|\Gamma}(s)$ is contained in $B_{\phi'(s)}(s)$, i.e. the ball centered at s with radius $\phi'(s)$.

2.2.1 Definition of a Self-Conforming Point Sample

We are now almost ready to define what constitutes a self-conforming point sample. One property of the local feature size that is crucial for most proofs is that the local feature size is a 1-Lipschitz function [2]:

Definition 4 For $\alpha \in \mathbb{R}^+$, a non-negative, real-valued function f is α -Lipschitz if $f(x) \leq f(x') + \alpha \cdot |xx'|$ for all $x, x' \in dom(f)$.

Incorporating this property into the requirement that the point sample should be locally not too dense yields the following definition:

Definition 5 A subset $S' \subset S$ is self-conforming, if there is a control function $f : S \to \mathbb{R}^+$ and constants $0 < \alpha, \beta < 1$ such that the following holds:

- 1. The function f is α -Lipschitz.
- 2. For each $s \in S'$, we have $B_{\beta \cdot f(s)}(s) \cap S' = \{s\}$.

We note that a self-conforming point sample is defined similarly to a *locally uniform* point sample (see Funke and Ramos [6]). The latter definition requires more properties for the point set but allows for an *approximate* control function. Since our proofs require a subset of the properties of a locally uniform point sample and since we can afford to compute an exact control function, we consider it instructive to (introduce and) use this new definition.

2.2.2 Construction of a Lipschitz Control Function

As noted by Funke and Ramos [6, p. 785], a "natural" way to make a function α -Lipschitz is to encode the maximum distance to any other point in the domain. Using the constants derived in the previous paragraph, we define the function ψ as follows:

$$\psi(s): S \to \mathbb{R}, s \mapsto \max_{s' \in S} \left\{ \phi'(s') - 1.19 \cdot \frac{1}{22} \cdot |ss'| \right\}$$
(1)

We note that the domain of ψ could be extended to Γ in the following way: For each Voronoi cell Vor(s), $s \in S$, and each point $x \in \text{Vor}_{|\Gamma}(s)$, define $\psi(x) := \psi(s)$ (breaking ties for the boundary of $\text{Vor}_{|\Gamma}(s)$ arbitrarily). Doing so allows us to prove that $2.056 \cdot \psi$ is an approximate control function in the sense of Funke and Ramos' definition (*local unifomity*), but increases all constants in the following by a factor of roughly two.

Lemma 5 The function ψ is a $\frac{1}{18}$ -Lipschitz function.

2.2.3 Decimation of the ε -Sample

To derive a sample S^{conf} that leads to a self-conforming polyhedron, we use the "Decimation Step" algorithm from Funke and Ramos [6, p. 789] (Algorithm 1). In contrast to their setting, we can afford to spend quadratic time during preprocessing, and thus this step can be implemented straightforward, i.e. without the need for approximate range-reporting structures.

Algorithm 1 Coarsening an input ε -sample [6].1: function DECIMATE(Points S, Function ψ)2: $S^{\text{conf}} = \emptyset, S^{\text{dense}} = S.$ 3: while $S^{\text{dense}} \neq \emptyset$ do4: Let s be an arbitrary point in $S^{\text{dense}}.$ 5: $S^{\text{conf}} = S^{\text{conf}} \cup \{s\}.$ 6: $S^{\text{dense}} = S^{\text{dense}} \setminus (S^{\text{dense}} \cap B_{\psi(s)}(s)).$ 7: Return $S^{\text{conf}}.$

This approach allows to prove the following lemma:

Lemma 6 For $s \in S^{conf}$, $B_{\frac{17}{16} \cdot \psi(s)}(s) \cap S^{conf} = \{s\}$.

Lemmas 5 and 6 together imply that ψ is a control function in the sense of Definition 5:

Corollary 7 For an ε -sample S of a 2-manifold Γ , we can derive a self-conforming sample S^{conf} with control function ψ and $\alpha = \frac{1}{18}$, $\beta = \frac{17}{18}$ in $\mathcal{O}(|S|^2)$ time.

2.3 Construction of $\text{Del}_{|\Gamma}(S_{ssrc}^{\text{conf}})$

We first observe that no point $x \in \Gamma$ is "too far" away from a point s of the self-conforming sample S^{conf} .

Lemma 8 For each $x \in \Gamma$, there is some $s \in S^{conf}$ such that $|xs| \leq 2.056 \cdot \psi(s)$.

Using this property, we can prove that in a restricted Delaunay tetrahedrization of a self-conforming sample the edge lengths of a single face are bounded relative to each other and that all faces have a minimum inner angle lower-bounded by a constant. First, we prove:

Lemma 9 For each edge $\overline{s_1s_2}$ in $\text{Del}_{|\Gamma}(S^{conf})$, $|s_1s_2|$ is upper-bounded by $4.112 \cdot \min\{\psi(s_1), \psi(s_2)\}$.

For the estimation of the approximation quality of our algorithm, we will also need to relate the length of the Delaunay edges to the local feature size.

Lemma 10 For each point s in the input ε -sample S, $\psi(s) \leq 1.19 \cdot \varepsilon \cdot lfs(s)$.

Since ψ ist fixed before the decimation, Lemma 10 also holds for each $s \in S^{\text{conf}}$.

Corollary 11 For each edge $\overline{s_1 s_2}$ in $\text{Del}_{|\Gamma}(S^{conf})$, $|s_1 s_2|$ is upper-bounded by $5 \cdot \varepsilon \cdot \min\{lfs(s_1), lfs(s_2)\}$.

An immediate implication is that the degree of each vertex in $\operatorname{Del}_{|\Gamma}(S^{\operatorname{conf}})$ is bounded by a constant.

Lemma 12 The degree of each vertex in $\text{Del}_{|\Gamma}(S^{conf})$ is bounded by 3925.

If, for each point $s \in S^{\text{conf}}$, we have its 3925 nearest neighbors at hand, Lemma 12 ensures that we can compute $\operatorname{Vor}_{|\Gamma}(S^{\text{conf}})$ and thus $\operatorname{Del}_{|\Gamma}(S^{\text{conf}})$ in $\mathcal{O}(|S| \log |S|)$ time (the first step of the algorithm provides us with the approximations of the planes tangent to Γ needed for restricting the Voronoi diagram).

In comparison, Funke and Ramos [6] also prove that a *locally uniform* sample admits a constant-degree tetrahedrization. Again, however, the bound is only given asymptotically, i.e. depending on $1/\varepsilon$. Since our algorithm needs an *upper* bound on the number of Voronoi neighbors (to ensure that no relevant neighbor is missed), the *upper* bound on ε , i.e. the *lower* bound on $1/\varepsilon$ is not sufficient for our purpose.

2.4 Intermediate Summary

As mentioned in Section 2.1, our approach is to compute a self-conforming polyhedron such that we can use Schreiber's algorithm to efficiently compute shortestpath maps with respect to each point in the input sample. The above description indicates that a crucial step for doing this is to decimate the input sample. On the other hand, when computing a shortest path map with respect to some point $s_{\rm src}$, this point needs to be present in the decimated sample. Since the decimation step is computationally expensive, we cannot afford to repeat it a linear number of times. Thus, we decimate the sample once and then, in each iteration, ensure that the point $s_{\rm src}$ with respect to which the shortest-path map is computed is present in the set for which the restricted Delaunay tetrahedrization is computed.

It should be pointed out that the set $S_{s_{res}}^{\text{conf}} := S^{\text{conf}} \cup$ $\{s_{\rm src}\}$ for which the restricted Delaunay tetrahedrization is computed may no longer be a self-conforming set if $s_{\rm src} \notin S^{\rm conf}$, since $s_{\rm src}$ may violate Property (2) of Definition 5. Except for Lemma 12, however, no Lemma in the previous section relies on this property, so all other Lemmas are still valid. For Lemma 12, we observe that the introduction of $s_{\rm src}$ may increase each other vertex's degree by one (thus, the constant needs to be adjusted accordingly). A close look at the proof of Lemma 12, however, shows that Property (2) of Definition 5 is *not* needed for the point s whose degree is to be bounded but only for the points inside $B_{4.112 \cdot \psi(s)}(s)$. Thus, the statement of Lemma 12 also holds for the point $s_{\rm src}$. For ease of exposition we thus assume that the degree of all vertices in $S_{s_{\rm src}}^{\rm conf}$ is bounded by 3925. Also, we assume that the computation of $\operatorname{Del}_{|\Gamma}(s)$ for each $s \in S_{s_{\operatorname{src}}}^{\operatorname{conf}}$ takes $s_{\rm src}$ into account when computing ${\rm Vor}_{|\Gamma}(s)$ based upon its nearest neighbors. This results in Algorithm 2.

Algo	rithm 2 Approximating geodesic distances between points of an ε -sat	mple S .
1: f t	inction ApproximateGeodesicDistances(Points S)	
2:	Let Distance be an $ S \times S $ -matrix, each entry of which is initialized	ed with $+\infty$.
3:	Compute $\phi(s)$ and tangent plane for each $s \in S$. \triangleright Use the alg	gorithm by Scheffer and Vahrenhold [10].
4:	Compute ψ from ϕ (Lipschitziation).	\triangleright See Section 2.2.2.
5:	$S^{\operatorname{conf}} = \operatorname{Decimate}(S, \psi).$	\triangleright Use Algorithm 1.
6:	Compute for each point in S^{conf} its 3925 nearest neighbors.	\triangleright Use the brute-force algorithm.
7:	Compute for each point in $S \setminus S^{\text{conf}}$ its nearest neighbor in S^{conf} .	\triangleright Use the brute-force algorithm.
8:	for each $s_{\rm src} \in S$ do	
9:	$S_{s_{ m src}}^{ m conf} := S^{ m conf} \cup \{s_{ m src}\}.$	
10:	Compute $\operatorname{Del}_{ \Gamma}(S^{\operatorname{conf}}_{s_{\operatorname{src}}})$.	\triangleright See Section 2.3.
11:	$SPMap = SCHREIBERSPREPROCESSING(Del_{ \Gamma}(S_{s_{src}}^{conf}), s_{src}).$	\triangleright Use the algorithm by Schreiber [11].
12:	for each $s \in S$ do	
13:	$\mathbf{if} s \in S^{\mathrm{conf}}_{s_{\mathrm{src}}} \mathbf{then}$	
14:	$Distance[s_{\rm src}, s] = SCHREIBERSQUERY(SPMap, s).$	\triangleright Use the algorithm by Schreiber [11].
15:	else	
16:	Let s' be s's nearest neighbor in $S_{s_{\rm src}}^{\rm conf}$.	
17:	for each face f of the $\mathcal{O}(1)$ faces adjacent to s' do	
18:	Let s'' be the point on f closest to s .	
19:	$Distance[s_{\rm src}, s] = \min\{Distance[s_{\rm src}, s], SCHREIBERSG\}$	$UERY(SPMap, s'')\}.$
		\triangleright Use the algorithm by Schreiber [11].

20: Return Distance.



Figure 1: Region $R^{\text{strip}}(e)$ for an edge $e \in \text{Del}_{|\Gamma}(S_{src}^{\text{conf}})$.

Lemma 13 Assuming that $\text{Del}_{|\Gamma}(S_{ssc}^{conf})$ is self-conforming, Algorithm 2 runs in $\mathcal{O}(|S|^2 \log |S|)$ time.

2.5 Properties of $\text{Del}_{|\Gamma}(S_{servent}^{\text{conf}})$

To show that $\operatorname{Del}_{|\Gamma}(S_{s_{\operatorname{src}}}^{\operatorname{conf}})$ is self-conforming in accordance to Definition 2, we need to show that for each edge e of a facet of $\operatorname{Del}_{|\Gamma}(S_{s_{\operatorname{src}}}^{\operatorname{conf}})$, there is a constant-size region $R^{\operatorname{strip}}(e)$ containing e so that the shortest-path distance from e to any edge e' of $\partial R^{\operatorname{strip}}(e)$ is at least $2c \cdot \max\{|e|, |e'|\}$, where c is a positive constant.

Intuitively, we define $R^{\text{strip}}(e)$ as a (constant-size, yet not necessary minimal) triangle patch surrounding esuch that if s_{src} lies inside $R^{\text{strip}}(e)$, it is neither a vertex of $\partial R^{\text{strip}}(e)$ nor adjacent to a vertex of $\partial R^{\text{strip}}(e)$.

Definition 6 Let e be an edge of a facet of $\operatorname{Del}_{|\Gamma}(S_{ssrc}^{conf})$ and define R(e) as the set of facets adjacent to e. If $s_{src} \notin R(e)$, define $R^{strip}(e) := R(e)$ —see Figure 1(a). Otherwise, depending on whether $s_{src} \in e$ or $s_{src} \in$ $R(e) \setminus \{e\}$, define $R^{strip}(e) := R(R(e))$ (see Figure 1(b)) or $R^{strip}(e) := R(R(R(e)))$ (see Figure 1(c)), where R(R(e)) consists of R(e) and all facets adjacent to R(e)and where R(R(R(e))) is defined analogously. By Lemma 12, each vertex of $R^{\text{strip}}(e)$ has constant degree, and thus $R^{\text{strip}}(e)$ consists of $\mathcal{O}(1)$ facets. To bound the shortest-path distance in $R^{\text{strip}}(e)$, we need the following technical lemma that relates the edge lengths on the boundary of $\partial R^{\text{strip}}(e)$ and the boundary of the "next inner" strip, i.e., $\partial R(R(e))$, $\partial R(e)$, or e.

Lemma 14 Let Δ be a facet of $\operatorname{Del}_{|\Gamma}(S_{s_{s_{rc}}}^{conf})$ such that $s_{s_{rc}}$ is no vertex of Δ . Then, for any two edges e, e' of Δ , we have $\frac{1}{6} \leq \frac{|e|}{|e'|} \leq 6$ and for any vertex s and any edge e of Δ , we have $\frac{2}{6} \cdot |e| \leq 2.056 \cdot \psi(s) \leq \frac{6}{2} \cdot |e|$.

Using Lemma 14 and elementary trigonometry, we can prove the following:

Lemma 15 Let Δ be a facet of $\text{Del}_{|\Gamma}(S_{s_{s_{rc}}}^{conf})$ such that $s_{s_{rc}}$ is no vertex of Δ . Then, each angle of Δ is of size at least $\pi/20$.

The constant factor of 4.112 in the upper bound for the edge length in $\text{Del}_{|\Gamma}(S^{\text{conf}})$ (Lemma 9), the "edgelength" ratio of 1:6 (Lemma 14), and the minimum angle of $\pi/20$ (Lemma 15) allow to determine the constant cneeded in the definition of a self-conforming polyhedron.

Lemma 16 Let e be an edge of a facet of $\operatorname{Del}_{|\Gamma}(S_{ssc}^{conf})$. Then the shortest path distance from e to any edge e' of $\partial R^{strip}(e)$ is at least $c \cdot \max\{|e|, |e'|\}$ for $c := \left(1 - \frac{1}{18} \cdot 4.112 \cdot \left(2 + \frac{1}{18} \cdot 4.112 + \left(1 + \frac{1}{18} \cdot 4.112\right)^2\right)\right)$. $\sin\left(\frac{\pi}{20}\right) \cdot \frac{1}{6^3}$.

Corollary 17 $\text{Del}_{|\Gamma}(S^{conf}_{s_{src}})$ is self-conforming.

Corollary 18 Algorithm 2 takes $\mathcal{O}(|S|^2 \log |S|)$ time.

3 Analysis of the Approximation Quality

We fix two points s_1, s_2 in the original ε -sample S and use $L_{\text{Del}} = L_{\text{Del}}(s_1, s_2)$ to denote their geodesic distance as returned by Algorithm 2 and $L_{\Gamma} = L_{\Gamma}(s_1, s_2)$ to denote their (unknown) geodesic distance on Γ .

The following lemma needed to bound the approximation quality is derived from Bernstein *et al.* [3, Cor. 4].

Lemma 19 Let x_1 and x_2 be two arbitrary points on Γ with $|x_1x_2| \leq \sqrt{\varepsilon^{-1}} \cdot \max \{ lfs(x_1), lfs(x_2) \}$. Then we have $L_{Del}(x_1, x_2) \geq (1 - \mathcal{O}(\epsilon)) \cdot L_{\Gamma}(x_1, x_2)$.

For the lower bound for L_{Del} relative to L_{Γ} we iteratively partition the geodesic path on $\text{Del}_{|\Gamma}(S_{s_{\text{src}}}^{\text{conf}})$ between s_1 and s_2 into subpaths whose endpoints fulfill the assumptions of Lemma 19. With rather technical computations synchronized over all subpaths we obtain the following lemma.

Lemma 20
$$\left(1 - \mathcal{O}\left(\varepsilon^{\frac{1}{2}}\right)\right) \cdot L_{\Gamma} \leq L_{Del}.$$

For the upper bound for L_{Del} we first establish a bijection between the facets of $\text{Del}_{|\Gamma}(S_{s_{\text{src}}}^{\text{conf}})$ and patches on Γ . We then replace each subpath of the (unknown) geodesic path on Γ between s_1 and s_2 that crosses a patch in Γ by a path along the boundary of the corresponding facet of $\text{Del}_{|\Gamma}(S_{s_{\text{src}}}^{\text{conf}})$. Exploiting the lower bound on the minimum inner angle of the facets (Lemma 15) and the approximation quality of the facet w.r.t. the patch on Γ , we obtain the following result.

Lemma 21 $(1 - \mathcal{O}(\varepsilon))^{-1} \cdot L_{\Gamma} \geq L_{Del}.$

4 Using Aleksandrov et al.'s Algorithm

The algorithm of Aleksandrov *et al.* [1] is a $(1 \pm \delta)$ -approximation algorithm for all-pair shortest distance queries which could also be applied instead of repeatedly using Schreiber's (involved) algorithm. Lemma 22 shows that this leads to either the same running time and strictly worse approximation quality or to the same approximation quality and strictly worse running time.

Since an ε -sample S can be augmented by arbitrarily many points while still remaining an ε -sample, we need to restrict ourselves to *tight* ε -samples when comparing the dependence between |S|, the running time, and the approximation quality for the two algorithms. A tight ε -sample is an ε -sample S for which there is a positive constant η such that $B_{\eta \cdot \varepsilon \cdot lfs(s)}(s) \cap S = \{s\}$ for $s \in S$.

Lemma 22 Let S be a tight ε -sample of a 2-manifold Γ and assume that we use the Algorithm of Alexandrov et al. as part of Algorithm 2 to compute approximate shortest paths on $\text{Del}_{|\Gamma}(S_{sam}^{conf})$.

1. If the running time of Algorithm 2 is to remain in $\mathcal{O}(|S|^2 \log |S|)$, L_{Γ} and L_{Del} relate as follows:

(a)
$$\left(1 - \Omega\left(\varepsilon^{0.427}\right)\right) \cdot L_{\Gamma} \leq L_{Del}.$$

(b) $\left(1 - \mathcal{O}\left(\varepsilon^{\frac{1}{1.08}}\right)\right)^{-1} \cdot L_{\Gamma} \geq L_{Del}.$

2. If the approximation quality of Algorithm 2 is to remain as given in Lemmas 20 and 21, a tight sample of a higher density than S is required. The size of this sample implies a running time of $\omega\left(|S|^{\frac{19}{8}}\right)$.

References

- [1] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. *Discrete & Computational Geometry*, 44:762–801, 2010.
- [2] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. Discrete & Computational Geometry, 22(4):481–504, Dec. 1999.
- [3] M. Bernstein, V. de Silva, J. C. Langford, and J. B. Tenenbaum. Graph approximations to geodesics on embedded manifolds. Unpublished, http:// isomap.stanford.edu/BdSLT.pdf, Dec. 2000.
- [4] P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer. A survey of geodesic paths on 3D surfaces. Submitted, a technical report version is available from http://arxiv.org/abs/0904.2550, Aug. 2009.
- [5] J. Chen and Y. Han. Shortest paths on a polyhedron. International Journal of Computational Geometry and Applications, 6:127–144, 1996.
- [6] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. Thirteenth* Symp. on Discrete Algorithms, pp. 781–790, 2002
- [7] R. Kimmel and J. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Science*, 95(15):8431–8435, July 1998.
- [8] F. Mémoli and G. Sapiro. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. *Journal of Computational Physics*, 173(2):730–764, 2001.
- [9] F. Mémoli and G. Sapiro. Distance functions and geodesics on submanifolds of ℝ^d and point clouds. SIAM Journal of Applied Mathematics, 65(4):1227–1260, 2005.
- [10] C. Scheffer and J. Vahrenhold. Learning a 2manifold with a boundary in ℝ³. In M. Hoffmann, editor, Proceedings of the 27th European Workshop on Computational Geometry, pp. 213–216, 2011.
- [11] Y. Schreiber. An optimal-time algorithm for shortest paths on realistic polyhedra. *Discrete & Computational Geometry*, 43:21–53, 2010.

An In-Place Priority Search Tree*

Minati De^{\dagger}

Anil Maheshwari[‡]

Subhas C. Nandy[†]

Michiel Smid[‡]

Abstract

One of the classic data structures for storing point sets in \mathbb{R}^2 is the priority search tree, introduced by Mc-Creight in 1985. We show that this data structure can be made in-place, i.e., it can be stored in an array such that each entry only stores one point of the point set. We show that the standard query operations can be answered within the same time bounds as for the original priority search tree, while using only O(1) extra space.

1 Introduction

Let P be a set of n points in \mathbb{R}^2 . A priority search tree, as introduced by McCreight [2], is a binary tree T with exactly one node for each point of P and that has the following two properties:

- For each non-root node u, the point stored at u has a smaller y-coordinate than the y-coordinate stored at the parent of u.
- For each internal node u, all points in the left subtree of u have an x-coordinate which is less than the x-coordinate of any point in the right subtree of u.

The first property implies that T is a max-heap on the ycoordinates of the points in P. The second property implies that T is a binary search tree on the x-coordinates of the points in P, except that there is no relation between the x-coordinates of the points stored at u and any of its children.

In order to use T as a binary search tree on the xcoordinates, McCreight stored at each internal node uone additional point p_u of P, viz., the point in the left subtree of u whose x-coordinate is maximum. Thus, the data structure uses O(n) space and, by taking for T a balanced tree, several types of range queries can be answered efficiently:

• HIGHESTNE (x_0, y_0) : report the highest point of P in the north-east quadrant of the query point (x_0, y_0) .

- LEFTMOSTNE (x_0, y_0) : report the leftmost point of P in the north-east quadrant of the query point (x_0, y_0) .
- HIGHEST3SIDED (x_0, x_1, y_0) ; report the highest point of P in the 3-sided query range $[x_0, x_1] \times [y_0, \infty)$.
- ENUMERATE3SIDED (x_0, x_1, y_0) ; report all points of P in the 3-sided query range $[x_0, x_1] \times [y_0, \infty)$.

The first three queries can be answered in $O(\log n)$ time, whereas the fourth query takes $O(\log n+m)$ time, where m is the number of points of P that are in the query range.

In this paper, we show that these results can also be obtained without storing the "splitting" points p_u at the internal nodes of the tree. Thus, any node of the tree stores exactly one point of P and, as a result, we obtain an *in-place* implementation of the priority search tree: We take for T a binary tree of height $h = \lfloor \log n \rfloor$, such that the levels¹ $0, 1, \ldots, h - 1$ are full and level hconsists of $n - (2^h - 1)$ nodes which are aligned as far as possible to the left. This allows us to store the tree, like in a standard heap, in an array $P[1 \ldots n]$; the root is stored at P[1], its left and right children in P[2] and P[3], etc.

In the rest of this paper, we will present algorithms for constructing the in-place priority search tree and answering the above queries. Each of these algorithms uses, besides the array $P[1 \dots n]$, only O(1) extra space, in the sense that a constant number of variables are used, each one being an integer of $O(\log n)$ bits. The main result of this paper is the following:

Theorem 1 Let P be a set of n points in \mathbb{R}^2 .

- 1. The in-place priority search tree can be constructed in $O(n \log n)$ time using O(1) extra space.
- 2. Each of the queries HIGHESTNE, LEFTMOSTNE, and HIGHEST3SIDED can be answered in $O(\log n)$ time using O(1) extra space.
- 3. The query ENUMERATE3SIDED can be answered in $O(\log n + m)$ time using O(1) extra space, where m is the number of points of P that are in the query range.

^{*}Research supported by NSERC and the Commonwealth Scholarship Program of DFAIT.

[†]Indian Statistical Institute, Kolkata, India. Part of this work was done while M.D. was visiting Carleton University, Ottawa, Canada. minati.isi@gmail.com

 $^{^{\}ddagger}\mbox{School}$ of Computer Science, Carleton University, Ottawa, Canada.

¹The root is at level 0.

For ease of presentation, we assume that no two points in the set P have the same x-coordinates and no two points in P have the same y-coordinates. The x- and y-coordinates of a point p in \mathbb{R}^2 will be denoted by x(p) and y(p), respectively.

2 Constructing the in-place priority search tree

Let $h = \lfloor \log n \rfloor$ be the height of the priority search tree. Our algorithm constructs the tree level by level and maintains the following invariant:

• The subarray $P[1 \dots 2^i - 1]$ stores levels $0, 1, \dots, i - 1$ of the tree, and the points in the subarray $P[2^i \dots n]$ are sorted by their *x*-coordinates.

Algorithm 1: CONSTRUCTPST

Input: An array $P[1 \dots n]$ of points in \mathbb{R}^2 . **Output**: The priority search tree of those points stored in P. 1 $h = |\log n|; A = n - (2^{h} - 1);$ **2** HEAPSORT(1, n); for i = 0 to h - 1 do 3 $k = |A/2^{h-i}|;$ 4 $K_1 = 2^{h+1-i} - 1;$ $K_2 = 2^{h-i} - 1 + A - k2^{h-i};$ 5 6 $K_3 = 2^{h-i} - 1;$ 7 for j = 1 to k do 8 $\ell = \text{index in}$ 9 $\{2^i + (j-1)K_1, \dots, 2^i + jK_1 - 1\}$ such that $y(P[\ell])$ is maximum; swap $P[\ell]$ and $P[2^i + j - 1]$; $\mathbf{10}$ if $k < 2^i$ then 11 $\ell = \text{index in}$ $\mathbf{12}$ $\{2^i + kK_1, \dots, 2^i + kK_1 + K_2 - 1\}$ such that $y(P[\ell])$ is maximum; swap $P[\ell]$ and $P[2^i + k]$; $\mathbf{13}$ $m = 2^i + kK_1 + K_2;$ 14 for j = 1 to $2^i - k - 1$ do 15 $\ell = \text{index in}$ 16 $\{m + (j-1)K_3, \dots, m + jK_3 - 1\}$ such that $y(P[\ell])$ is maximum; swap $P[\ell]$ and $P[2^i + k + j];$ $\mathbf{17}$ HEAPSORT $(2^{i+1}, n);$ $\mathbf{18}$

The algorithm starts by sorting the array P[1...n] by x-coordinates. After this sorting step, the invariant holds with i = 0.

Let *i* be an index with $0 \leq i < h$, and consider the *i*-th step of the algorithm. Let $A = n - (2^h - 1)$ be the number of nodes at level *h* of the tree, and let $k = \lfloor A/2^{h-i} \rfloor$. Level *i* consists of 2^i nodes. If $k = 2^i$, then each of these nodes is the root of a subtree of size $2^{h+1-i} - 1$. Otherwise, we have $k < 2^i$, in which case level *i* consists of, from left to right,

- 1. k nodes, which are roots of subtrees, each of size $K_1 = 2^{h+1-i} 1$,
- 2. one node, which is the root of a subtree of size $K_2 = 2^{h-i} 1 + A k2^{h-i}$,
- 3. $2^{i} 1 k$ nodes, which are roots of subtrees, each of size $K_{3} = 2^{h-i} 1$.

We divide the subarray $P[2^i \dots n]$ into 2^i blocks: If $k = 2^i$, then there are k blocks of size $2^{h+1-i} - 1$. Otherwise, there are, from left to right, (i) k blocks of size K_1 , (ii) one block of size K_2 , and (iii) $2^i - 1 - k$ blocks of size K_3 .

The algorithm scans the subarray $P[2^i \dots n]$ and in each of the 2^i blocks, finds the highest point. These highest points are swapped with the subarray $P[2^i \dots 2^{i+1} - 1]$. At this moment, level *i* of the tree has been constructed, but the elements in the subarray $P[2^{i+1} \dots n]$ may not be sorted by their *x*-coordinates. Therefore, the algorithm runs the heapsort algorithm on this subarray.

The complete algorithm for constructing the in-place priority search tree is given in Algorithm 1. It uses algorithm HEAPSORT(m, n), which runs the heapsort algorithm on the subarray $P[m \dots n]$.

The correctness of this algorithm follows by observing that the invariant is correctly maintained. The initial sorting in line 2 takes $O(n \log n)$ time using O(1) extra space. Each of the $h = \lfloor \log n \rfloor$ iterations of the main for-loop takes $O(n \log n)$ time and O(1) extra space. We can use one extra variable to maintain the value 2^i , so that it does not have to be recomputed during the forloop. Thus, the entire algorithm CONSTRUCTPST takes $O(n \log^2 n)$ time and uses O(1) extra space.²

3 Queries on the in-place priority search tree

In this section, we present the algorithms for the query problems mentioned in Section 1. For ease of presentation, we describe the algorithms using the terminology of trees. We will denote by T the priority search tree that is implicitly defined by the array $P[1 \dots n]$ that results by running algorithm CONSTRUCTPST. Recall that the root of T, denoted by root(T), is stored at P[1]. Consider a node whose index in P is i. If $2i \leq n$, then this node has a left child, which is stored at P[2i]. If $2i + 1 \leq n$, then this node has a right child, which is stored at P[2i + 1]. This node is a leaf if and only if 2i > n. We will identify each node in T with the point of P stored at that node. For any p in P, we denote by T_p the subtree rooted at p. Furthermore, the left and

²Using the in-place algorithm of Katajainen and Pasanen [1] that stably sorts a sequence of n bits in O(n) time, the running time can be improved to $O(n \log n)$ with O(1) extra space. The details will be given in the full paper.

right children of p (if they exist) are denoted by p_l and p_r , respectively.

3.1 HIGHESTNE (x_0, y_0)

For two given real numbers x_0 and y_0 , let $Q = [x_0, \infty) \times [y_0, \infty)$ be the north-east quadrant of the point (x_0, y_0) . If $Q \cap P \neq \emptyset$, define p^* to be the highest point of P in Q. If $Q \cap P = \emptyset$, define p^* to be the point $(\infty, -\infty)$. Algorithm HIGHESTNE (x_0, y_0) will return the point p^* .

The algorithm uses two variables best and p, which satisfy the following invariant:

- If $Q \cap P \neq \emptyset$, then $p^* \in \{best\} \cup T_p$.
- If $Q \cap P = \emptyset$, then $p^* = best$.

The algorithm initializes $best = (\infty, -\infty)$ and p = root(T). During the algorithm, p moves down the tree according to the relative positions of p, its children, and the quadrant Q. The algorithm is given in Algorithm 2. It uses the procedure UPDATEHIGHEST(t), which takes as input a point t and does the following: If $t \in Q$ and y(t) > y(best) then it assigns best = t.

Algorithm 2: HIGHESTNE (x_0, y_0) **Input**: Real numbers x_0 and y_0 defining the north-east quadrant Q. **Output**: The highest point p^* in $Q \cap P$, if it exists; otherwise the point $(\infty, -\infty)$. 1 best = $(\infty, -\infty)$; p = root(T); 2 while p is not a leaf do if $p \in Q$ then 3 UPDATEHIGHEST $(p); p = p_l;$ $\mathbf{4}$ else if $y(p) < y_0$ then $\mathbf{5}$ 6 $p = p_l$: else if p has one child then $\mathbf{7}$ 8 $p=p_l;$ else if $x(p_r) \leq x_0$ then 9 10 $p = p_r;$ else if $x(p_l) \ge x_0$ then 11 p =higher among p_l and p_r ; 12else if $y(p_r) < y_0$ then 13 14 $p = p_l;$ 15else UPDATEHIGHEST $(p_r); p = p_l;$ 16 **17** UPDATEHIGHEST(p): 18 return *best*;

The correctness of this algorithm follows from the fact that the invariant is correctly maintained. Since in each iteration, p moves down the tree, the while-loop makes $O(\log n)$ iterations, each one taking O(1) time. Thus, the total time for algorithm HIGHESTNE is $O(\log n)$. It follows from the algorithm that it uses O(1) extra space.



Figure 1: Two cases for LEFTMOSTNE (x_0, y_0) .

3.2 LEFTMOSTNE (x_0, y_0)

As before, let $Q = [x_0, \infty) \times [y_0, \infty)$ be the north-east quadrant of the point (x_0, y_0) . If $Q \cap P \neq \emptyset$, define p^* to be the leftmost point of P in Q. If $Q \cap P = \emptyset$, define p^* to be the point (∞, ∞) . Algorithm LEFTMOSTNE (x_0, y_0) will return the point p^* .

The algorithm uses three variables best, p, and q, which satisfy the following invariant:

- If $Q \cap P \neq \emptyset$, then $p^* \in \{best\} \cup T_p \cup T_q$.
- If $Q \cap P = \emptyset$, then $p^* = best$.
- p and q are at the same level of T and $x(p) \le x(q)$.

The algorithm starts by initializing $best = (\infty, \infty)$, p = root(T), and q = root(T). During the algorithm, p and q move down the tree according to the relative positions of their children and the quadrant Q. The algorithm is given in Algorithm 3. It uses the procedure UPDATELEFTMOST(t), which takes as input a point t and does the following: If $t \in Q$ and x(t) < x(best) then it assigns best = t.

It follows by a careful case analysis that the invariant is correctly maintained, implying the correctness of the algorithm. In each iteration of the while-loop, p and q move down the tree, except in line 12. In the latter case, however, p will become a leaf in the next iteration. As a result, the while-loop makes $O(\log n)$ iterations. Since each iteration takes O(1) time, the total time for algorithm LEFTMOSTNE is $O(\log n)$. It follows from the algorithm that it uses O(1) extra space.

3.3 HIGHEST3SIDED (x_0, x_1, y_0)

The three real numbers x_0, x_1 , and y_0 define the threesided range $Q = [x_0, x_1] \times [y_0, \infty)$. If $Q \cap P \neq \emptyset$, define p^* to be the highest point of P in Q. If $Q \cap P = \emptyset$, define p^* to be the point $(\infty, -\infty)$. Algorithm HIGHEST3SIDED (x_0, x_1, y_0) returns the point p^* .

The algorithm uses two bits L and R, and three variables *best*, p, and q. As before, *best* stores the highest point in Q found so far. The bit L indicates whether or not p^* may be in the subtree of p; if L = 1, then p is to the left of Q. Similarly, the bit R indicates whether or not p^* may be in the subtree of q; if R = 1, then q is to

Algorithm 3: LEFTMOSTNE (x_0, y_0)

```
Input: Real numbers x_0 and y_0 defining the
            north-east quadrant Q.
   Output: The leftmost point p^* in Q \cap P, if it
               exists; otherwise the point (\infty, \infty).
 1 best = (\infty, \infty); p = root(T); q = root(T);
 \mathbf{2}
   while p is not a leaf do
       UPDATELEFTMOST(p); UPDATELEFTMOST(q);
 3
       if p = q then
 \mathbf{4}
           if p has one child then
 \mathbf{5}
               q = p_l; p = p_l;
 6
 7
           else
 8
               q = p_r; p = p_l;
       else
 9
           // p \neq q
\mathbf{10}
           if q is leaf then
11
               q = p;
12
           else if q has one child then
13
               if y(q_l) < y_0 then
14
                  q = p_r; p = p_l;
15
               else if y(p_r) < y_0 then
\mathbf{16}
                  p = p_l; q = q_l;
17
               else if x(q_l) < x_0 then
18
                   p = q_l; q = q_l;
19
               else if x(p_r) < x_0 then
20
                   p = p_r; q = q_l;
\mathbf{21}
               else
22
23
                   q = p_r; p = p_l;
           else
\mathbf{24}
               // q has two children
\mathbf{25}
               if x(p_r) \ge x_0 and y(p_r) \ge y_0 then
26
                   q = p_r; p = p_l; // Fig. 1(a)
27
               else if x(p_r) < x_0; then
28
                   if x(q_l) < x_0 then
29
                       p = q_l; q = q_r;
30
                   else if y(q_l) < y_0 then
31
                       p = p_r; q = q_r;
32
                   else
33
34
                     | p = p_r; q = q_l;
               else
35
                   // x(p_r) \ge x_0 and y(p_r) < y_0
36
                   if y(p_l) < y_0 then
37
                       p = q_l; q = q_r; // Fig. 1(b)
38
                   else
39
40
                       p = p_l;
                       if y(q_l) \ge y_0 then
41
42
                           q = q_l
43
                        else
44
                           q = q_r
   UPDATELEFTMOST(p); UPDATELEFTMOSTq;
45
46 return best;
```

the right of Q. More formally, the variables satisfy the following invariant:

- If L = 1 then $x(p) < x_0$.
- If R = 1 then $x(q) > x_1$.
- If $Q \cap P \neq \emptyset$, then $p^* \in \{best\} \cup (\cup_{z \in \mathcal{I}} T_{N(z)})$, where $\mathcal{I} = \{z \in \{L, R\} | z = 1\}$ and

$$N(z) = \begin{cases} p & \text{if } z = L, \\ q & \text{if } z = R. \end{cases}$$

• If $Q \cap P = \emptyset$, then $best = (\infty, -\infty)$.

The algorithm is given in Algorithm 4. In the initialization, the variables L, R, best, p, and q are assigned depending on the position of the root of T with respect to the query region Q.

At any moment during the algorithm, if L = 1, then we say that p is an observing point. Similarly, if R = 1, we say that q is an observing point.

Consider one iteration of the while-loop. The algorithm chooses an observing point that is closest to the root of T. (For ease of presentation, our pseudocode does not explicitly maintain the levels in T of p and q.) If this point is p, algorithm CHECKLEFT(p) is called; otherwise, algorithm CHECKRIGHT(q) is called.

```
Algorithm 4: HIGHEST3SIDED(x_0, x_1, y_0)
   Input: Real numbers x_0, x_1, and y_0 defining the
            region Q = [x_0, x_1] \times [y_0, \infty).
   Output: The highest point p^* in Q \cap P, if it
               exists; otherwise the point (\infty, -\infty).
 1 best = (\infty, -\infty);
 2 if x_0 \leq x(root(T)) \leq x_1 then
       L = 0; R = 0;
 3
 4
       if y(root(T)) \ge y_0 then
          best = root(T)
 5
 6 else if x(root(T)) < x_0 then
       p = root(T); L = 1; R = 0;
 7
 8 else
       q = root(T); L = 0; R = 1
 9
10 while L = 1 \lor R = 1 do
       \mathcal{I} = \{ z \in \{L, R\} | z = 1 \};
11
       z = element of \mathcal{I} for which level(N(z)) is
12
       minimum;
       if z = L then
13
\mathbf{14}
           CHECKLEFT(p);
       else
15
           CHECKRIGHT(q);
16
17 return best:
```

We describe the procedure for CHECKLEFT(p) in Algorithm 5. The procedure for CHECKRIGHT(q) is symmetric and omitted from this paper. Both these procedures use algorithm UPDATEHIGHEST(t), which takes as input a point t and does the following: If $t \in Q$ and y(t) > y(best) then it assigns best = t.

Algorithm 5: CHECKLEFT(p)

Input: A node p such that $x(p) < x_0$. 1 if p is a leaf then 2 L = 0**3** else if *p* has one child then if $x_0 \leq x(p_l) \leq x_1$ then 4 $\mathbf{5}$ UPDATEHIGHEST (p_l) ; L = 0; else if $x(p_l) < x_0$ then 6 7 $p = p_l$ else 8 $q = p_l; R = 1; L = 0$ 9 10 else // p has two children 11 if $x(p_l) < x_0$ then 12if $x(p_r) < x_0$ then $\mathbf{13}$ $p = p_r$ 14 else if $x(p_r) \leq x_1$ then 15 UPDATEHIGHEST (p_r) ; 16 $\mathbf{17}$ $p = p_l;$ else $\mathbf{18}$ $q = p_r; p = p_l; R = 1$ 19 else if $x(p_l) \leq x_1$ then 20 UPDATEHIGHEST $(p_l); L = 0;$ $\mathbf{21}$ if $x(p_r) > x_1$ then 22 $q = p_r; R = 1;$ $\mathbf{23}$ else $\mathbf{24}$ UPDATEHIGHEST (p_r) ; $\mathbf{25}$ else 26 $q = p_l; L = 0; R = 1$ $\mathbf{27}$

Consider the set \mathcal{I} and the value of $\ell = level(N(z))$ in lines 11 and 12 of algorithm HIGHEST3SIDED. Assume that algorithm CHECKLEFT(p) is called. During this algorithm, either p moves one level down in the tree Tor the bit L is set to 0. In addition, the point q either stays the same or it becomes a child of (the original) p. Therefore, in one iteration of the while-loop in algorithm HIGHEST3SIDED, the value of $\ell = level(N(z))$ either increases, or ℓ does not change in which case the size of the set $\{z' \in \mathcal{I} | level(N(z')) = \ell\}$ decreases. It follows that the number of iterations of the while-loop of algorithm HIGHEST3SIDED is at most twice the height of T, i.e., $O(\log n)$. Since each iteration takes O(1) time, it follows that the total time for algorithm HIGHEST3SIDED is $O(\log n)$. It follows from the algorithm that it uses O(1) extra space.

3.4 ENUMERATE3SIDED (x_0, x_1, y_0)

Given three real numbers x_0 , x_1 , and y_0 , define the three-sided range $Q = [x_0, x_1] \times [y_0, \infty)$. Algorithm ENUMERATE3SIDED (x_0, x_1, y_0) returns all elements of $Q \cap P$. This algorithm uses the same approach as algorithm HIGHEST3SIDED. Besides the two bits L and R, it uses two additional bits L' and R'. Each of these four bits L, L', R, and R' corresponds to a subtree of T rooted at the points p, p', q, and q', respectively; if the bit is equal to one, then the subtree may contain points that are in the query region Q.

Algorithm 6: ENUMERATE3SIDED (x_0, x_1, y_0)			
Input : Real numbers x_0 , x_1 , and y_0 defining the			
region $Q = [x_0, x_1] \times [y_0, \infty).$			
Output : All elements of $Q \cap P$.			
1 if $y(root(T)) < y_0$ then			
2 $L = L' = R = R' = 0$			
\mathbf{s} else if $x(root(T)) < x_0$ then			
4 $p = root(T); L = 1; L' = R = R' = 0$			
5 else if $x(root(T)) < x_1$ then			
6 $p' = root(T); L' = 1; L = R = R' = 0$			
7 else			
8 $q = root(T); R = 1; L = L' = R' = 0$			
while $L = 1 \lor L' = 1 \lor R = 1 \lor R' = 1$ do			
10 $\mathcal{I} = \{ z \in \{L, L', R, R\} z = 1 \};$			
11 $z = \text{element of } \mathcal{I} \text{ for which } level(N(z)) \text{ is}$			
minimum;			
12 if $z = L$ then			
13 ENUMERATELEFT (p) ;			
14 else if $z = L'$ then			
15 ENUMERATELEFTIN (p') ;			
16 else if $z = R$ then			
17 ENUMERATERIGHT (q) ;			
18 else			
19 ENUMERATERIGHTIN (q') ;			

The following invariant will be maintained:

- If L = 1 then $x(p) < x_0$.
- If L' = 1 then $x_0 \le x(p') \le x_1$.
- If R = 1 then $x(q) > x_1$.
- If R' = 1 then $x_0 \le x(q') \le x_1$.
- If L' = 1 and R' = 1 then $x(p') \le x(q')$.
- All points in $(Q \cap P) \setminus (\bigcup_{z \in \mathcal{I}} T_{N(z)})$ have been reported, where $\mathcal{I} = \{z \in \{L, L', R, R'\} | z = 1\}$ and

$$N(z) = \begin{cases} p & \text{if } z = L, \\ p' & \text{if } z = L', \\ q & \text{if } z = R, \\ q' & \text{if } z = R'. \end{cases}$$

The algorithm is given in Algorithm 6. In one iteration of the while-loop, the algorithm chooses an observing point that is closest to the root. Depending on this point, one of the procedures ENUMERATELEFT, ENUMERATELEFTIN, ENUMERATERIGHT, and ENUMERATERIGHTIN is called. The first two procedures are given in Algorithms 7 and 8; the other two are symmetric and omitted from this paper.

	Algorithm 7: ENUMERATELEFT (p)				
	Input : A node p such that $x(p) < x_0$.				
1	1 if p is a leaf then				
2	L = 0				
3	else if p has one child then				
4	if $x_0 \leq x(p_l) \leq x_1$ then				
5	if $L' = 1 \land R' = 1$ then				
6	EXPLORE (p') ;				
7	else if $L' = 1$ then				
8	$ q' = p'; \ R' = 1$				
9	$p' = p_l; L' = 1; L = 0;$				
10	else if $x(p_l) < x_0$ then				
11	$p = p_l$				
12	else				
13	$ q = p_l; R = 1; L = 0$				
14	else				
15	/* p has two children */				
16	if $x(p_l) < x_0$ then				
17	if $x(p_r) < x_0$ then				
18	$p = p_r$				
19	else if $x(p_r) \leq x_1$ then				
20	if $L' = 1 \land R' = 1$ then				
21	$ \begin{bmatrix} EXPLORE(p'); \\ EXPLORE(p'); \end{bmatrix} $				
22	else if $L' = 1$ then				
23	q' = p'; R' = 1				
24	$p' = p_r; p = p_l; L' = 1$				
25	else				
26	$ q = p_r; \ p = p_l; \ R = 1$				
27	else if $x(p_l) \le x_1$ then				
28	if $x(p_r) > x_1$ then				
29	$q = p_r; p' = p_l; L = 0; L' = R = 1;$				
30					
31	$\mathbf{I} \mathbf{I} \mathbf{R} = \mathbf{I} \wedge \mathbf{L} = \mathbf{I} \mathbf{then}$				
32	$ = \sum_{r=1}^{n} EXPLORE(p'); EXPLORE(p_r); $				
33	else if $L' = 1$ then EVELODE(π): $\pi' = \pi' + D' = 1$				
34	$ = \sum_{r=1}^{n} EXPLORE(p_r); q = p; R = 1 $				
35	else il $\kappa = 1$ then Explore $(m) \in L' = 1$				
36	$ EXPLORE(p_r); L = 1 $				
37	$\begin{vmatrix} e i se \\ a' - m \cdot I' - P' - 1 \end{vmatrix}$				
38	$ \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad $				
39	$ p = p_l; \ L = 0$				
40	else $a = m \cdot I = 0 \cdot P \cdot 1$				
41	$q = p_l; L = 0; \kappa = 1$				

These procedures use algorithm EXPLORE(t), which takes as input a node t in T and reports all points in T_t whose y-coordinates are at least y_0 . This algorithm does an in-order traversal of T_t , using O(1) extra space, and runs in time $O(1 + |Q \cap T_t|)$.

As in Section 3.3, it can be shown that the number of iterations of the while-loop of algorithm ENUMERATE3SIDED is at most four times the height of T, i.e., $O(\log n)$. It follows that the total time for algorithm HIGHEST3SIDED is $O(\log n + |Q \cap P|)$. It follows from the algorithm that it uses O(1) extra space.

Algorithm 8: ENUMERATELEFTIN (p')			
Input : A node p' such that $x_0 \leq x(p') \leq x_1$.			
1 if $y(p') \ge y_0$ then			
2 report p' ;			
3 if p' is a leaf then			
4 $L' = 0$			
5 else if p' has one child then			
6 if $x_0 \le x(p'_l) \le x_1$ then			
$7 p' = p'_l;$			
s else if $x(p'_l) < x_0$ then			
9 $p = p'_l; L' = 0; L = 1$			
10 else			
11 $ q = p'_l; R = 1; L' = 0$			
12 else			
13 // p' has two children			
14 if $x(p'_l) < x_0$ then			
15 if $x(p'_r) < x_0$ then			
16 $p = p'_r; L = 1; L' = 0$			
17 else if $x(p'_r) \le x_1$ then			
18 $p = p'_l; p' = p'_r; L = 1;$			
19 else			
20 $\qquad \qquad \qquad$			
21 else if $x(p'_l) \le x_1$ then			
22 if $x(p'_r) > x_1$ then			
23 $\qquad \qquad \qquad$			
24 else			
25 if $R' = 1$ then			
26 EXPLORE $(p'_r); p' = p'_l$			
27 else			
28 $q' = p'_r; p' = p'_l; R' = 1$			
29 else			
30 $ q = p'_l; L' = 0; R = 1$			

4 Conclusion

Our motivation for creating an in-place priority search tree was for designing an in-place algorithm for finding the maximum area axis-parallel empty rectangle among a set of n point obstacles in a rectangular region. It can be shown that using our in-place priority search tree one can recognize the desired rectangle in $O(m \log n)$ time using O(1) extra-space, where m is the number of all possible maximal empty rectangles. It will be worthwhile to find other applications where this tree may help in saving space.

References

- J. Katajainen and T. Pasanen. Stable minimum space partitioning in linear time. BIT, 32(4):580–585, 1992.
- [2] E. M. McCreight. Priority search trees. SIAM J. Comput., 14(2):257–276, 1985.

Orthogonal Range Search using a Distributed Computing Model

Pouya Bisadi*

Bradford G. Nickerson[†]

Abstract

We present a novel approach for distributed orthogonal range search on a set of N points stored on n nodes. The non-redundant rainbow skip graph [Goodrich et al [12] is used to coordinate message passing among nodes. We show that the maximum number of levels L in such a graph is $L = W(n \ln 2) / \ln 2$, where W is the lambertW function. Experimental validation is performed using 24 nodes, with $N = 2.4 \times 10^7$ points distributed in a uniform random fashion in a $[0,1]^2$ space. Each node stores an equal number of points, with the distribution of points among nodes controlled by point x coordinates. The experiments were implemented using the Message Passing Interface (MPI) communication model running on a high performance computer cluster. Our results show that the expected number of messages required to answer a point query originating from any node matches the theoretical bound of $\Theta(\log n)$ messages.

1 Introduction

We wish to preprocess a set S of N points into a data structure, so that for an axis aligned rectangle range query γ , the points in $S \cap \gamma$ can be reported or counted efficiently [3]. Increased reliability arises if multiple copies of the data are stored in multiple locations. In addition, increased flexibility (e.g. for access control) can arise for data maintenance by different organizations at each of the different locations. Distributed data structures are useful in these settings.

The performance measure of a data structure is related to the model of computation in which it is defined. Range search complexity is the number of memory accesses in the RAM and pointer-machine models, and the number of I/Os in the I/O model [3]. In the distributed computing model it is assumed that the cost of sending a message is higher than the cost of an I/O, so the number of messages exchanged when answering a query becomes the complexity measure.

For the last four decades, orthogonal range search was and continues to be one of the most important problems in data structures, and many people worked on it [10]. The most efficient data structure for worst case 2-dimensional range queries in the RAM model is a modified version of the layered range tree, described by Chazelle [6]. Chazelle succeeded in improving the storage to $O(\frac{N \log N}{\log \log N})$ with query time of $O(\log N + k)$, for k points reported in range. Chazelle [7, 8] also proved that this time and space bound are optimal in the worst case. Arge et al [4] provided a two dimensional I/O-efficient structure for general range searching which occupies $O(\frac{N \log (N/B)}{B \log \log B})$ disk blocks and answers queries in $O(\log_B N + \frac{T}{B})$ I/Os, which are optimal in the worst case. Afshani et al [2] presented a space optimal pointer machine data structure for 3-d orthogonal range reporting that answers queries in $O(\log N + k)$ time.

None of these optimum solutions considers a distributed model where reducing node congestion and improving fault tolerance are important. Sridhar et. al. [18] presented a parallel algorithm to report the in-range set of points in a rectangular range-search in $O(\log N)$ time, with $O(\log^2 N)$ processors on an EREW-PRAM model (Exclusive-Read-Exclusive-Write Parallel Random Access Model). A shared memory model presented by Sridhar et. al. [18] can be used in a network, but they did not consider fault tolerance and reliability because their model is for a single machine with multiple processors. Hash functions do not preserve the order of keys and methods like Chord [19] and CAN [17] which use distributed hash tables (DHT) are good for lookup (single point) queries. Aspnes and G. Shah [5] have presented a distributed data structure which supports range queries along single atribute 1-D. However, the Skip Graph stores $\log n$ pointers for each node and asigning one point to each node requires $n \log n$ space which is not practical.

For range search using a distributed model, the basic idea is to divide S into n subsets (maybe with overlap) and to distribute them among n nodes. Each one of these nodes can be a representative of a host in a physical network. Generally, in a distributed data structure each node has a key (or name) m and an address a (like an IP address), so a pointer to a node is a pair (m, a). A lookup query in these data structures can be interpreted as "What is the address of the node that has the key m?". In the case of range search, the question is a bit different; i.e. "What are the addresses of the nodes storing points intersecting with the range query γ ?". We would like to find the answer to this question by sending the minimum number of messages. The query can

^{*}Faculty of Computer Science, University of New Brunswick, j3ngr@unb.ca

[†]Faculty of Computer Science, University of New Brunswick, bgn@unb.ca

be issued from any node u among the n nodes. Once the destination nodes are found, within-node search can be performed using e.g. an I/O-efficient data structure supporting range search.

Many distributed data structures have been presented for general applications in a distributed model. The skip graph [5], family tree [20] and rainbow skip graph [12] are a few of them. Zatloukal and Harvey [20] use a modified SkipNet [15] to construct a structure they call the family tree, achieving $O(\log n)$ expected messages for search and update, while restricting required space (number of stored pointers) for each node to be O(1), which is optimal.

Goodrich et al [12] presented a peer-to-peer data structure called the rainbow skip graph that achieves high fault-tolerance, constant-sized nodes, and fast update and query times for ordered data. In this paper, a non-redundant rainbow skip graph [12] is used for routing purposes.

2 Our Results

We utilized the non-redundant rainbow skip graph to implement an orthogonal range search structure. To our knowledge, this is the first implementation of this routing data structure for range search on spatial data. In this data structure, the cost of search is independent of the query issuer. Our experimental results support this statement. We prove that the maximum number of levels in a rainbow skip graph is $L = \frac{W(n \ln 2)}{\ln 2}$ where W is the lambertW [9] function and n is the number of nodes in the non-redundant rainbow skip graph [12].

3 Data Structure and Search Algorithm

3.1 Non-Redundant Rainbow Skip Graphs

A skip graph [5] is a distributed data structure which consists of skip lists [16]. It has all the functionality of a balanced tree in a distributed system and its algorithms for insertion and deletion are the same as a skip list (see Figure 1). The search algorithm in a skip graph is almost the same as searching a skip list. The main difference is that every node is in every level of a skip graph.

To implement a distributed orthogonal range search data structure, a non-redundant rainbow skip graph is used because it provides all the features necessary for a general purpose peer–to–peer data structure. Based on the Goodrich et al [12] definition, a non-redundant rainbow skip graph on n nodes consists of a skip graph [5] on $\Theta(\frac{n}{\log n})$ supernodes, where a supernode consists of $\Theta(\log n)$ nodes that are maintained in a doubly-linked list called the core list of the supernode. As explained in the next section, $\Theta(\log n)$ is not the optimum size of supernodes. The keys of the nodes of each supernode are



Figure 1: An example of skip graph from [11]. The levels are separated by dashed lines.

a contiguous subsequence of the ordered sequence of all keys. The smallest key of each supernode V is referred to as the key of V, and the skip graph is defined over these keys. For each supernode V, a different member of V is associated with each level i of the skip graph, and this member is the level *i* representative of V, which is denoted as V_i . The level *i* list to which V belongs contains V_i . These lists of the skip graph are called the level lists. V_i , which can be chosen arbitrarily from among the elements of V, is connected to V_{i+1} and V_{i-1} , which respectively are called the parent and child of V_i . These vertical connections form another linked list associated with supernode V that is referred as the *tower list* of V. Each supernode has one tower list. Each element of a supernode is a member of at most three lists; the core list, the tower list, and one level list. Figure 2 shows a rainbow skip graph created over the same data shown in Figure 1. For example, to search for the key 19 from node 2, nodes 2, 5, 22, 16, 27, 16, 22 are visited to find that key 19 is not in the graph.



Figure 2: Nine nodes are grouped into four supernodes to create a rainbow skip graph with two levels and four tower lists.

3.2 Supernode Size

Lemma 1 Considering that a node cannot be in multiple level lists, the maximum number of levels for a nonredundant rainbow skip graph with n nodes is when the size of all supernodes are equal to the number of levels.

Proof. A non-redundant rainbow skip graph is the result of creating a skip graph from supernodes (groups of nodes) with each supernode belongs to many level lists. There are two constraints: First, at each level i a different member of a supernode is the supernode representative V_i in that level, and a supernode cannot be a member of level i + 1 if it is not a member of level i. If the number of nodes in a supernode is less than the number of levels in the skip graph, this supernode cannot be in all the levels.

As the second constraint, the number of supernodes is $\frac{n}{|V|}$ and the number of levels is related to the number of supernodes. Therefore the number of levels $\log_2 \frac{n}{|V|}$ is related to size of the supernodes. If |V| goes up, the number of levels goes down.

Therefore, the number of levels is always the minumim of |V| and $\log_2 \frac{n}{|V|}$ (see Figure 3).



Figure 3: Relationship of the size of supernodes |V| and number of levels L when n = 24.

Theorem 2 The maximum number of levels L for a non-redundant rainbow skip graph is

$$L = \frac{W(n\ln 2)}{\ln 2} \tag{1}$$

where n is the number of nodes and W is the lambertW function which is the inverse function of $z = We^{W}$.

Proof. When every node is a member of a level list, there is no node that is just in the core list of its supernode. In other words, to maximize the value of L from Lemma 1, the size of supernode V and the number of levels L should be equal. Such a rainbow skip graph has L levels and each of its supernodes has L members as their level representative. Consequently, there are 2^L supernodes with a size of L. As a result, the number of all nodes is $L \times 2^L = n$. Solving this equation for L gives the value for n which is based on the *lambertW* [9] function shown in equation (1).

3.3 Data Distribution

Routing in the non-redundant rainbow skip graph requires a set of keys having a total order relation. Having a total order relation [13] on the set of keys makes it possible for each node to determine whether the requested node of a query is one of the successor or predecessor nodes without knowledge about the whole data structure. Therefore, each node can route the received query message in the correct direction.



Figure 4: A distribution of a 2D space among 5 nodes. The hatched area is a rectangular query. Also, the lower and upper bounds L_2 and U_2 of the second node region are labelled.

To use the non-redundant rainbow skip graph, the entire space must be split into regions in such a way that a total order binary relation [13] (here denoted by \leq) is definable on this set of regions. The simplest distribution is splitting points based on one of their coordinates (e.g. x) which is shown in Figure 4. For example, a suitable key for the presented distribution in Figure 4 can be $m(L_i, U_i)$ where i is the node number, and L_i and U_i are the lower and upper x coordinate bounds of node region, respectively. In this way, the key set has a total order based on the x coordinate.

In orthogonal range search, if a point is not in the query range in one dimension, its coordinate value (in that dimension) is either greater or lower than all the points in the query range. As the distribution of points is based on one dimension, if the query range has intersection with nodes i - 1 and i + 1, it has intersection with node i for sure. Consequently, having the address of node i whose region intersects the query $\gamma([L_x, U_x], [L_y, U_y])$, node i - 1 should be checked too unless $L_i \leq L_x \leq U_i$ and node i + 1 should be checked unless $L_i \leq U_x \leq U_i$.

3.4 Range Search

We created a non-redundant rainbow skip graph using the lower bound of each node region as its key. The rainbow skip graph normal routing algorithm is used to find the node whose region covers L_x . In order to answer a query $\gamma([L_x, U_x], [L_y, U_y])$ from node u, the rainbow skip graph search algorithm [12] is used to find the node that stores the lower bound of γ . First we find the top level representative of the supernode of node u. Then, by a standard skip graph [5] search, we find the supernode whose key (x lower bound) is the maximum key lower than L_x . The next step is performing a linear scan through the core list to find the first node that stores points in range query γ . Then, this node reports the points to u (the query issuer) and passes the query to its successor node if the upper bound of the query (U_x) is outside its region; the next node does the same. Each of these steps (except the reporting part) requires $O(\log n)$ messages, then the complexity of point search using the non-redundant rainbow skip graph is $O(\log n)$ messages.

In the distributed computing model, the notion of failed nodes is important. If no messages are received in response to a query, we assume the nodes intersecting the query range have failed. In the worst case, a query intersects all n regions, but finds no points in range. A message indicating this empty set is required at each queried node. This leads to O(n) messages for range search on a set of N points distributed on n nodes of a non-redundant rainbow skip graph. The same worst case search complexity holds if k > 0 and we assume O(1) messages can hold the k points reported in range.

4 Experimental Validation

To test this data structure, 2.4×10^7 two dimensional points drawn from a uniform random distribution \in $[0, 1]^2$ are distributed based on their *x* coordinate among 24 nodes. Therefore each node covered around 4% of the whole area. Around 2,400 queries (see Figure 6) were randomly generated such that:

- query center $(x_i, y_i) \in [0.1, 0.9]^2$
- lower bound $(x_L, y_L) = (x_i, y_i) (\Delta x_i, \Delta y_i)$
- upper bound $(x_U, y_U) = (x_i, y_i) + (\Delta x_i, \Delta y_i)$
- Δx_i and Δy_i are uniform random $\in [0.0, 0.1]$

The experiments used the Message Passing Interface [14] (MPI) on the Atlantic Computational Excellence Network [1](ACEnet). Figure 5 shows the implemented data structure. This structure consists of three levels and each supernode has 3 nodes which are its representatives in different levels. Notice that in this figure, levels are shown by different dashed lines for their links.



Figure 5: The rainbow skip graph which is used for experimental validation. Level lists are shown by curved lines. There are 8 supernodes, each containing three nodes.

As the tower list is exactly the same as the core list, the maximum number of connections for each node is 4. The program uses 25 slots, the first one (index 0) as the test harness and the rest as the nodes. Node 0 randomly sent commands for issuing queries to nodes and collected the data. We made the simplifying assumption that messages are big enough for nodes to report back all the points in their intersection with γ in one message.



Figure 6: The center and size of query rectangles are generated from a uniform random distribution function.

As shown in Figure 7, each query was answered by passing an average of 13 messages through the network, with the number of passed messages averaging no more than 16 and no less than 11.

Figure 8 and 9 show two graphs representing the relationship between query cost and size of query rectangles. Each dot in figures 8-11 represents one of the 2,400 range query results. As expected, the number of messages is independent of the height of query rectangles because each node covers the height of the total area. The cost for responding to queries rises linearly with increasing query rectangle width.

As shown in Figure 10, the query cost is from 1 to 24 messages when the area of the query is small (see



Figure 7: The average number of messages for answering queries issued from each node.



Figure 8: The number of messages for answering queries is not related to the height of query rectangles. The black line is a linear trendline for the number of messages.



Figure 9: Number of messages increases linearly with increasing query rectangle width. The black line is a linear trendline for the number of messages.

Figure 10). The maximum messages arising on the left of Figure 10 is high as a very thin horizontal query rectangle requires a lot of message passing to answer even if it has a relatively small area.

The maximum cost of queries with aspect ratio less than one is lower than the queries with aspect ratio > 1 (see Figure 11). There is a higher probability for low



Figure 10: Number of messages increases with growth of the query size since the size is dependent on query width. The black line is a linear trendline for the number of messages.

aspect ratio query rectangles to have a smaller width and thus intersect with fewer node regions.



Figure 11: Number of messages rises linearly with increasing query rectangle aspect ratio.

5 Simulation

We performed a simulation of the performance of the non-redundant rainbow skip graph for n = 24, 36, 48, 64, 80 and 96. The results are shown in Figure 12. The average cost to find the first node in range corresponds to a point search issued from any node. As Figure 12 shows, a point search costs $< 2 \log_2 n$ messages, which matches the expected number of messages reported in [12]. Figure 12 also shows the average cost that includes messages sent to report the points in range.

6 Conclusion

The aim of using a distributed model for orthogonal range search is to provide reliability, flexibility and robustness to the data structure. In this paper we presented a novel approach for distributed orthogonal range search using the non-redundant rainbow skip graph.



Figure 12: Average number of messages to answer one query based on the number of nodes.

We proved that the maximum number of levels in a non-redundant rainbow skip graph occurs when the size of each supernode is equal to number of levels. The maximum number of levels $L = \frac{W(n \log 2)}{\log 2}$. We also showed experimentally that a point search cost requires $\Theta(\log n)$ messages, which matches the expected results in Goodrich et al [12]. The experimental results showed that distributed range search cost using the nonredundant rainbow skip graph was independent of which node issued the query. In addition we showed that the number of messages required to answer a range query increased linearly with increasing query rectangle width.

It remains to determine the optimum size of a supernode in the non-redundant rainbow skip graph such that the number of messages passed to answer a range query is minimized.

7 Acknowledgements

The authors would like to acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada and the UNB Faculty of Computer Science.

References

- [1] The atlantic computational excellence network (http://www.ace-net.ca/wiki/acenet).
- [2] P. Afshani, L. Arge, and K. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of the 2010 annual symposium on Computational geometry*, pages 240–246. ACM, 2010.
- [3] P. K. Agarwal. Range searching. CRC Handbook of Discrete and Computational Geometry. CRC Press, Inc., 2004.
- [4] L. Arge, V. Samoladas, and J. Vitter. On twodimensional indexability and optimal range search

indexing. In Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 346–357. ACM, 1999.

- [5] J. Aspnes and G. Shah. Skip graphs. ACM Transactions on Algorithms, 3(4), 2007.
- [6] B. Chazelle. Filtering search: A new approach to queryanswering. SIAM J. Comput., 15(3):703-724, 1986.
- [7] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. J. ACM, 37(2):200–212, 1990.
- [8] B. Chazelle. Lower bounds for orthogonal range searching: Ii. the arithmetic model. J. ACM, 37(3):439–463, 1990.
- [9] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth. On the lambertw function. Advances in Computational mathematics, 5(1):329–359, 1996.
- [10] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry. Springer, 2008.
- [11] M. Goodrich, M. Nelson, and J. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proceedings of the seventeenth annual* ACM-SIAM symposium on Discrete algorithm, pages 384–393. ACM, 2006.
- [12] M. Goodrich, M. Nelson, and J. Sun. The rainbow skip graph: A fault-tolerant constant-degree p2p relay structure. pages 384–393, New York, NY, USA, 2009. ACM.
- [13] G. Gratzer. Lattice theory. WH Freeman, 1971.
- [14] W. Gropp, E. Lusk, and A. Skjellum. Using MPI: portable parallel programming with the message passing interface. 1999.
- [15] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: a scalable overlay network with practical locality properties. In USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, pages 9–9, Berkeley, CA, USA, 2003. USENIX Association.
- [16] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM, 33(6):668– 676, 1990.
- [17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. pages 161–172, 2001.
- [18] R. Sridhar, S. Iyengar, and S. Rajanarayanan. Range search in parallel using distributed data structures. *Journal of Parallel And Distributed Computing*, 15(1):70–74, 1992.
- [19] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIG-COMM Computer Communication Review, 31(4):149– 160, 2001.
- [20] K. C. Zatloukal and N. J. A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pages 308–317, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

On Finding Skyline Points for Range Queries in Plane

Anil Kishore Kalavagattu *

Ananda Swarup Das[†]

Kishore Kothapalli[‡]

Kannan Srinathan §

Abstract

We consider the dominating point set reporting problem in two-dimension. We propose a data structure for finding the set of dominating points inside a given orthogonal query rectangle. Given a set of n points in the plane, it supports 4-sided queries in $O(\log n + k)$, where k is size of the output, using $O(n \log n)$ space. This work can be of application when range queries are generated using mobile devices with limited display capacity.

1 Introduction

Range searching is one of the widely studied topics in computational geometry. As stated in [1], the fascination for range searching is due to the fact that it has wide applications in geographic information systems, CAD tools, database retrieval, etc. Informally range aggregate is defined as follows: We are given a set S of objects which can be points or line segments and the like. We need to preprocess the data set into a data structure such that given a query object q, we can efficiently return the objects in $S \cap q$, the objects that belong to both S and q. A careful note of the above definition reflects that in a traditional range aggregate query, an algorithm designer is more focused in finding the result set efficiently and is not much bothered about the size of the result set. But this cannot be true any further. The advancement of technology has introduced the era of information revolution which led to information explosion. Imagine a user who is accessing a database in a server through his mobile device whose computing power as well as display model mechanism is sufficiently limited. Computation can be outsourced to server but returning the entire result set may not be a clever decision. Rahul et al. [7] have proposed an algorithm to return the top k answers of the result set. But for this purpose, our points in the data set have to be weighted by means of a preference function. But in practice, finding a good preference function is not an easy task. To address the issue, we borrow the concept of skyline query from the database community.

2 Definitions

We are given a set of n points $P = \{p_1, \ldots, p_n\}$ in \mathbb{R}^2 . Assuming all the points have distinct coordinates in each dimension, a point p_i is said to dominate a point p_i if $p_i(x) > p_i(x)$ and $p_i(y) > p_i(y)$. The set $P' (\subseteq P)$ of all the points, each of which is not dominated by any other point in P is called the dominating set of P. They are also called maximal elements, or maxima, of the set P [2]. In the database terminology, these maximal points are also known as skyline points. A skyline query is then, given a set of points P, report the skyline points of P. The skyline point set P' forms a sample (representation) for the set P. In practice, often |P'| < |P|. However in the worst case scenario, |P'| = |P|. More information about skyline queries can be found in [6].

3 The Problem



Figure 1: The nodes filled black are not dominated by any other point. a.) skyline of all the points. b.) skyline of points in the query rectangle (dotted border).

In this work, we study the following problem.

Problem 1 We are given a set S of n points in \mathbb{R}^2 . We wish to pre-process S into a data structure such that given an orthogonal query rectangle q, we can efficiently report the skyline points of $S \cap q$, where $S \cap q$ is the set of points in both S and q.

Consider Fig 1.a There are ten points and among them only four points are not dominated by any other point. The same set of points are queried with an orthogonal rectangle, as shown in Fig 1.b and its easy to see that

^{*}International Institute of Information Technology, Hyderabad, India, anilkishore@research.iiit.ac.in

[†]International Institute of Information Technology, Hyderabad, India, anandaswarup@gmail.com

[‡]International Institute of Information Technology, Hyderabad, India, kkishore@iiit.ac.in

[§]International Institute of Information Technology, Hyderabad, India, srinathan@iiit.ac.in

the result can contain points which may not be in the skyline point set obtained by considering all the points. In rest of the paper, whenever we say size of result, we actually mean number of skyline points inside the query rectangle.

4 The General Algorithm



Figure 2: The query rectangle is $q = [P_1(x), P_2(x)] \times [P_1(y), P_2(y)]$ and the point $P_3 = (P_3(x), P_3(y))$ is the point with the largest x coordinate inside q.

A sketch of solution for the problem is as follows:

- 1. Let the set of points in the plane be S and the query rectangle be $q = [P_1(x), P_2(x)] \times [P_1(y), P_2(y)].$
- 2. Find the point with the largest x coordinate in $S \cap q$. Let the point be $P_3 = (P_3(x), P_3(y))$. Add P_3 to the skyline point set.
- 3. Create a new rectangle $q' = [P_1(x), P_3(x)] \times [P_3(y), P_2(y)]$ and update $q \leftarrow q'$. See Fig. 2.
- 4. Repeat the steps 2, 3 until we get a rectangle q' such that $S \cap q' = \emptyset$. All the intermediate query rectangles encountered are also shown in Fig. 2.

As can be seen, our algorithm depends on a solution of range successor problem in plane. Range successor problem is widely studied in literature and Yu et al. in [9] proposed a data structure of size O(n) using which range successor queries can be efficiently answered in $O(\frac{\log n}{\log \log n})$ time. Let k be the size of output, then a solution which repeatedly uses range successor query in step 2 above, will have a query time of $O(k \frac{\log n}{\log \log n})$. We propose a data structure of size $O(n \log n)$ using which our algorithm will have a query time of $O(\log n + k)$. Clearly using the proposed data structure, our algorithm will perform better whenever $k \ge O(\log n)$. Here we considered the static version of the problem, where the input points are fixed. A dynamic version of this problem with $O(\log^2 n + k)$ query time using $O(n \log n)$ space and $O(\log^2 n)$ time per update is studied in [4].

5 The Data Structure

5.1 Preprocessing

The data structure is a standard layered range tree [3] in which the main tree stores the points sorted by xcoordinates. Each secondary structure is an array storing *u*-coordinates in sorted order (decreasing), along with the corresponding point. This can be constructed by carrying out merge sort using y-coordinates as keys. In order to speed up query time, we use fractional cascading [3] at the cost of storing additional pointers at each node. Let w and v be the two children of μ . While merging the secondary arrays A_w and A_v to construct A_{μ} , we create and store pointers as follows. Each index i of A_{μ} stores a pointer to the smallest value in A_{w} which is greater than or equal to $A_{\mu}[i]$ and a pointer to the largest value in A_w which is smaller than or equal to $A_{\mu}[i]$. Similarly, two more such pointers are stored pointing to elements of A_v . Also, each index of A_w and A_v has a pointer to its corresponding position in the parent array A_{μ} . Each of these arrays A is preprocessed for range maxima queries [10] such that given two indices i, j of A_{μ} , we can find the point with maximum x coordinate among the points whose y coordinates are stored between $A_{\mu}[i]$ and $A_{\mu}[j]$ in O(1) time.

Lemma 1 The storage space needed for the data structure is $O(n \log n)$. The preprocessing time needed to construct the data structure is $O(n \log n)$.

This data structure is similar to the data structure used in [5] [8].

5.2 The Query Algorithm



Figure 3: The dark nodes are the ones to which the segment [a, b] of the query $q = [a, b] \times [c, d]$ is allocated.

After we construct the data structure, the query algorithm for a query rectangle $q = [a, b] \times [c, d]$ is as follows.

- 1. The range of x-coordinates in [a, b] can be expressed as the disjoint union of $l = O(\log n)$ canonical subsets. Let the canonical subsets of nodes be $\nu_1, \nu_2, \ldots, \nu_l$ from left to right in that order, as shown in Fig. 3.
- 2. Find the node ν_{split} , which is the least common ancestor of ν_1 and ν_l . Find the largest sub-range of y-coordinates $\in [c, d]$ in $A_{\nu_{split}}$ using binary search and store the indices of the two ends.
- 3. Process the canonical nodes in reverse order, starting from ν_l back to ν_1 , as follows. Initialize $i \leftarrow l$, $y_{low} \leftarrow c$ and $y_{high} \leftarrow d$.
- 4. Consider the node ν_i . Find the smallest index ltand the largest index rt such that $y_{high} \ge A_{\nu_i}[lt] \ge A_{\nu_i}[rt] \ge y_{low}$. Note that the y-coordinates are sorted in decreasing order in A_{ν_i} . This can be done by following the pointers from $A_{\nu_{i+1}}$ along the path to the node ν_i . For the starting node ν_l , follow the pointers from $A_{\nu_{split}}$ found in step 2.
- 5. Find the point with the largest x coordinate in $A_{\nu_i}[lt \dots rt]$ using a range maxima query. Let this point be p' and its y-coordinate be p'(y). Report the point p' and move rt to the position of the array just before p' and update $y_{low} \leftarrow p'(y)$.
- 6. While there are points still left in $A_{\nu_i}[lt \dots rt]$, i.e., $lt \leq rt$, repeat the above step.
- 7. At this point, we processed the nodes $\nu_l, \nu_{l-1}, \ldots, \nu_i$ and also have two pointers to the current limits on the y-coordinate $[y_{low}, y_{high}]$ at ν_i .
- 8. Set $i \leftarrow i 1$. If $i \ge 1$, move to the node ν_i along with the pointers and repeat from step 4, else exit.

Lemma 2 The time needed for the query algorithm above is $O(\log n + k)$.

Proof : At each of the $O(\log n)$ levels of the tree, at most two nodes are visited [3]. Among them, each canonical node with all its *x*-coordinates $\in [a, b]$ is visited at most once and each of the other nodes along the path traversed is visited at most three times, first time from its parent and at most one more time from each of its two children \square

Using the above lemma, we can now conclude the section stating the following theorem.

Theorem 3 A set S of n points in \mathbb{R}^2 can be preprocessed into a data structure of size $O(n \log n)$ such that given an orthogonal query rectangle q, we can efficiently report the set of points in $S \cap q$ not dominated by any other point in $S \cap q$ in $O(\log n + k)$ time where k is the size of the output.

Our solution can be easily modified to report the topm sky line points having largest(or smallest) y(or x)coordinates in $O(\log n + m)$ time.

6 Conclusion

In this work we studied the problem of finding the dominating set of points inside a query rectangle. Our solution is static and restricted to the plane. It will be interesting to see dynamic version of this problem and in higher dimensions.

7 Acknowledgements

We would like to thank the reviewers for their helpful and positive comments which have improved the paper and for pointing to [4] which handles the dynamic version of the problem discussed in this paper.

References

- P. Agarwal, S. Govindrajan, S. Muthukrishnan. Range Searching in Categorical Data: Colored Range Searching on Grid. *In Proceedings of ESA*, pp. 17–28, 2002
- [2] Jon Louis Bentley. Multidimensional divide-andconquer. Communications of the ACM, v.23 n.4, p.214-229, April 1980
- [3] M. de. Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry: Algorithms and Applications. *ISBN 3-540-65620-0, Springer Verlag*, 2000
- [4] G. S. Brodal, K. Tsakalidis. Dynamic Planar Range Maxima Queries. In Proc. 38th International Colloquium on Automata, Languages, and Programming, vol 6755 of LNCS, pp. 256-267. Springer Verlag, Berlin, 2011
- [5] A. S. Das, P. Gupta, K. Srinathan, K. Kothapalli. Finding Maximum Density Axes Parallel Regions for Weighted Point Sets. *submitted to CCCG 2011*
- [6] D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In Proc. International Conference on Very Large Data Bases, pp. 275-286, 2002
- [7] S. Rahul, P. Gupta, R. Janardan, K. S. Rajan. Efficient top-k queries for orthogonal ranges. In Proc. International Workshop on Algorithms and Computation, Springer Verlag LNCS No. 6552, pp. 110–121
- [8] Sanjeev Saxena. Dominance made simple. Information Processing Letters, v.109 n.9, p.419-421, April, 2009
- [9] C. C. Yu, W. K. Hon, B. F. Wang. Improved Data Structures for Orthogonal Range Successor Queries. *Computational Geometry: Theory and Applications* 44 , pp. 148–159, 2011
- [10] H. Yuan, M. Atallah. Data Structures for Range Minimum Queries in Multidimensional Arrays. In Proceedings of SODA, pp. 150–160, 2010

Space-efficient Algorithms for Empty Space Recognition among a Point Set in 2D and 3D

Minati De^{*†}

Subhas C. Nandy^{*}

Abstract

In this paper, we consider the problem of designing in-place algorithms for computing the maximum area empty rectangle of arbitrary orientation among a set of points in 2D, and the maximum volume empty axisparallel cuboid among a set of points in 3D. If n points are given in an array of size n, the worst case time complexity of our proposed algorithms for both the problems is $O(n^3)$; both the algorithms use O(1) extra space in addition to the array containing the input points.

1 Introduction

Designing low memory algorithms is considered to be an important paradigm for the data-streaming and datamining applications. Here the amount of data available is huge, and it is wise to consider as much data as possible to get precise result. In other areas also, the low memory algorithms are important in spite of the fact that computer hardware has become extremely cheap now-a-days. For an example, consider the VLSI physical design applications, where the number of circuit modules in a VLSI chip are rapidly growing day by day, and running the standard routing, placement, verification algorithms, are becoming impossible even in the modern computers due to the size of data. In sensor network applications, it is often found that in order to get precise information, a huge number of sensors are deployed. Moreover in tiny devices, for example, sensors, GPS systems, mobile hand-sets, small robots, etc, in order to maintain its size, one needs to lower down the memory size. For all these reasons, designing lowmemory algorithms for practical problems have become a challenging task to the algorithm researchers.

In computational geometry, in-place algorithms are studied for a very few problems. For the convex hull problem in 2D, the best known result is an $O(n \log h)$ algorithm with O(1) extra space [5]. Bronnimann et al. [4] showed that the upper hull of a set of n points in 3D can be computed in $O(n \log^3 n)$ time using O(1) extra space. The best known algorithm for this problem runs in $O(n \log n)$ expected time [8]. Bose et al. [3] used an in-place divide and conquer technique to solve the following problems in 2D using O(1) extra space: (i) a deterministic $O(n \log n)$ time algorithm for the closest pair problem, (ii) a randomized expected $O(n \log n)$ time algorithm for the bichromatic closest pair problem, and (iii) a deterministic $O(n \log n + k)$ time algorithm for computing the intersections among orthogonal line segments. For computing the intersections among arbitrary line segments, two algorithms are available in [7]. If the input array can be used for storing intermediate results, then the problem can be solved in $O((n+k)\log n)$ time and O(1) space. but, if the input array is not allowed to be destroyed, then the time complexity increases by a factor of $\log n$; it also requires $O(\log^2 n)$ extra space. Vahrenhold [15] proposed an $O(n^{\frac{3}{2}} \log n)$ time and O(1)extra space algorithm for the Klee's measure problem, where the objective is to compute the union of n axisparallel rectangles of arbitrary sizes. Asano and Rote [2] showed that all the Delaunay triangles among a set of n points can be computed in $O(n^2)$ time using O(1)space. This, in turn, recognizes the largest empty circle among a point set with the same time complexity.

We will now consider the algorithms for recognizing the maximum area empty rectangle among a set of npoints in a region \mathcal{R} in 2D. The axis-parallel version of the problem was first introduced by Namaad et al. [13]. They introduced the concept of maximal empty rectangle (MER). It is an empty rectangle, not properly contained in any other empty rectangle. They showed that the number of MERs (m) among a set of npoints may be $\Omega(n^2)$ in the worst case; but if the points are randomly placed, then the expected value of m is $O(n \log n)$. In the same paper, an $O(min(n^2, m \log n))$ time algorithm for identifying the largest MER was also proposed. Orlowski [14] proposed an $O(m + n \log n)$ time algorithm for finding the largest MER that inspects all the MERs present in \mathcal{R} , and identifies the largest one. The best known algorithm for this problem runs in $O(n \log^2 n)$ time in the worst case [1]. All these algorithms use O(n) extra space. The worst case time and space complexities for computing the largest empty rectangle of arbitrary orientation among a set of n points are $O(n^3)$ and $O(n^2)$ respectively [6]. Recently, an in-place algorithm for recognizing the largest empty axis-parallel rectangle is proposed that runs in $O((m+n)\log n)$ time and uses O(1) extra space in ad-

^{*}Indian Statistical Institute, Kolkata, India.

[†]Presently visiting Carleton University, Canada. minati.isi@gmail.com

dition to the array containing the input points [9]. It uses a novel way of maintaining priority search tree in an in-place manner. In 3D, the largest empty axis-parallel cuboid among a set of n points in an axis-parallel cuboid region \mathcal{R} can be computed in $O(C + n^2 \log n)$ time with O(n) extra space, where C is the number of maximal empty axis-parallel cuboids in \mathcal{R} , which may be $O(n^3)$ in worst case [12].

We first describe an in-place algorithm for computing the maximum area empty rectangle of any arbitrary orientation among a set of n points in a 2D rectangular region. We will also consider a simplified 3D version of the problem, where the objective is to identify the maximum volume empty axis-parallel cuboid among a set of n points in a 3D axis-parallel region. The time complexity of both the algorithms is $O(n^3)$, and they need O(1) space in addition to the input array.

2 Computing largest MER of arbitrary orientation

We now propose an in-place algorithm for finding maximum area empty rectangle of arbitrary orientation among a set of points P inside a rectangular region \mathcal{R} . The problem was addressed by Chaudhuri et al. [6]. They introduced the concept of PMER. A PMER, defined by four points $p_i, p_j, p_k, p_\ell \in P$, is the maximum area rectangle of any arbitrary orientation whose four sides pass through p_i, p_j, p_k and p_ℓ , and the interior of the rectangle does not contain any member of P. It is shown that the number of PMERs is bounded above by $O(n^3)$. It follows from the following observation:

Observation 1 [6] At least one side of a PMER must contain two points from P, and other three sides either contain at least one point of P or the boundary of \mathcal{R} .

2.1 Algorithm

Observation 1 plays the central role in our algorithm. We consider each pair of points $p, q \in P$, and compute all the PMERs with one side passing through p, q. We use geometric duality for solving this problem. The duality transform in 2D maps a point $p = (\alpha, \beta)$ in the primal plane into a line $p' = \alpha x - \beta$ in the dual plane and maps a non-vertical line $\ell : y = mx - c$ in the primal plane into the point $\ell' = (m, c)$ in the dual plane. For the standard properties of duality transform, see [10].

Observation 2 Let v be a point on a vertical line \mathcal{L} in the dual plane, and q'_1, q'_2, \ldots, q'_m be m lines in the dual plane that intersect \mathcal{L} in one side (above or below) of v, and are arranged in increasing order of their distances from v along \mathcal{L} . Now, all the points q_1, q_2, \ldots, q_m are in one side (below or above) of the line v' in the primal plane and the perpendicular distances of q_1, q_2, \ldots, q_m from the line v' are also in increasing order.

We will consider the arrangement $\mathcal{A}(P)$ of the set of dual lines corresponding to all the points in the array P. Its each vertex v_{ij} obtained by the intersection of the dual lines p'_i and p'_j , corresponds to the line ℓ_{ij} passing through $p_i, p_j \in P$ in the primal plane. Thus, in order to get the lines passing through each pair of points in P, we need to visit all the vertices in $\mathcal{A}(P)$.

Note that, each element of P corresponding to an input point also represents the corresponding dual line. We first identify the left-most vertex in $\mathcal{A}(P)$ by computing the intersections of all the $O(n^2)$ pairs of dual lines. Now, a vertical line \mathcal{L} starts sweeping from that position. We execute a sorting step to arrange the members in P such that the y-coordinates of the points of intersection of those dual lines and the sweep line \mathcal{L} are in increasing order. Thus, P also serves the role of the sweep line status array. During the sweep, this property of P is always maintained. Here the two lines, say p'and q', incident to the next vertex $v \in \mathcal{A}(P)$ will remain consecutive, say at P[i] and P[i+1]. All the lines below (resp. above) v are to the right of P[i+1] (resp. left of P[i]) in the array P, and are in increasing order of their distances from the point v along the line \mathcal{L} . We process v to compute all the MERs whose one side passes through (p,q) using the procedure **process**(p,q). The procedure **get_next_vertex** computes the next vertex of $\mathcal{A}(P)$ that \mathcal{L} faces to the right of v during the sweep.

2.1.1 get_next_event

After processing a vertex v (intersection of a pair of dual lines p' and q' stored at P[i] and P[i+1] respectively), when \mathcal{L} moves to the right of v, p' and q' are swapped in P for maintaining their order along \mathcal{L} . We do not maintain the event queue. At each step, we compute the next vertex in $\mathcal{A}(P)$ to be processed.

Observation 3 [11] At any instant of time during the sweep, the vertex closest to \mathcal{L} to its right side is the point of intersection of a pair of dual lines that are consecutive in the ordered list of dual lines.

We compute the intersection of each pair of consecutive dual lines in the array P. If it is to the right of \mathcal{L} , then it is a *feasible intersection point* (FIP). By Observation 3, The next vertex of $\mathcal{A}(P)$ to the right of \mathcal{L} corresponds the left-most FIP. If no such FIP is obtained, the sweep stops. Thus, the time complexity for getting the next vertex of $\mathcal{A}(P)$ for processing is O(n).

2.1.2 Process(p,q)

Let v be the vertex in $\mathcal{A}(P)$ under process. It corresponds to the pair of points $p, q \in P$ stored at P[i] and P[i+1] respectively. Let λ be the straight line passing through p, q. By Observation 2 the points below λ are $\Pi_1 = \{P[i+2], P[i+3], \ldots, P[n]\}$ in increasing order of their distances from λ . We now describe the method of computing all the PMERs with (p,q) at its top boundary. The method of computing all the PMERs with (p,q) at their bottom boundary with the points $\Pi_2 = \{P[i-1], P[i-2], \ldots, p[1]\}$ is the same.

Our algorithm considers a curtain whose two sides are bounded by the boundary of \mathcal{R} , and top boundary is attached to both p, q. The curtain falls in a manner parallel to the line λ . As soon as it hits a point $a \in \Pi_1$ it reports a PMER. This point is easily obtained from the sorted list Π_1 . If the projection a^* of the point aon λ lies inside the interval [p, q], the processing of λ stops. Otherwise, the curtain is truncated at a^* , and the process continues to process the next point in Π_1 .

2.2 Complexity analysis

We have considered all the $O(n^2)$ vertices of $\mathcal{A}(P)$. Generation of each vertex v needs O(n) time with O(1) additional space. The time required for processing the vertex v for computing all the PMERs with one side passing through the pair of points (p,q) corresponding to the vertex v is also O(n), and it needs O(1) extra work-space. The algorithm needs to maintain a global counter to store the maximum area/perimeter PMER.

Theorem 1 Given an array with n points, the maximum area/perimeter rectangle of arbitrary orientation can be computed in $O(n^3)$ time with O(1) extra space.

Corollary 1.1 The method proposed in $\operatorname{process}(p,q)$ can also be used to compute the largest empty axisparallel rectangle in $O(n^2)$ time.

Proof. For computing the largest empty axis-parallel rectangle, we need not have to consider the duals of the points in P. Here for each point $p_i \in P$, we need to execute four line sweep passes as follows:

• Sweep a horizontal line upwards (resp. downwards) to get the largest axis-parallel MER with bottom (resp. top) boundary passing through p_i , and

• Sweep a vertical line towards left (resp. right) to get the largest axis-parallel MER with right (resp. left) boundary passing through p_i .

To execute the horizontal (resp. vertical) line sweep for all the points, we need to sort the points in P with respect to their y-coordinates (resp. x-coordinates) once only. Then the time complexity of the line sweep for each point $p_i \in P$ is O(n).

3 Computing largest axis-parallel MEC

We now describe the method of computing the largest empty cuboid among a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in a 3D axis-parallel parallelopiped (cuboid) \mathcal{R} bounded by six axis-parallel planes. The coordinate of the point p_i is denoted by (x_i, y_i, z_i) . A maximal empty cuboid (MEC) is a cuboid whose each face either coincides with a face of \mathcal{R} or passes through a point in P, and its interior does not contain any point in P. The objective is to identify an MEC of maximum volume. There are three types of MECs' inside \mathcal{R} .

- **type-1:** the MEC with both top and bottom faces aligned with the top and bottom faces of \mathcal{R} ,
- **type-2:** the MEC whose top face is aligned with the top face of \mathcal{R} , but bottom face passes through a point in P, and
- **type-3:** the MEC whose top face passes through some point in P. Bottom face may pass through a point in P or may coincide with the bottom face of \mathcal{R} .

Theorem 2 [12] The number of type-1, type-2 and type-3 MECs' inside \mathcal{R} are $O(n^2)$, $O(n^2)$ and $O(n^3)$ respectively in the worst case.

From now onwards, we use P to denote the array of size n containing the input points. We show that the methods proposed in [12] for identifying the largest *type-1*, *type-2* and *type-3* MECs can be made in-place with O(1) extra work-space in addition to the input array.

3.1 Computation of largest *type-1* MEC

Consider the projections of the points in P on the top face H of \mathcal{R} . Note that, each maximal empty axisparallel rectangle (MER) on H corresponds to a *type-1* MEC. Since the height of all these MECs' are the same, the problem reduces to computing the maximum area MER in H. Corollary 1.1 suggests the following result:

Lemma 3 The largest empty type-1 MEC can be computed in $O(n^2)$ time using O(1) extra work-space.

3.2 Computation of largest type-2 MEC

We assume that the points in P are sorted in decreasing order of their z-coordinates. We consider each point $p_i \in P$ in order, and compute $MEC(p_i)$, the largest type-2 MEC whose bottom face passes through p_i . Let $H(p_i)$ be the horizontal plane passing through p_i , and



Figure 1: Empty orthoconvex polygon around p_i

 $P_i = \{p_1, p_2, \dots, p_k\}$ be the set of points strictly above $H(p_i)$. Note that, $MEC(p_i)$ corresponds to the largest MER on $H(p_i)$ containing the point p_i among the projections of the points in P_i on $H(p_i)$ as obstacles.

Let us partition the plane $H(p_i)$ into four quadrants by drawing two mutually perpendicular axis-parallel lines passing through p_i . In O(n) time, we will be able to partition the portion of the array P[1, 2, ..., k] into four parts, namely P_i^{θ} , $\theta = 1, 2, 3, 4$, where P_i^{θ} denote the points in the θ -th quadrant. The members in P_i^{θ} are in consecutive positions in the array P.

In each quadrant θ , we define the maximal closest stair $STAIR_{\theta}$ around p_i with a subset of points of P_i^{θ} as in [12]. $STAIR_{\theta}$ is unique in the θ -th quadrant. The concatenation of these four stairs describe an empty axisparallel orthoconvex polygon OP (see Figure 1 for illustration). The problem of locating the largest type-2MEC with p_i on its bottom face reduces to finding the largest MER inside OP containing the point p_i . We explain the method of computing $STAIR_1$. The other stairs are computed in a similar manner. Next, we explain the method of computing $MER(p_i)$ in OP.

3.2.1 Computation of *STAIR*₁

We sort the points in P_i^1 in increasing order of their y-coordinates. Now, sweep a line parallel to the x-axis on $H(p_i)$ to identify $STAIR_1$. The points in $STAIR_1$ are maintained at the beginning of the array P_i^1 , and the points in P_i^1 that are not in $STAIR_1$, are stored at the end of P_i^1 . The points in $STAIR_1$ are stored in decreasing order of their x-coordinates. Two index variables α and β are maintained during the execution; α indicates the index of the point in P_i^1 under processing, and β indicates the index of the last point in $STAIR_1$ (i.e., having minimum x-coordinate among the ones identified so far). During the sweep, if $p_{\alpha} = (x_{\alpha}, y_{\alpha}, z_{\alpha}) \in P_i^1$ satisfies $x_{\alpha} > x_{\beta}$, then p_{α} does not appear on $STAIR_1$. However, if $x_{\alpha} < x_{\beta}$, then p_{α} appears in *STAIR*₁. In such a case, if $\alpha = \beta + 1$, then both α and β are incremented by 1. But, if $\alpha > \beta + 1$, then (i) β is incremented, (ii) $P_i^1[\alpha]$ and $P_i^1[\beta]$ are swapped, and (iii) α is incremented to process the next point of P_i^1 .

3.2.2 Computation of $MER(p_i)$

It is easy to observe that, for every MER inside the orthoconvex polygon OP, its north side will contain a point in $STAIR_1 \cup STAIR_2$, and its south side will contain a point $STAIR_3 \cup STAIR_4$. In our algorithm for computing $MER(p_i)$, we will consider each point in $STAIR_1 \cup STAIR_2$, and compute all the MERs with north side passing through it.

The MERs with north side touching a point $p_j \in STAIR_1$ are obtained as follows. We draw the projections q_1 and q_2 of p_j on $STAIR_2$ and $STAIR_4$ respectively as shown in Figure 2. Let q_1 lies on the vertical line passing through $p_{\alpha} \in STAIR_2$ and q_2 lies on the horizontal line passing through $p_{\beta} \in STAIR_4$. Thus, p_{α} satisfies $y(q_1) \in [y(p_{\alpha'}), y(p_{\alpha})]$, where p_{α} and $p_{\alpha'}$ are two consecutive points on $STAIR_2$. Thus, q_1 can be obtained by performing binary search in $STAIR_2$. Similarly, q_2 can be obtained by performing binary search in $STAIR_4$. Now, we compute the projections of q_1 and q_2 on $STAIR_3$. Let these two points be q_3 and q_4 respectively. Now, two situations may arise:

 $[\mathbf{y}(\mathbf{q}_3) \leq \mathbf{y}(\mathbf{q}_4):]$ Here only one MER with p_j on its north boundary is possible. Its west and south sides will contain p_{α} and p_{β} respectively; its east side will contain a point $p'_j \in STAIR_1$ adjacent to p_j to the right side $(y(p'_j) < y(p_j))$ or a point $q \in STAIR_4$ adjacent to q_2 to the right side $(y(q) > y(q_2))$. See Figure 2(a).

 $[\mathbf{y}(\mathbf{q_3}) > \mathbf{y}(\mathbf{q_4}):]$ Here more than one MER with p_j on its north boundary may exist (Figure 2(b)). Let $\mu_1, \mu_2, \ldots, \mu_m$ be the consecutive points in $STAIR_3$ with $y(\mu_1) < y(\mu_2) < \ldots < y(\mu_m)$, and $y(\mu_r) \in [y(q_3), y(q_4)]$ for $r = 1, 2, \ldots, m$. Similarly, $\nu_1, \nu_2, \ldots, \nu_\ell$ are consecutive points in $STAIR_4$ with $y(\nu_1) < y(\nu_2) < \ldots < y(\nu_\ell)$, and $y(\nu_r) \in [y(q_3), y(q_4]$ for $r = 1, 2, \ldots, \ell$. Here, $\ell + m + 1$ MERs are possible with north boundary passing through p_j . Their south boundaries will pass through $q_2, \mu_1, \mu_2, \ldots, \mu_k, \nu_1, \nu_2, \ldots, \nu_\ell$ respectively. The east and west sides of these MERs are uniquely defined, and are obtained by traversing the stairs in four quadrants.

Lemma 4 The time complexity of computing the largest type-2 MEC is $O(n^2 \log n)$ with O(1) extra space.

Proof. We prove this lemma by showing that the time complexity of generating all the *type-2* MECs with p_i on its bottom face is $O(C_i + n \log n)$; C_i is the number of such MECs present in \mathcal{R} . Processing of the point p_i consists of the following three steps:

[Step 1:] Partitioning the points above p_i into P_i^{θ} , for $\theta = 1, 2, 3, 4$. This needs O(n) time in the worst case.

[Step 2:] Computing $STAIR_{\theta}$, $\theta = 1, 2, 3, 4$. This needs $O(n \log n)$ time since a sorting step among the points



Figure 2: Computation of $MER(p_i)$

in P_i^{θ} with respect to their *y*-coordinates is involved here. After the sorting, the line sweep for constructing $STAIR_{\theta}$ needs O(n) time.

[Step 3:] Computing $MER(p_i)$. This needs $O(C_i + n \log n)$ time. The second component in the time complexity appears due to the fact that for each point $p_j \in STAIR_1 \cup STAIR_2$, we need to execute binary searches for computing its projections q_1 and q_2 in the adjacent stairs. Again we may need two binary searches to get the set of feasible points in $STAIR_3$ that may appear in the south boundary of the generated MERs.

Since (i) we need to process all the points $p_i \in P$, (ii) $\mathcal{C} = \sum_{i=1}^{n} \mathcal{C}_i$, and (iii) $|\mathcal{C}| = O(n^2)$ in the worst case (see Theorem 2), the time complexity result follows.

We have used four integer locations n_1, n_2, n_3, n_4 , six index variables $q_1, q_2, q_3, q_4, \alpha, \beta$, and a space for swap operation. Thus, the space complexity follows. \Box

3.3 Computation of largest type-3 MEC

Here we describe the method of generating all the *type-*3 MECs with top face passing through a point $p_i \in P$. Let the points in P be in decreasing order of their *z*coordinates. Consider the horizontal plane $H(p_i)$ passing through p_i and sweep it downwards. When the sweeping plane hits a point $p_j \in P$, the points inside the two horizontal planes $H(p_i)$ and $H(p_j)$ will participate in computing the MECs with top and bottom faces passing through p_i and p_j respectively.

As in Subsection 3.2.1, here also we use P_i^{θ} to denote the subset of points in P that lie in θ -th quadrant, $\theta = 1, 2, 3, 4$, determined by the horizontal and vertical lines through the point p_i on $H(p_i)$. The points in $\bigcup_{\theta=1}^{4} P_i^{\theta}$ are stored in the array-positions $P[i+1], P[i+2], \ldots, P[n]$. The members in P_i^{θ} are in the consecutive locations of the array P in decreasing order of their z-coordinates. We maintain four integer variables n_{θ} and four index variables $\chi_{\theta}, \theta = 1, 2, 3, 4$. n_{θ} denotes $|P_i^{\theta}|$ and χ_{θ} indicates the last point hit by the sweeping plane in the θ -th quadrant. At an instant of time the point hit by the sweeping plane is

obtained by comparing the z-coordinates of the points $\{P[\chi_{\theta}+1], \theta=1, 2, 3, 4\}.$

Let the point p_j be under process. The empty orthoconvex polygon OP around the point p_i is determined by four stairs { $STAIR_{\theta}, \theta = 1, 2, 3, 4$ } using the points lying inside the horizontal slab bounded by $H(p_i)$ and $H(p_j)$ (but not including p_i and p_j). The points determining $STAIR_{\theta}$ are stored at the begining of the subarray P_i^{θ} in order of their y-coordinates.

In order to compute the largest MEC with top and bottom faces passing through p_i and p_j respectively, we need to compute $MER(p_i, p_j)$, the largest MER in the orthoonvex polygon OP that contains both p_i and projection p'_j of p_j on $H(p_i)$. Here, the following two tasks need to be performed: (i) Computing all the MERs in OP that contains both p_i and p'_j , and (ii) updating OPby inserting p'_j for processing the next point p_{j+1} .

3.3.1 Computing $MER(p_i, p_j)$

Without loss of generality, assume that p'_j is in the first quadrant. If p'_j is in some other quadrant, the situation is similarly tackled. We now determine the subset of points in $STAIR_1 \cup STAIR_2$ that can appear in the north boundary of an MER containing both p_i and p'_j .

Let $STAIR_1 = \{q_k, k = 1, 2..., m\} \subseteq P_i^1$, and the points in $Q = \{q_{\alpha}, q_{\alpha+1}, \ldots, q_{\beta}\} \subseteq STAIR_1$ satisfy $x(q_k) > x(p_i)$ and $y(q_k) > y(p_i)$. All the MERs in *OP* with north boundary passing through q_k , k = $\alpha, \alpha + 1, \dots, \beta + 1$ and containing p_i in its proper interior will contain p'_i also. We draw the projections of p'_i and q_{β} on *STAIR*₂. Let these two points be μ and ν respectively. If $x(\mu) = x(\nu)$, then no point on *STAIR*₂ can appear on the north boundary of a desired MER. But if $x(\mu) < x(\nu)$, then all the points $q' \in STAIR_2$ satisfying $x(\mu) < x(q') < x(\nu)$ can appear on the north boundary of a desired MER. In Figure 3, the set of points that can appear on the north boundary of an MER are marked with empty dots. The method of computing an MER with a point $q_k \in STAIR_1 \cup STAIR_2$ on its north boundary is same as that in Subsection 3.2.2.



Figure 3: Computation of type-3 MEC

3.3.2 Updating OP

After computing the set of MERs in *OP* containing p_i and p'_i in its interior, we update *OP* by inserting p'_i in the respective stair. We have already assumed that p'_i lies in the first quadrant, and each member $q_k \in Q$ satisfies $x(q_k) > x(p_i)$ and $y(q_k) > y(p_i)$. In order to insert p'_i in STAIR₁, we need to remove the members in Q from $STAIR_1$. We maintain two index variables α and β ; α indicates the last point of *STAIR*₁ observed so far, and β indicates the point p_j under consideration in P_i^1 , $\alpha \leq \beta - 1$. If $\alpha < \beta - 1$, then the points in the positions $\alpha + 1, \dots, \beta - 1$ of P_i^1 are already considered, but their projections are not present in $STAIR_1$. While inserting p'_{i} in STAIR₁, we place p_{j} in its desired location as follows: (i) swap $P[\beta+1]$ and $P[\alpha]$, and then (ii) execute a sequence of swap swap(P[r], P[r-1]) starting from $r = \beta + 1$ until a point $P[r] \in STAIR_1$ is found such that y(P[r]) < y(P[r-1]). Now, if |Q| > 0, then we remove the members in Q using two index variables rand s. We start with $r = \gamma + 1$ and $s = \gamma + |Q| + 1$. At each step, we execute $\operatorname{swap}(P[r], P[s])$ and increment r and s by 1 until $s = \beta$. This needs $O(\max(|Q|, (\beta - \gamma)))$ time which may be $O(|P_i^1|)$ in the worst case.

After computing the largest *type-3* MEC with p_i on its top boundary, we need to sort the points again with respect to their z-coordinates. This is required for processing p_{i+1} . Thus we have the following result:

Lemma 5 The time required for processing p_i is $O(n^2 + C'_i)$ in the worst case, where C'_i is the number of type-3 MECs with p_i on its top boundary.

Proof. The time required for computing $MER(p_i, p_j)$ may be $O(|P_{ij}| + C_{ij})$, where P_{ij} denotes the number of points inside the horizontal slab bounded by $H(p_i)$ and $H(p_j)$, and C_{ij} denotes the number of MERs containing both p_i and p'_j inside OP with the projection of points P_{ij} on $H(p_i)$. In order to compute the largest type-3 MEC with p_i on its top boundary, we need to compute $MER(p_i, p_j)$ for all j > i, $C'_i = \sum_{j=i+1}^n C_{ij}$, and $\sum_{j=i+1}^n |P_{ij}| = O((n-i)^2)$. Finally after the processing of p_i , the sorting step takes $O(n \log n)$ time. \Box **Theorem 6** The worst case time complexity of our inplace algorithm for computing the largest MEC is $O(n^3)$, and it takes O(1) extra space.

Proof. The time complexity for computing the largest type-1 MEC is $O(n^2)$ (see Corollary 1.1). Lemma 4 and the fact that the number of type-2 MECs is $O(n^2)$ in the worst case [12], indicate that the worst case time complexity of computing the largest type-2 MEC is alo $O(n^2 \log n)$. Finally, Lemma 5 says that the worst case time complexity of computing the largest type-3 MEC is $O(n^3)$. Needless to mention that we have used only few index variables, four integer variables to maintain the number of points in the four quadrants on $H(p_i)$, and a temporary variable for the swap operation.

References

- A. Aggarwal and S. Suri. Fast algorithm for computing the largest empty rectangle. In Symp. on Comput. Geom., pages 278-290, 1987.
- [2] T. Asano and G. Rote. Constant working-space algorithms for geometric problems. In *Canad. Conf. on Comput. Geom.*, pages 87-90, 2009.
- [3] P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. H. M. Smid and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geome*try, 37(3):209-227, 2007.
- [4] H. Brönnimann, T. M. Chan and E. Y. Chen. Towards in-place geometric algorithms and data structures. In *Symp. on Comput. Geom.*, pages 239-246, 2004.
- [5] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison and G. T. Toussaint. Space-efficient planar convex hull algorithms. *Theoretical Computer Science*, 321(1):25-40, 2004.
- [6] J. Chaudhuri, S. C. Nandy and S. Das. Largest empty rectangle among a point set. J. Algorithms, 46(1):54-78, 2003.
- [7] E. Y. Chen and T. M. Chan. A space-efficient algorithm for segment intersection. In *Canad. Conf. on Comput. Geom.*, pages 68-71, 2003.
- [8] T. M. Chan, E. Y. Chen. Optimal in-place algorithms for 3-D convex hulls and 2-D segment intersection. In ACM Symp. on Comput. Geom., pages 80-87, 2009.
- [9] M. De, A. Maheswari, S. C. Nandy and M.Smid. An inplace min-max priority search tree. *Manuscript*, 2011.
- [10] H. Edelsbrunner. Algorithms in Combinatorial Geometry, Springer, Berlin, 1987.
- [11] R. Janardan and F. P. Preparata. Widest corridor problem. Nordic J. Computing. 1(2):231-245, 1994.
- [12] S. C. Nandy and B. B. Bhattacharya. Maximal empty cuboids among points and blocks. *Computers and mathematics with Applications*, 36(3):11-20, 1998.
- [13] A. Naamad, D. T. Lee and W. -L. Hsu. On the maximum empty rectangle problem. *Discrete Applied Mathematics*, 8(3):267-277, 1984.

- [14] M. Orlowski. A new algorithm for the largest empty rectangle problem. Algorithmica, 5(1-4):65-73, 1990.
- [15] J. Vahrenhold. An in-place algorithm for Klee's measure problem in two dimensions. *Information Process*ing Letters, 102(4):169-174, 2007.

Realizing Site Permutations*

Stephane Durocher[†]

Saeed Mehrabi[†]

Debajyoti Mondal[†]

Matthew Skala[†]

Abstract

Given n fixed sites on the plane, there are several ways to determine a permutation of the sites as a function of a unit vector u or a vantage point v. Given such a scheme and a permutation π , we can ask whether there is any unit vector or vantage point for which the permutation is π . We give linear-time algorithms for this realization problem under three schemes for determining permutations: sweeping a line across the sites in a direction u; expanding a circle starting from a vantage point v; and sweeping a ray from v to give a cyclic permutation.

1 Introduction

Given an arrangement of points called *sites* on the plane, there are several ways to choose a permutation of the sites. For instance, we could sweep a line across the arrangement and enumerate the sites in the order the line touches them. We could start from some vantage point and consider the sites in order of increasing distance from the vantage point. We could instead sweep a ray from the vantage point radially through all possible angles and consider the circular ordering of the sites it encounters. Other rules are also possible. Given a set of sites S and a geometric rule for defining a permutation of S as a function of a sweep direction or vantage point, some permutations can be realized by some choice of sweep direction or vantage point, and other permutations cannot be realized. In this work we consider the algorithmic problem of recognizing realizable permutations, and describe linear-time algorithms for this problem under three different geometric rules.

Problems of this type have applications in settings that involve computing the position of an observer such as a robot [8] within its environment relative to a sequence of observations made using a directional sensor (such as a sonar, radar, or camera).

2 Definitions and notation

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of points on the Euclidean plane, called the *sites*. Let \mathbb{S}^1 represent the set of directions, or unit vectors, in the plane. Assume that

all points and directions in the problem are in general position: that is, no two points are coincident; no three points are collinear; no point is equidistant from two others; no four points w, x, y, and z have the relationship that the line wx is parallel to the line yz; and (in the case of sweep-line permutations) the given sweep direction u is not orthogonal to the line connecting any two points.

For any unit vector $u \in \mathbb{S}^1$ in general position relative to S, let the *sweep-line permutation* of u be the permutation of sites determined by sweeping a line orthogonal to u across the sites in the direction u and enumerating the sites in the order encountered. It would be equivalent to say that we project all the sites onto a directed line parallel to u and define the permutation by the order of the projected sites along the line.

Instead of sweeping a line in a direction, we might start from a point v called the vantage point and enumerate the sites in order of increasing distance from vto form a *distance permutation*. This derivation can be imagined as expanding a circle centred on v and enumerating the sites in the order encountered; or as sending out a sonar ping and recording the order of the echoes received.

Another way of determining a permutation would be by taking a ray starting from v and sweeping it counterclockwise through a complete rotation of 360° , enumerating the sites in the order the ray encounters them.¹ Then we obtain a cyclic permutation; that is, an equivalence class of permutations up to rotation. This *radial permutation* is analogous to scanning the sites with a rotating search light or radar beam, and recording the order in which we see them without regard for the angles or the starting orientation of the sweep.

Figure 1 illustrates the three kinds of site permutations we consider. In the figure, a line swept in the direction u encounters the sites in the order dcba. An expanding circle starting at v encounters the sites in the order bdac; and a ray originating at v and swept counterclockwise encounters them in the order cdba, up to a rotation that depends on the starting orientation of the sweep. For any of these schemes, given a permutation or cyclic permutation π and a set of sites, a unit vector u or vantage point v is said to *realize* π if π is the permutation determined by u or v for the given

^{*}Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]Department of Computer Science, University of Manitoba, {durocher,mehrabi,jyoti,mskala}@cs.umanitoba.ca

¹We describe angles using degrees to avoid confusion with the symbol π used for a generic permutation.



Figure 1: A. sweep-line permutation in direction u: dcba. B. distance permutation centred at v: bdac. C. radial permutation centred at v: cdba.

scheme and sites. Then π is said to be *realizable* if and only if there is a vantage point or unit vector realizing it. In this work we consider the problem of deciding whether a permutation π is realizable and, if so, computing a corresponding unit vector u or vantage point vthat realizes π .

If the vantage point v is sufficiently far from the sites in the direction opposite to u, then the expanding circle centred on v when it passes over the sites is equivalent to a line orthogonal to u and sweeping in the direction u. Similarly, if the vantage point v is sufficiently far from the sites in a direction 90° counterclockwise from u, then the sweeping ray from v when it passes over the sites is equivalent a line sweeping in the direction u. We can thus make the following observation.

Observation 1 Every sweep-line permutation for an arrangement of sites is also realized as a distance permutation and a radial permutation.

Throughout our algorithmic results we assume a real RAM model of computation, in which we can perform basic arithmetic operations in unit time. This is a standard assumption for computational geometry algorithms in general; and in particular, the linear-time linear programming algorithm of Megiddo [5], which we use, is only linear-time under the assumption it can complete in constant time the multiplication and division operations needed to find the intersections of lines given as input. Analysing the algorithms under some other model to force a superlinear result would be primarily an exploration of the complexity of arithmetic in general without giving specific insight into these algorithms.

3 Previous work

The cyclic sequence of sweep-line permutations formed by a site arrangement as we rotate the sweep direction through a full circle is called an *allowable sequence*, and allowable sequences are well-studied. Goodman and Pollack pioneered the use of allowable sequences in characterizing the order type of the sites [4]. The allowable sequence for a site arrangement is closely connected to the oriented matroid associated with the site arrangement, and that connection leads to many combinatorial insights [2].

Chávez, Figueroa, and Navarro introduced distance permutations in a database context, as a way of classifying points in high-dimensional general metric spaces to support efficient proximity queries [3]. Note that this kind of permutation (possibly with a tiebreaking assumption added to handle degenerate cases) is defined for any space with a real distance function—it need not even be a metric. Skala proved bounds on the number of distinct distance permutations that can occur as a function of the number of sites in various spaces, including an exact count for Euclidean spaces [6].

Bieri and Schmidt studied radial permutations as well as sweep-line permutations and a variation on radial permutations in which a line is swept instead of a ray [1]. Noting that the number of radial permutations realized by a site arrangement is $\Theta(n^4)$ (which follows from the number of bisectors and the fact that k lines in general position on the plane divide the plane into $\Theta(k^2)$ cells), they give an algorithm to generate all the permutations in $\Theta(n^4)$ time—interesting because the naive size of the output would be $\Theta(n^5)$. To achieve the faster running time, they order the permutations in such a way that each except the first differs from some previous permutation by one swap of adjacent elements; then the swaps can be found in $O(n^4)$ time. Tovar, Freda, and LaValle studied radial permutations in the context of robot navigation; assuming a robot with a sensor that detects the radial permutation of landmarks as seen from its current location, they show how the robot can achieve navigational goals [8].

4 Bisectors and Voronoi diagrams

The sweep-line method of finding a permutation implicitly divides the set of possible directions into intervals corresponding to the realizable permutations. Similarly, the distance and radial permutations correspond to cells of a Voronoi-like diagram in the plane. These divisions are shown in Figure 2. Note that the unbounded cells for distance and radial permutations correspond to the permutations realized by points at infinity, and thus to the sweep-line permutations (Observation 1).

Every pair of sites s_i and s_j determines a *bisector*: a set of points where the ordering of s_i and s_j is not



Figure 2: Division of space by permutation schemes: (a) sweep-line, (b) distance, (c) radial.

uniquely defined. If we imagine a point wandering continuously through the space (like Tovar, Freda, and LaValle's robot [8]), the permutation it observes will change by a swap of adjacent elements each time it crosses a bisector. For sweep-line permutations, the bisector of s_i and s_j consists of the two unit vectors parallel to the line between s_i and s_j . For distance permutations, it is the set of all points equidistant from s_i and s_j , which is the line orthogonally bisecting the segment that connects the two sites. For radial permutations, it is the line connecting s_i and s_j , with the segment between them removed. The radial bisector is unusual because it can be said to cut the plane into just one piece: with two sites, only one permutation exists up to rotation, so there is only one cell. Radial bisectors become more meaningful once there are three or more sites.

Examination of these divisions of space leads to simple counts or bounds on the number of permutations realized. For sweep-line permutations, the bisectors each consist of two points, and distinct bisectors never coincide when sites are in general position, so it is trivial that the number of intervals and thus permutations for *n* sites is $2\binom{n}{2}$. For distance permutations, $\binom{n}{2}$ bisectors and the quadratic bound on number of cells formed by lines in general position gives an upper bound of $O(n^4)$ permutations; Skala notes that bisectors are not in general position because of transitivity, and gives an exact recurrence for the number of permutations, as well as generalizing the question to higher dimensions of Euclidean space; in d dimensions the number of permutations is shown to be $\Theta(n^{2d})$ [6]. For radial permutations, the same kind of argument gives an obvious $O(n^4)$ upper bound, but the possibility for a permutation's cell to be non-convex or even disconnected (as in Figure 2(c)) complicates matters. Bieri and Schmidt state as a theorem (without detailed proof) that the upper bound is achieved by some arrangement of n sites for every n [1].

5 Radial permutations in the dual space

For each site s_i in s_1, s_2, \ldots, s_n , define a line s_i^* as follows: let (x_i, y_i) be the coordinates of s_i , and then let s_i^* be the line dual to s_i , defined by $y = x \cdot x_i - y_i$.

Let $v = x_v, y_v$ be a point in the plane, not equal to any of the sites, and similarly define its dual line v^* by $y = x \cdot x_v - y_v$. These points and lines are shown in Figure 3. The vantage point v was chosen to be the origin for convenience in making and understanding the figure; its image in dual space is the x axis. The sorted sequence of the segments (v, s_i) around v corresponds to an ordered sequence of intersections between the lines v^* and s_i^* , as a line connecting two points in primal space corresponds to the intersection of two lines in dual space.

Let L be the vertical line passing through v. L divides the plane into two half-planes. In Figure 3(a), the right half-plane contains s_3 , s_5 , and s_6 , and the left half-plane contains s_1 , s_2 , and s_4 . In Figure 3(b), the crossing points for s_3^* , s_5^* , and s_6^* (shown in white) appear consecutively right to left. Similarly, the crossing points for s_1^* , s_2^* , and s_4^* (shown in black) appear consecutively left to right. We can concatenate the two lists to obtain a radial permutation π of the sites around v: $s_1, s_2, s_4, s_3, s_5, s_6$.

The dual space naturally suggests another sequence of the sites, that found by examining all the crossings (not black and white separately) along the line. From right to left that sequence is $s_1^*, s_2^*, s_3^*, s_5^*, s_4^*, s_6^*$. In the primal space it corresponds to rotating a line, not a ray, passing through v, starting at vertical and then 180° counterclockwise until it becomes vertical again, and enumerating the sites in the order the line encounters them. This variation of radial permutations corresponds to the *undirected stars* described by Streinu [7]. If we add a sign to each element in the sequence describing whether it was hit by the head or the tail of the line during the radial sweep, the result is a *directed star* (or simply a *star*) as described by Streinu; in Figure 3, using Streinu's notation, the star would be $12\overline{3}\overline{5}4\overline{6}$, where x and \bar{x} denote that element x was met by the head or tail end, respectively, of the rotating line. Given a directed star, it is straightforward to construct the corresponding radial permutation in linear time.



Figure 3: A radial permutation in (a) primal and (b) dual spaces.

6 Results

The bisectors for sweep-line permutations suggest a simple linear-time algorithm for realizing permutations; in fact, because the cells are simply intervals around the circle, we not only compute a single unit vector to realize the permutation, but also completely describe the set of all such vectors in the same asymptotic time.

Theorem 1 There exists a linear-time algorithm that given sites s_1, s_2, \ldots, s_n and a permutation π on the indices, finds the set of all directions u for which the sweep-line permutation is π .

Proof. By transitivity, it suffices to enforce the n-1 constraints that the sweep line reaches $s_{\pi(1)}$ before $s_{\pi(2)}$, $s_{\pi(2)}$ before $s_{\pi(3)}$, and so on. Each of those constraints corresponds to an interval of allowed values for u in \mathbb{S}^1 . Each interval is open and has length 180°; therefore the intersection of any two of them is a single, possibly smaller, interval; and by associativity we can compute the intersection of all of them in O(n) time.

In the case of distance permutations, linear time does not allow us to examine all of the quadratic number of bisectors; but because the bisectors correspond to the transitive "less than" relation on distances, we can obtain all the necessary information by examining a linearsized subset of them.

Theorem 2 There exists a linear-time algorithm that given sites s_1, s_2, \ldots, s_n and a permutation π on the indices, finds a vantage point v for which the distance permutation is π , if such a v exists.

Proof. By transitivity, it suffices to enforce the n-1 constraints that v is closer to $s_{\pi(1)}$ than to $s_{\pi(2)}$, closer to $s_{\pi(2)}$ than to $s_{\pi(3)}$, and so on. Each of those corresponds to a half-plane (linear) constraint. By the linear-time two-dimensional linear programming algorithm of Megiddo [5], we can find a point v satisfying all the constraints in O(n) time.

Radial permutations present a greater challenge, primarily because we are seeking not a single permutation



Figure 4: Angles measured around v.

but an equivalence class of permutations. We begin by proving a connection between the realization problem and linear programming.

Lemma 3 There exists a linear-time algorithm that given sites s_1, s_2, \ldots, s_n and a permutation π on the indices, constructs a set of linear constraints such that any vantage point v for which the radial permutation is π up to rotation satisfies all, or all but one, of the constraints.

Proof. Where v is the vantage point, for each integer $1 \leq i \leq n$, let θ_i denote the angle measured counterclockwise around v from the ray pointing at $s_{\pi(i)}$ to the ray pointing at $s_{\pi(j)}$, where $j = (i \mod n) + 1$, as shown in Figure 4. Let $\Theta = \sum_{i=1}^{n} \theta_i$, that is, the sum of all the θ_i .

For each pair of successive sites $s_{\pi(i)}$ and $s_{\pi(j)}$ where $\theta_i < 180^\circ$, it must be that $v, s_{\pi(i)}$, and $s_{\pi(j)}$ form a triangle in counterclockwise order, like $v, s_{\pi(1)}$, and $s_{\pi(2)}$ in Figure 4. That is equivalent to the statement that v is on the left side of the directed line from $s_{\pi(i)}$ to $s_{\pi(j)}$, and we can express that statement as a half-plane constraint. We create such a constraint for each pair of successive sites.

One way v might realize π would be if it were in the kernel of a star-shaped polygon formed by the sites in the order described by π ; then every $\theta_i < 180^\circ$ and all the linear constraints would be satisfied. This situation is illustrated in Figure 5(a).



Figure 5: Realizing a permutation while violating (a) zero or (b) one of the linear constraints (π is the identity).

It is also possible for the vantage point v to lie outside the kernel of the star-shaped polygon, as shown in Figure 5(b). However, if v realizes π , then summing all the θ_i corresponds to making one full sweep around v; $\Theta = 360^{\circ}$. Thus, at most one of the θ_i can be greater than 180°, corresponding to a violated constraint; all the others must be satisfied. Therefore, at most one of the constraints can be violated.

To actually solve the realization problem we must not only perform linear programming but also determine which constraint to violate, if any. Here we exploit the special properties of Megiddo's linear programming algorithm [5], which either finds a solution to the realization problem immediately, or gives us a clue to where the permutation must start.

Theorem 4 There exists a linear-time algorithm that given sites s_1, s_2, \ldots, s_n and a permutation π on the indices, finds a vantage point v for which the radial permutation is π up to rotation, if such a v exists.

Proof. We invoke the linear-time two-dimensional linear programming algorithm of Megiddo [5] to find a point satisfying all the constraints of Lemma 3, if possible. We can then distinguish two cases: (1) such a point exists; or (2) no such point exists.

Case 1. It is possible that a point v could satisfy all the constraints but not realize the permutation π , if the sequence of sites described by π winds more than once around v. An example demonstrating this situation is shown in Figure 6. When the linear program returns a solution v, it is easy to test in linear time whether v realizes the permutation π . If it does, the algorithm returns it immediately.

Suppose v does not realize π . The cumulative angle Θ must be an integer multiple of 360°; and when v is a solution to the linear program but does not realize the desired permutation, it must be at least 720°. Removing one constraint (changing the polygon to a path,

which might still be self-intersecting) reduces the sum for the remaining pairs of sites by strictly less than 360° , leaving it strictly greater than 360°. Now suppose we start our ray sweep with a ray pointing from v', a solution to the linear program with one constraint relaxed, to the site at the start of the path. If we sweep to each successive site on the path in turn, we will complete a full angle (360°) and see the start of the path again, before we complete the sweep at the end of the path. That means we must already have seen the end of the path, before its proper place at the end of the sweep. Therefore v' cannot realize the permutation π . In intuitive terms, if the polygon wraps more than once around some solution, it must wrap at least twice, and then the path formed by deleting one edge from the polygon (which subtracts less than 360°) must still wrap more than once around every solution.

We have that if there exists a vantage point v that is a solution to the linear program but does not realize the permutation π , then no point v' which is a solution to any linear program formed by relaxing one of the original constraints, can realize the permutation π . Since every point realizing π must be a solution to our original linear program with at most one constraint relaxed, then there can be no point realizing π at all. Thus, in Case 1, where the linear program is feasible, it suffices to test whether the solution v realizes π , return it if it does realize π , and return failure if it does not.

Case 2. If the first linear program is infeasible, then any v realizing π must be a solution to the linear program with exactly one constraint relaxed. Megiddo's algorithm [5] works by examining constant-sized subsets of the input constraints and, at each one, attempting to prove that at least one of the constraints is unnecessary for the optimal solution. His analysis shows that in each of the subsets at least one constraint can always be removed if the input is feasible, allowing the algorithm to stop after a linear number of steps with either a solution or a proof of infeasibility. That approach has the impor-



Figure 6: The sites wind more than once around v (π is the identity).



Figure 7: An infeasibility certificate.

tant consequence that in case of an infeasible input, the algorithm actually finds a constant-sized certificate of infeasibility, namely the last subset of input constraints it examined before halting. The algorithm can easily be modified to produce the certificate as output, in the form of at most three constraints that cannot all be satisfied. In general those constraints will be arranged as shown in Figure 7; with input not in general position a certificate consisting of two non-intersecting parallel half-planes would also be possible.

In order to be a vantage point realizing the desired radial permutation, v would have to satisfy all except at most one of the constraints in the original linear programming problem. If v can satisfy all except one, but not all of the constraints, then every infeasible subset of the constraints must include that one constraint, so it must be among the at most three returned when Megiddo's algorithm failed. By invoking Megiddo's algorithm at most three more times, with each constraint from the certificate removed in turn, we can find a value for v that realizes the permutation π , if any exists. \Box

7 Conclusion

In this paper, we considered the problem of realizing a permutation π on a set of n sites in the plane. We gave three linear-time algorithms for this kind of problem, corresponding to three schemes of determining permutations: sweeping a line in direction u, measuring distance from a vantage point v, and sweeping a ray counterclockwise around v. One obvious direction for future work is to consider other ways of determining a permutation; for example, rotating a line through v instead of a ray starting at v. We might also consider more general kinds of constraint satisfaction involving site permutations; for instance, finding a point that realizes any permutation containing a given contiguous subsequence.

Acknowledgments

We wish to thank Pak Ching Li and Jason Morrison for insightful discussions on these and related problems.

References

- H. Bieri and P.-M. Schmidt. On the permutations generated by rotational sweeps of planar point sets. In Proceedings of the 8th Canadian Conference on Computational Geometry, Ottawa, Canada (CCCG 1996), pages 179–184, August 12–15 1996.
- [2] A. Björner, M. L. Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. *Oriented Matroids*. Cambridge University Press, 2nd edition, 1999.
- [3] E. Chávez, K. Figueroa, and G. Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *Proceedings of the 4th Mexican International Conference on Artificial Intelligence (MICAI* 2005), Monterrey, Mexico, volume 3789 of Lecture Notes in Computer Science, pages 405–414. Springer, November 14–18 2005.
- [4] J. E. Goodman. On the combinatorial classification of nondegenerate configurations in the plane. *Journal of Combinatorial Theory, Series A*, 29(2):220–235, September 1980.
- [5] N. Megiddo. Linear-time algorithms for linear programming in \mathbb{R}^3 and related problems. SIAM Journal on Computing, 12(4):759–776, November 1983.
- [6] M. Skala. Counting distance permutations. Journal of Discrete Algorithms, 7(1):49–61, March 2009.
- [7] I. Streinu. Clusters of stars. In Proceedings of the 13th Annual Symposium on Computational Geometry (SCG 1997), Nice, France, pages 439–441, June 4–6 1997.
- [8] B. Tovar, L. Freda, and S. M. LaValle. Learning combinatorial map information from permutations of landmarks. *International Journal of Robotics Research*, October 2010.
Establishing Strong Connectivity using Optimal Radius Half-Disk Antennas

Greg Aloupis*

Mirela Damian[†]

Robin Flatland[‡]

Özgür Özkan[¶]

David Rappaport[∥]

Stefanie Wuhrer**

Matias Korman[§]

Abstract

Given a set S of points in the plane representing wireless devices, each point equipped with a directional antenna of radius r and aperture angle $\alpha > 180^{\circ}$, our goal is to find orientations and a minimum r for these antennas such that the induced communication graph is strongly connected. We show that $r = \sqrt{3}$ suffices to establish strong connectivity, assuming that the longest edge in the Euclidean minimum spanning tree for S is 1. This result is optimal in the sense that $r = \sqrt{3}$ is necessary in the worst-case for $\alpha \in [180^\circ, 240^\circ)$. In contrast, r = 2is sometimes necessary when $\alpha < 180^{\circ}$.

1 Introduction

Consider a wireless network modeled by a set of planar point sites S each equipped with a transceiver having a transmission radius r. Typically one assumes that communication is omni-directional and two nodes can directly communicate with each other if the distance separating them is r or less. Geometrically the transmission region of an antenna at a point p is modeled by a circle of radius r centered at p. The connectivity of the network can be represented by a communication graph G(S), which has a node for each point and an edge between each pair of nodes separated by distance r or less.

Recently there has been interest in using directional antennas in place of their omni-directional counterparts [2, 3, 4, 5, 6, 7, 8]. Some advantages of using directional antennas are that security can be enhanced

School of Computing, Queen's University, Canada daver@cs.queensu.ca. Research supported by NSERC Discovery grant 388-329.

**Institute for Information Technology, National Research Council, Canada, stefanie.wuhrer@nrc-cnrc.gc.ca

and communication interference can be reduced. Furthermore, if directional antennas are used cleverly the power consumption of the network may be reduced. The transmission region of a directional antenna at a node p is geometrically represented by the sector of a circle with its apex at p, a central angle α , and a radius r. Its orientation is determined by a rotation θ about p. We assume that all antennas have the same α and r; it is only θ that varies. Thus communication between two nodes is no longer symmetric and is best modeled by a directed communication graph in which a directed edge \overrightarrow{pq} indicates that q lies in p's sector.

The direction assignment problem is the task of finding orientations for a set of directional antennas such that the induced communication graph has certain desired properties. In this paper we focus on obtaining a strongly connected communication graph using minimal r. We will assume S is normalized so that the length of the longest edge in a Euclidean minimum spanning tree is 1. It is not difficult to see that to achieve connectivity in the normalized point set, r must be at least 1. Caragiannis et al. [3] show that, for antennas with $\alpha < 240^{\circ}$, an increase in r by a factor of $\sqrt{3}$ is sometimes necessary to guarantee strong connectivity in the communication graph. We show here that, for $\alpha > 180^{\circ}$, an increase factor of $\sqrt{3}$ is always sufficient. In contrast, when $\alpha < 180$, the communication range must sometimes increase by a factor of 2 (i.e., consider points at unit intervals on a line).

We review some related results. In addition to providing lower bounds on r, Caragiannis *et al.* [3] also give an algorithm for orienting antennas with $180^{\circ} \leq \alpha < 288^{\circ}$ to obtain strong connectivity using $r = 2\sin(180^\circ \alpha/2$). Thus the algorithm presented here (with $r = \sqrt{3}$) improves upon their result when $180^{\circ} \leq \alpha < 240^{\circ}$. Damian and Flatland [6] consider directional antenna angles of 120° and 90°, and provide bounds of r = 5and r = 7 (resp.) while at the same time bounding the number of hops to 5 and 6 (resp.) for nodes within unit distance. Bose et al. [8] have recently shown that a connected network using omni-directional communication, can be replaced with directional antennas (with any $\alpha > 0^{\circ}$) so that the increase of r and hop distance are bounded by constant factors (which depend on α).

In other related work, Nijnatten [2] also considers the problem of finding suitable orientations of α -antennas

^{*}Université Libre de Bruxelles (ULB),Belgique, aloupis.greg@gmail.com

[†]Department of Computer Science, Villanova University, USA mirela.damian@villanova.edu

[‡]Department of Computer Science, Siena College, USA, flatland@siena.edu

[§]Université Bruxelles (ULB), Libre de Belgique. mkormanc@ulb.ac.be

[¶]Department of Computer Science and Engineering, Polytechnic Institute of NYU, USA, ozgurozkan@gmail.com. Research supported by US Department of Education grant P200A090157.

to form a strongly connected graph, but in his variant of the problem he allows a different r for each antenna and seeks to minimize the overall power consumption of the network. Ben-Moshe *et al.* [5] consider 90°-antennas but restrict the orientations to one of the four standard quadrant directions. Bhattacharya *et al.* [4] consider nodes with multiple directional antennas and focus on minimizing the sum of the antenna angles for a fixed r. Kranakis *et al.* [7] have recently published a survey of results pertaining to the use of directional antennas in wireless networks.

2 Orienting Antennas Using $r = \sqrt{3}$

Here we establish an upper bound of $\sqrt{3}$ for r, by means of an algorithm for orienting 180°-antennas of radius $r = \sqrt{3}$ to achieve a strongly connected communication graph. Let MST_5 be a minimum spanning tree of P with maximum degree of five, such as the one described in [1]. Our algorithm processes nodes in the order in which they are visited in a breath-first traversal of MST_5 . When a node is visited, it is assigned other nodes (within distance $\sqrt{3}$) for its antenna to cover (so as to satisfy certain invariants). If a node v is assigned to cover node w, we will say that "v points to w," and we use the notation $v \to w$. During the traversal of MST_5 , nodes are colored white, gray, or black. Initially all nodes are white, meaning that they have not yet been visited and do not point to any nodes. Visited nodes are black, and they point to at least one and at most two other gray or black nodes. Gray nodes are direct children of visited nodes but have not themselves been visited. They point to one other gray or black node.

Let the gray/black communication subgraph be the graph consisting of the gray/black nodes and having a directed edge \overrightarrow{uw} between each pair of nodes where $u \to w$. Our goal is to assign/adjust what the nodes point to by inserting/updating edges of length at most $\sqrt{3}$ in the gray/black communication subgraph such that, throughout the tree traversal, the gray/black communication subgraph is strongly-connected. For each gray/black node, observe that it is trivial to determine an orientation for its 180°-antenna that covers the one or two nodes it points to. We note that the full communication graph induced by these nodes may include additional edges, since a node's 180°-antenna may (by chance) cover nodes in addition to the one or two explicitly assigned to it, but these edges are not needed for strong connectivity.

Let the root of MST_5 be any node with degree one. To get started, we color the root node black and its child gray, and we constrain them to point to each other. Starting with the root's child, we visit the nodes one by one in a breadth-first search order. When a node v is vis-



Figure 1: Solid edges are MST_5 edges; the arrows represent directed edges in the communication graph; the dotted arrow in (b,e) represents v's directed edge to some other gray/black node (by Invariant (I2)).

ited, it is initially gray. During the visit, we change its color from gray to black and change the color of its children from white to gray. We then locally update/insert directed edges in the gray/black communication subgraph so that the following invariants are satisfied:

- (I1) Each black node points to at least one and at most two gray/black nodes.
- (I2) Each gray node points to exactly one gray/black node.
- (I3) For each gray node v, one of the following is true:
 - (I3a) v points to its parent p. (Fig. 1a)
 - (I3b) p points to v. (Fig. 1b)
 - (I3c) p has children s and d that are (resp.) the first child clockwise and first child counterclockwise from v, and $p \to s \to v \to d \to p$. In addition, $\widehat{spv} + \widehat{vpd} \leq 180^{\circ}$, and s and d lie on opposite sides of the line passing through v and p. (Fig. 1c)
- (I4) The gray/black communication subgraph has edges no longer than $\sqrt{3}$ and is strongly connected.

We describe inductively on the number of black nodes how to maintain these invariants. In the base case there is one black node, the root of MST_5 , and it points to its single gray child, which points back to the root. Observe that invariant (I1) holds for the root and invariants (I2, I3a) hold for its child. Also, observe that invariant (I4) holds for the root and its child. Assume inductively that the invariants hold after visiting and coloring *i* nodes black. Let *v* be the (i + 1)-st node visited.

We introduce some definitions so that we can process v in a uniform manner independent of its degree. Let **boundary**(v) be the children of v angularly adjacent to its

be the nodes adjacent to v in counter-clockwise order with $v_0 = p$. Then, if deg(v) = 1, let $boundary(v) = \emptyset$; otherwise, let boundary $(v) = \{v_1, v_{k-1}\}$ (e.g., see v_1 and v_3 in Fig. 1b.) An *isolated* child is a boundary child that is angularly separated from v's other children. In other words, if $\deg(v) < 3$, then $\operatorname{isolated}(v) = \emptyset$; otherwise if $\widehat{v_1vv_2} > 120^\circ$, then $v_1 \in \mathsf{isolated}(v)$, and similarly, if $v_{k-1}vv_{k-2} > 120^\circ$, then $v_{k-1} \in \mathsf{isolated}(v)$ (e.g., in Fig. 1b v_3 is isolated, but v_1 is not.) Let $\mathsf{Children}(v) =$ $\{v_1, \ldots, v_{k-1}\}$. The predicate in-range(v, w) = true iff $dist(v,w) \leq \sqrt{3}$. For any gray node v, let source(v) be the node constrained to point to node v, and let dest(v)be the node that v is constrained to point to. We use these terms only when well-defined within the context of Invariant I3. For instance, if v satisfies (I3a), then dest(v) = p; if v satisfies (I3b), then source(v) = p; and if v satisfies (I3c), then source(v) = s and dest(v) = d(see Fig. 1c).

Algorithm 1 details how we update/insert edges in the gray/black communication subgraph when visiting v. We begin by describing the operation of the algorithm. The IF statement in lines 1-13 initializes the variables v_{from} and v_{to} to be two nodes with a directed edge between them such that one of the two nodes is v. The existence of such an edge is guaranteed by Invariant (I3). For example, in Fig 1b, $v_{from} = p$ and $v_{to} = v$. In Fig. 1c, there are two directed edges incident to v, one of which will be used to initialize v_{from} and v_{to} ; in this case, there are no isolated children and we will assume v_4 is within range of d, so we set $v_{from} = v$ and $v_{to} = d$.

The remaining pseudocode (lines 14-20) first determines if there is a boundary child of v that is within distance $\sqrt{3}$ of both v_{from} and v_{to} . If so, then variable v_{via} is initialized to one such child, with preference being given in lines 17-18 to an isolated boundary child (which will be explained shortly). For example, in Fig. 1b, $v_{via} = v_3$; in Fig. 1c, $v_{via} = v_4$. Then the algorithm does two things. First, it replaces the edge $v_{from} \rightarrow v_{to}$ with the two edges, $v_{from} \rightarrow v_{via}$ and $v_{via} \rightarrow v_{to}$ (line 19). This incorporates child v_{via} into the strongly connected subgraph of gray/black nodes. Second, it calls the subroutine CHAIN (line 20) which inserts edges that link vand its children other than v_{via} into a cycle. This incorporates the other children into the strongly connected subgraph of gray/black nodes. See figure pairs 1b, 1e and 1c, 1f showing before and after edge insertions. If, however, there is no boundary child in range of v_{from} and v_{to} in line 15, then $v_{via} = \emptyset$ when the call to CHAIN in line 20 is made and all of v's children are linked into a cycle, thus incorporating them into the gray/black strongly connected subgraph.

We give intuition regarding the isolated children and the algorithm's preference for them. If v has an isolated boundary child, v', then we may not be able to CHAIN

Algorithm 1: Visit(Node v)								
1	if $v \rightarrow p$ then /* Invariant I3a */							
2	$v_{to} = p$, and $v_{from} = v$							
3	else if $p \rightarrow v$ then /* Invariant I3b */							
4	$v_{to} = v$, and $v_{from} = p$							
5	else /* Invariant I3c */							
6	if $\exists v' \in isolated(v) \ s.t. \ in-range(v', dest(v))$							
	then							
7	\bigvee v _{from} = v, v _{to} = dest(v)							
8	else if $\exists v' \in isolated(v)$							
	s.t. in-range $(v', source(v))$ then							
9	$v_{from} = source(v), v_{to} = v$							
10	else if $\exists v' \in boundary(v)$							
	s.t. in-range $(v', dest(v))$ then							
11	\bigvee v _{from} = v, v _{to} = dest(v)							
12	else							
13	$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ $							
14	$4 v_{\text{via}} = \emptyset$							
15	if $\exists v' \in boundarv(v) \ s.t. \ in-range(v', v_{from}) \land$							
	in-range (v', v_{to}) then							
16	$ \mathbf{v}_{via} = v'$							
17	if $\exists v' \in isolated(v) \ s.t. \ in-range(v', v_{from}) \land$							
	in-range (v', v_{to}) then							
18	$v_{via} = v'$							
19	REPLACE $v_{from} \rightarrow v_{to} ~{\rm with}~ v_{from} \rightarrow v_{via}~{\rm and}$							
	\downarrow v _{via} \rightarrow v _{to}							
20	$CHAIN(v,Children(v)\setminus\{v_{via}\})$							

CCCG 2011, Toronto ON, August 10-12, 2011

Subroutine 2: $CHAIN(v, v_1', v_2' \dots, v_\ell')$	
Add edges: $v \to v'_1 \to v'_2 \to \dots v'_{\ell-1} \to v'_\ell \to v$	

v' with the other children since the angle between it and the next sibling (in clockwise or counter-clockwise order) is $> 120^{\circ}$, and thus the next sibling may be at a distance > $\sqrt{3}$. Observe that in the portion of the IF statement involving Invariant (I3c) (lines 5-13), there is a preference for initializing v_{from} and v_{to} such that they both are in range of an isolated child. (Since one of these two variables will be set to v which is within range of all its children, we only need to check if dest(v) = dor source(v) = s is within range.) This is done so that v_{via} will be set to an isolated child (in line 18), and thus an isolated child undergoes the REPLACE operation rather than being chained with the other children. (In Section 3 we prove that the remaining children can be chained.) If no isolated child is within range in lines 6-9, then the algorithm attempts to set v_{from} and v_{to} so that they are within range of a regular boundary child (lines 10-13). The reason for this is that to maintain our invariants, we must not chain more than 3 children. Thus if there is a boundary child in range, then it undergoes the REPLACE operation, and the other (at most 3) children are chained.

3 Proof of Correctness

We now prove that Algorithm 1 is correct. We begin by proving that the CHAIN and REPLACE operations only add edges between nodes that are in range of each other and that they maintain Invariants (I1), (I2), and (I3). We then show that these operations also ensure that Invariant (I4) is satisfied. In what follows, let p(w)denote the parent of node w.

Consider the REPLACE operation in line 19. Observe first that execution only reaches line 19 if in-range(v_{from} , v_{via}) = in-range(v_{via} , v_{to}) = true, and therefore the edge updates are valid. We now verify that v_{via} satisfies the invariants after REPLACE. If v satisfies (I3a), or if v satisfies (I3c) and in-range(v_{via} , dest(v)) = true, then $v_{from} = v$. After REPLACE, v_{via} will satisfy (I3b) since $v = v_{from} \rightarrow v_{via}$ and $v = p(v_{via})$. Otherwise, v satisfies (I3b), or v satisfies (I3c) and in-range(v_{via} , source(v)) = true, and so $v_{to} = v$. After REPLACE, v_{via} will satisfy (I3a) since v_{via} , source(v)) = true, and so $v_{to} = v$. After REPLACE, v_{via} will satisfy (I3a) since $v_{via} \rightarrow v_{to} = v$ and $v = p(v_{via})$. It is easy to verify that v_{via} satisfies (I2), and since REPLACE does not change the number of nodes pointed to by v_{from} and v_{to} , they continue to satisfy either (I1) or (I2).

We now prove the correctness of the CHAIN operation. It is easy to verify that the children involved in CHAIN satisfy (I2) afterwards, since their color changes from white to gray and CHAIN makes them each point to one node. In addition, v satisfies (I1) since v points to one gray/black node before CHAIN, and CHAIN makes it point to one more. Therefore, we focus on verifying (I3). In each case that follows, when $v_{via} \neq \emptyset$, we assume v_{via} is initialized to boundary child $v_{deg(v)-1}$; situations in which v_{via} is initialized to v_1 are analogous.

- Case 1 (deg(v) = 1) ∨ (deg(v) = 2 ∧ v_{via} ≠ Ø). In this case there are no points in Children(v) \ {v_{via}}.
- Case 2 (deg(v) = $2 \wedge v_{via} = \emptyset$) \lor (deg(v) = $3 \wedge v_{via} \neq \emptyset$). v_1 is the only child in Children(v) \setminus { v_{via} }, and CHAIN adds edges $v \rightarrow v_1 \rightarrow v$. Since $v = p(v_1)$, in-range(v, v_1) = true. Thus, v_1 satisfies (I3a).
- Case 3 (deg(v) = 3 \land v_{via} = \emptyset) \lor (deg(v) = $4 \land$ v_{via} $\neq \emptyset$). v_1 and v_2 are the two children in Children(v) \setminus {v_{via}}, and CHAIN adds edges $v \rightarrow v_1 \rightarrow v_2 \rightarrow v$. Note that since $v = p(v_1) = p(v_2)$, in-range(v, v_1) = in-range(v, v_2) = true. Also, by Lemma 4 in-range(v_1, v_2) = true. Thus, v_1 satisfies (I3b) and v_2 satisfies (I3a).

Case 4 (deg(v) = 4 ∧ v_{via} = Ø) ∨ (deg(v) = 5). Note that if deg(v) ≥ 4 then v_{via} ≠ Ø by Lemma 3. Therefore, we only need to handle the case when deg(v) = 5 ∧ v_{via} ≠ Ø. In this case, v₁ , v₂, and v₃ are the three children in Children(v) \ {v_{via}}, and CHAIN adds edges v → v₁ → v₂ → v₃ → v. Note that since v = p(v₁) = p(v₃), in-range(v, v₁) = in-range(v, v₃) = true. Also, by Lemma 4 in-range(v₁, v₂) = in-range(v₂, v₃) = true. Thus, v₁ satisfies (I3b) and v₃ satisfies (I3a).

To complete the proof, we show that v_2 satisfies Invariant (I3c). First we verify that $\widehat{v_1vv_2} + \widehat{v_2vv_3} \leq$ 180°. This is true since v_{via} is a boundary child of v, and thus the remaining children v_1 , v_2 and v_3 are radially consecutive about v. For a degree 5 node, any three radially consecutive adjacent nodes can span at most 180° , since otherwise the sum of all five angles is more than 360° (because the angle between radially consecutive adjacent edges in a MST is at least 60°). Finally, we verify that v_1 and v_3 are on opposite sides of the line through $v_2 v_2$. For contradiction, suppose they are on or to the same side of this line. Because v_1, v_2, v_3 are radially consecutive, this implies that all five nodes adjacent to v are on or to the same side of the line through $v_2 v$, which again is impossible in a MST.

We end by proving that (I4) is satisfied after visiting v. Let G be the gray/black communication subgraph just prior to v being visited. By the inductive hypothesis, G is strongly connected. Consider the REPLACE operation. Observe that both v_{from} and v_{to} are gray/black nodes and thus are in G. In addition, $v_{from} \rightarrow v_{to}$ corresponds to an edge in G. It is straightforward then to verify that adding node v_{via} to G and replacing edge $v_{from} \rightarrow v_{to}$ with $v_{from} \rightarrow v_{via}$ and $v_{via} \rightarrow v_{to}$ results in a strongly connected graph. Similarly, adding v's children, v'_1, \ldots, v'_ℓ , involved in the CHAIN operation to G along with edges $v \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_\ell \rightarrow v$ results in a strongly connected graph. This combined with the fact that v's children are all colored gray when v is visited ensures that Invariant (I4) is satisfied.

Due to space constraints, we omit the proof of the following lemma.

Lemma 1 Let (a, b, c, d) be a path in a minimum spanning tree T such that a and d lie on or to a same side of a line through bc. Then $\widehat{abc} + \widehat{bcd} > 150^{\circ}$.

Lemma 2 Let (a, b, c, d) be a path in MST₅ such that a and d lie on or to the same side of a line through bc. Furthermore, $60^{\circ} \le \widehat{abc} \le 150^{\circ}$, $60^{\circ} \le \widehat{bcd} \le 150^{\circ}$, and $\widehat{abc} + \widehat{bcd} \le 210^{\circ}$. Then $|ad| \le \sqrt{3}$.

Proof. Let D(p,r) denote the open disk of radius r centered at point p, let $\partial D(p,r)$ denote its boundary,



Figure 2: Lemma 2: Regions of diameter $\sqrt{3}$.

and let $D[p,r] = D(p,r) \cup \partial D(p,r)$ denote the closed disk. Rotate MST_5 so that bc is vertical (as shown in Fig. 2a), and a and d lie right of bc. For simplicity, let |bc| = 1, since this is the value for which |ad| is maximum. Assume $\widehat{bcd} \leq \widehat{abc}$. The case when $\widehat{bcd} \geq \widehat{abc}$ is symmetrical. We start with a small observation regarding certain regions of diameter $\sqrt{3}$.

Fix $0 \leq \alpha \leq 30^{\circ}$, and define the points $p = p(\alpha)$, $q = q(\alpha)$, $r = r(\alpha)$ and $s = s(\alpha)$ as follows: p is the point above and right of b, such that $\widehat{pbc} = 120^{\circ} + \alpha$ and |bp| = 1; q is the right intersection point between $\partial D(p, \sqrt{3})$ and $\partial D(b, 1)$; r is the intersection point between the ray with origin p passing through b, and $\partial D(p, \sqrt{3})$; and sis the corner of the equilateral triangle Δprs , right of r. Refer to Figure 2a. Then the following properties hold:

- (P1) $\widehat{pbc} + \widehat{bcq} = 210^{\circ} + \frac{\alpha}{2}$. This follows immediately from the fact that $\widehat{pbq} = 120^{\circ}$ (because |pb| = |bq| = 1and $|pq| = \sqrt{3}$), thus $\widehat{cbq} = \alpha$ and $\widehat{bcq} = 90 - \alpha/2$.
- (P2) The closed region formed by the intersection $D[p,\sqrt{3}] \cap D[r,\sqrt{3}] \cap D[s,\sqrt{3}]$ has diameter $\sqrt{3}$. We abuse the terminology here and denote this region by $\mathsf{lune}[p,r,s]$.

We apply these properties repeatedly, to determine regions for which the lemma holds. We start with the value $\alpha = 30^{\circ}$, so that \widehat{pbc} is at its maximum value of 150°. Let points p, q, r, s be as defined above. (See Figure 2b.) By property (P1), $\widehat{bcq} = 75^{\circ}$. If both a and d lie inside lune[p, r, s], then the lemma holds by property (P2). Suppose then that d lies outside lune[p, r, s]. Observe that d cannot lie in D(b, 1), because then |bd| < |bc| and |bc| is not an edge in MST₅. So it must be that $\widehat{bcd} \ge \widehat{bcq} = 75^{\circ}$, meaning that $\widehat{abc} \le 135^{\circ}$ (because their sum does not exceed 210°, by the lemma statement). We capture this situation by resetting $\alpha = 15^{\circ}$ and redefining p, q, r, s for this new α value. (See to

Figure 2c.) By property (P1), $\hat{bcq} = 82.5^{\circ}$. If both a and d lie inside lune[p, r, s], then the lemma holds by property (P2). If d lies outside lune[p, r, s], then $bcd \ge bcq = 82.5^{\circ}$, meaning that $abc \le 127.5^{\circ}$. After repeating these steps k times, we either get the result of the lemma, or we get $bcd \ge 60^{\circ} + 30^{\circ} \cdot (\frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^k}),$ and $\widehat{abc} \le 150^{\circ} - 30^{\circ} \cdot (\frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^k})$. In the limit, as $k \longrightarrow \infty$, we get that $\widehat{bcd} \ge 90^\circ$ and $\widehat{abc} \le 120^\circ$ (otherwise the lemma holds). When $\alpha = 0$, q and c coincide: $pbc = 120^{\circ}$ and $|pc| = \sqrt{3}$. (See Figure 2d.) Simple calculations show that $\widehat{bcp} = 30^\circ$, $\widehat{pcs} = 75^\circ$, therefore $\widehat{bcs} = 105^{\circ}$. If both a and d lie inside $\mathsf{lune}[p, r, s]$, then the lemma holds. Otherwise, if d lies outside |une[p, r, s], then $bcd \geq 105^{\circ}$ and therefore $abc \leq 105^{\circ}$. From this point on, we are in the situation $bcd \geq abc$, which is symmetric to the situation bcd < abc discussed above. This concludes the proof. \square

Lemma 3 If deg(v) ≥ 4 , then v_{via} is initialized. Furthermore, if v has an isolated child, then v_{via} is initialized to an isolated child.

Proof. If v satisfies invariant (I3a) or (I3b), then $v_{to}, v_{from} \in \{v, p\}$. Note that at most one boundary child v' of v may satisfy $\widehat{v'vp} > 120^\circ$, because each angle between radially consecutive children of v is at least 60°, and the sum of all these angles is 360°. It follows that the second boundary child (which always exists, because $\deg(v) \ge 4$) is within range of both p and v, therefore the condition of the IF statement on line 15 evaluates to true and v_{via} is initialized on line 16. By similar arguments, if v has an isolated boundary child, say v_1 , then with the exception of $\widehat{v_1vv_2}$, all other angles at v must be smaller than 120°. Thus v_1 is within range of p and v, and therefore v_{via} is initialized to an isolated child in line 18.

Next we discuss the more complex situation when v satisfies invariant (I3c), so v is involved in a cycle $p \rightarrow s \rightarrow v \rightarrow d \rightarrow p$. (See for example Fig. 1c.) First recall that by Invariant (I3c), s and d lie to opposite sides of the line through vp, and s, v and d are radially consecutive children of p, in counter-clockwise order. Since $\deg(v) \geq 4$ and radially consecutive adjacent edges in an MST form an angle of at least 60°, boundary children v_1 and v_{k-1} (where $k = \deg(v)$) cannot lie on the same side of the line through vp. Also recall that v_1 , p and v_{k-1} are radially consecutive neighbors of v, in clockwise order. It follows that s and v_1 are both on or to one side of the line through vp, and d and v_k are both on or to the other side. We will use this fact when applying Lemma 2 below.

Consider first the case when $\deg(v) = 4$. Assume first that v has no isolated children. Note that $\widehat{v_1vp} + \widehat{v_3vp} \leq 240^\circ$, because each of $\widehat{v_1vv_2}$ and $\widehat{v_2vv_3}$ is at least 60°, and the sum of all these angles is 360°. These together imply that $\widehat{v_1vp} + \widehat{v_3vp} + \widehat{spd} \leq 240^\circ + 180^\circ = 420^\circ$, so at least one of $\widehat{v_3vp} + \widehat{vpd}$ and $\widehat{v_1vp} + \widehat{vps}$ is no greater than 210°. For the pair whose angle sum is no more than 210° - 60° = 150°. Having verified the requirements of Lemma 2 for one of the two paths, (v_1, v, p, s) or (v_3, v, p, d) , we use it to show that either in-range $(v_3, d) =$ true or in-range $(v_1, s) =$ true (or both). If in-range $(v_3, d) =$ true, then v_{from} and v_{to} are initialized in line 13 of the algorithm. In either case, the condition of the IF statement in line 15 of the algorithm evaluates to true.

Assume now that v has an isolated child, say v_1 . By definition, $\widehat{v_1vv_2} > 120^\circ$. This along with the fact that $\widehat{v_2vv_3} \ge 60^\circ$ implies that $\widehat{v_1vp} + \widehat{v_3vp} \le 180^\circ$. It follows that $\widehat{v_1vp} + \widehat{v_3vp} + \widehat{spd} \le 360^\circ$. So by Lemma 1, $\widehat{v_3vp} + \widehat{vpd} > 150^\circ$. These together imply that $\widehat{v_1vp} + \widehat{vps} \le 210^\circ$, and each of these angles has a value in the interval $[60^\circ, 150^\circ]$. By Lemma 2, in-range $(v_1, s) =$ true. Then v_{from} and v_{to} are initialized in line 9 of the algorithm, and the conditions of both IF statements in lines 15 and 17 of the algorithm evaluate to true.

Consider now the case when $\deg(v) = 5$. In this case, v has no isolated children: each angle at v is at least 60°, the sum of all five angles is 360°, therefore each angle is at most 120°. It follows that $\widehat{v_1vp} + \widehat{v_4vp} \leq 240^\circ$. (In fact, a stronger upper bound is 180°, but this is irrelevant to the discussion here.) This situation is identical to the degree 4, no isolated children case. \Box

Proof. Recall that when deg(v) = 5, no angle between two radially consecutive children of v exceeds 120°, and so the lemma is clearly true. So consider the situation where $\deg(v) < 5$. By similar arguments, at most one angle between two radially consecutive children of vmay exceed 120°. Furthermore, one of these children is necessarily a boundary (isolated) child since all angles between radially consecutive children involve a boundary child when v is of degree 3 or 4. As noted previously, a degree 4 vertex can have at most one angle > 120° . So if v is of degree 4 and has an isolated child, then both its boundary children form an angle $< 120^{\circ}$ with p, and thus both are within range of p. When deg(v) = 3, if one child is isolated, then they both are (since there are only two children.) In this case, at least one of the two children must be within range of p or else the sum of the three angles at v is more than 360°. If v satisfies invariant (I3a) or (I3b), then $v_{to}, v_{from} \in \{v, p\}$, therefore the conditions of both IF statements on lines 15 and 17 evaluate to true. It follows that v_{via} is set to an isolated child of v in line 18, and $\mathsf{Children}(v) \setminus \{\mathsf{v}_{\mathsf{via}}\}\$ contains either one child of v (the degree 3 case), or two children of v within range of each other (the degree 4 case).

It remains to discuss the more complex situation when v satisfies invariant (I3c), so v is involved in a cycle $v \rightarrow d \rightarrow p \rightarrow s \rightarrow v$, and $\widehat{spv} + \widehat{vpd} \leq 180^{\circ}$. Assume without loss of generality that v_1 is isolated, and v_1 and s lie on the same side of vp (refer to Fig. 1c). If $\deg(v) = 3$, then $\widehat{v_1vp} + \widehat{v_2vp} \leq 240^{\circ}$ (because v_1 and v_2 are both isolated, by our assumption). Arguments similar to the ones used in the proof of Lemma 3 show that in this case either in-range $(v_2, d) = \text{true}$, or in-range $(v_1, s) = \text{true}$, or both. If in-range $(v_2, d) = \text{true}$, v_{from} and v_{to} are initialized in line 7 of the algorithm; otherwise, v_{from} and v_{to} are initialized in line 9 of the algorithm. In either case, the conditions of both IF statements in lines 15 and 17 of the algorithm evaluate to true, and Children $(v) \setminus \{v_{\text{via}}\}$ contains a single child of v.

If $\deg(v) = 4$, Lemma 3 shows that $\operatorname{in-range}(v_1, s) = \operatorname{true}$. This guarantees that line 11 of the algorithm gets executed and $v_{\mathsf{via}} = v_1$. It follows that $\operatorname{Children}(v) \setminus \{\mathsf{v}_{\mathsf{via}}\}$ contains two children of v within range of each other.

Acknowledgement. Many thanks to the Fields Institute of Canada for financial support, and to all participants of the Fields workshop for fruitful discussions.

References

- W. Wu, H. Du, X. Jia, Y. Li, and S.C.-H. Huang: Minimum connected dominating sets and maximal independent sets in unit disk graphs. *Theor. Comp. Sci.*, 352:1– 7, 2006.
- [2] F. van Nijnatten: Range Assignment with Directional Antennas. Master's Thesis, Technische Universiteit Eindhoven, 2008.
- [3] I. Caragiannis, C. Kaklamanis, E. Kranakis, D. Krizanc, and A. Wiese: Communication in wireless networks with directional antennae. Proc. of the 20th Symp. on Parallelism in Algorithms and Architectures, Proc. of SPAA, pp. 344–351, 2008.
- [4] B. Bhattacharya, Y. Hu, Q. Shi, E. Kranakis, and D. Krizanc: Sensor network connectivity with multiple directional antennae of a given angular sum. *Proc.* of *IPDPS*, pp. 1–11, 2009.
- [5] B. Ben-Moshe, P. Carmi, L. Chaitman, M.J. Katz, G. Morgenstern, and Y. Stein: Direction Assignment in Wireless Networks. *Proc. of CCCG*, pp. 39–42, 2010.
- [6] M. Damian, and R. Flatland: Spanning Properties of Graphs Induced by Directional Antennas. *Proc. of FWCG*, Stony Brook, NY, 2010.
- [7] E. Kranakis, D. Krizanc, and O. Morales: Maintaining Connectivity in Sensor Networks Using Directional Antennae. *Theor. Aspects of Distr. Comp. in Sensor Netw.*, Part 2, pp. 59–84, 2011.
- [8] P. Bose, P. Carmi, M. Damian, R. Flatland, M.J. Katz, and A. Maheshwari. Switching to Directional Antennas with Constant Increase in Radius and Hop Distance *To appear in WADS*, 2011.

Euclidean Movement Minimization

Nima Anari^{*}

MohammadAmin Fazli[†]

Mohammad Ghodsi^{†‡}

Pooya Jalaly Khalilabadi[†]

MohammadAli Safari^{†§}

Abstract

We consider a class of optimization problems called movement minimization on euclidean plane. Given a set of nodes on the plane, the aim is to achieve some specific property by minimum movement of the nodes. We consider two specific properties, namely the connectivity (CON) and realization of a given topology (TOPOL). By minimum movement, we mean either the sum of all movements (SUM) or the maximum movement (MAX). We obtain several approximation algorithms and some hardness results for these four problems. We obtain an O(m)-factor approximation for CONMAX and CONSUM and an $O(\sqrt{m/OPT})$ -factor approximation for CON-MAX. We also extend some known result on graphical grounds in [1, 2] and obtain inapproximability results on the geometrical grounds. For the TOPOL problem (where the final decoration of the nodes must correspond to a given configuration), we find it much simpler and provide FPTAS for both MAX and SUM versions.

1 DIntroduction

Consider a number of moveable robots distributed over a plane in a far-flung manner. Each robot has an antenna with a limited maximum range, denoted by r_{max} . Robot s can communicate directly with robot t if and only if their distance is less than r_{max} . Robot s can also communicate indirectly with t if there is an ordered set of robots $s = r_1, r_2, \dots, r_p = t$ so that each r_i can directly communicate with r_{i+1} . With this explanation, we can form a dynamic graph whose vertices are the moveable robots on the plane and edges are formed by connecting each robot to every other robot residing in the disk with radius r_{max} around it. These geometric graphs are called UDGs (Unit Disk Graphs). **Definition 1** Given some points $p_1, ..., p_m$ in the euclidean plane, the UDG on these points is defined as a simple graph G = (V, E), where $V = \{1, ..., m\}$ and $E = \{\{i, j\} \mid |p_i - p_j|_2 \leq 1\}$

Suppose that robots are initially located at points p_1, p_2, \cdots, p_m . It is clear that all robots can communicate directly or indirectly with each other if and only if their corresponding UDG is connected. Our aim is to have the robots move in a way that they form a connected UDG after relocation (the points $p_1^*, p_2^*, \cdots, p_m^*$). We also want to efficiently optimize the travel distance of the robots before they reach their final locations. The term *efficiently* can be defined in many ways. In this paper, we consider two of such measures: namely SUM and MAX. In SUM, the goal is to minimize the sum of the movements of all robots, or formally to minimize $\sum_{i=1}^{m} |p_i^* - p_i|_2$. This parameter roughly measures the total energy consumed by the robots. In MAX, the goal is to minimize the maximum movement of all robots, i.e. minimizing $\max_{i \in \{1,...,m\}} |p_i^* - p_i|_2$. This parameter measures the amount of time needed to reach the final locations.

Using these two functions, we define two problems: CONMAX and CONSUM.

Definition 2 In CONMAX (resp. CONSUM) we want to move the robots so as to form a connected UDG and the optimization goal is MAX (resp. SUM).

Each of these problems can be considered in both graphical or geometrical settings.

Definition 3 In a graphical setting, robots move on a graph. At first, robots are placed on some vertices of the graph and at each turn, each robot can move to one of the adjacent vertices (each edge is considered to have one unit of length). In geometrical settings, robots are points belonging to a geometrical space (\mathbb{R}^2 in this paper) and are free to move in any direction in the space.

1.1 Other Works

Demaine et al. [1, 2] first introduced movement problems in graphical settings and extensively studied them. They defined 15 types of movement problems (borrowing from their terminology, from here on we use the

^{*}Computer Science Division, University of California Berkeley, email: anari@cs.berkeley.edu

[†]Department of Computer Engineering, Sharif University of Technology, emails: fazli, jalaly@ce.sharif.edu, ghodsi, safari@sharif.edu

 $^{^{\}ddagger}$ Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. This author's research was partially supported by IPM under grant No: CS1389-2-01

 $^{^{\$}}$ The research was partially supported by the Institute for Research in Fundamental Sciences under grant No: CS1389-4-09

words robot and *pebble* interchangeably). They consider five properties: connectivity, directed connectivity, path, independent set and matching and consider three objective functions: maximum movements, total movement and number of pebbles that move. This results in the following 15 problems: CONMAX, CONSUM, CONNUM, DIRCONMAX, DIRCONSUM, DIRCONNUM, PATHMAX, PATHSUM, PATHNUM, INDMAX, INDSUM, INDNUM, INDMAX, INDSUM, INDNUM.

Most of their salient results were proven in the context of graphs. They proposed an $\mathcal{O}\left(\sqrt{\frac{m}{OPT}}\right)$ -factor approximation algorithm for CONMAX and PATHMAX(mis the number of pebbles) and proved $\Omega\left(n^{1-\varepsilon}\right)$ inapproximability result for CONSUM and DIRCONMAX (nis the number of vertices in the ground graph) in graphical settings. They also gave an $\mathcal{O}(1)$ -approximation for INDMAX wit an additive error of $\mathcal{O}(1)$ in geometrical settings.

Note that all the algorithms presented in [1, 2] are in fact polynomial in n, the number of the nodes in the base graph, which makes them inefficient when $n \gg m$ which is a realistic assumption. Dealing with this, given that the number of mobile agents is typically much smaller than the complexity of the environment, in [3] the authors turn to fixed-parameter tractability. They characterize the boundary between tractable and intractable movement problems in a very general set up and show that many movement problems of interest have fixed parameter tractable algorithms.

1.2 Our Results

Our results include algorithms for CONMAX, CONSUM, TOPOLMAX, TOPOLSUM and an inapproximability result for CONMAX.

In section 2.1 we prove $(2 - \frac{\sqrt{2}}{2})$ -inapproximability for CONMAX in geometric settings which extends the hardness result of Demaine et al. [1, 2] about CONMAX in graphical settings.

Theorem 1 There is no polynomial algorithm for CONMAX in geometrical settings with an approximation factor of less than $2 - \frac{\sqrt{2}}{2}$, unless $\mathcal{P} = \mathcal{NP}$

In section 2.2 and 2.3 we give approximation algorithms for CONMAX and CONSUM on geometrical grounds. We present $\mathcal{O}(m)$ -factor approximation algorithm for both problems which improve the $\mathcal{O}(\sqrt{\frac{m}{OPT}})$ -approximation algorithm (with additive error of $\mathcal{O}(1)$) of Demaine et al. [2] in the cases where OPT is very small.

Theorem 2 There is an $\mathcal{O}(m)$ -factor approximation algorithm for CONMAX and CONSUM on geometrical grounds.

In the final part of this paper, we introduce a new kind of movement problems which is more constrained, in some sense, than the previously proposed problems: TOPOLMAX and TOPOLSUM.

Invariant 1 In problems TOPOLMAX and TOPOL-SUM, we are given m initial points $p_1, \ldots, p_m \in \mathbb{R}^2$ and a set of edges $E \subseteq \{\{i, j\} \mid i, j \in \{1, \ldots, m\}\}$. We are supposed to determine m points $p_1^*, \ldots, p_n^* \in \mathbb{R}^2$ in such a way that the UDG defined on p_1^*, \ldots, p_m^* contain all of the edges in E. The objective function we are trying to minimize can be either MAX or SUM which results in two different problems we call TOPOLMAX and TOPOL-SUM.

Although our results are stated in two dimensions, most of them can be easily extended to higher dimensions. In particular all of our approximation algorithms work for higher dimensions too.

Theorem 3 There is a FPTAS for the problems TOPOLMAX and TOPOLSUM.

2 CONMAX and CONSUM

2.1 Hardness Results

In this section we prove Theorem 1. First, we prove that CONMAX is 2-approximable on UDGs (graphical ground) only if $\mathcal{P}=\mathcal{NP}$. Then, with minor modifications, we prove Theorem 1. Our main idea is a proof of Demaine et al. in [1, 2] for hardness of CONMAX problem in graphical settings, but our case is more involved and needs many modifications.

For this, we reduce the hamiltonian cycle problem on 3-regular planar graphs which is known to be \mathcal{NP} hard [7]. Let us call this problem 3PHP.

We first start with a useful way of embedding planar graphs:

Lemma 4 (Valiant [6]). A planar graph G with maximum degree 4 can be embedded in the plane using $\mathcal{O}(|V|)$ area in such a way that its vertices are at integer coordinates and its edges are drawn so that they are made up of line segments of the form x = i or y = j, for integers i and j.

There is also a polynomial time algorithm to compute such an embedding [8].

We are now ready to prove the hardness of the CON-MAX problem on UDG grounds.

Theorem 5 There is no polynomial algorithm for CONMAX on UDG graphical grounds with approximation factor less than 2 unless $\mathcal{P} = \mathcal{NP}$

Proof. We prove this by reducing 3PHP. Assume that we have an instance of 3PHP problem; a 3-regular planar graph G in which we want to check for the existence



Figure 1: Graph G and its transformation process.

of a hamiltonian path between two specified vertices s and t. See Figure. 1(a) for an example.

First we use Lemma 4 to get an embedding H of G with integer coordinate vertices and horizontal or vertical edges (Figure. 1(b)).

Next we scale up all vertex coordinates by 6.0 to make each edge six times longer. The length of every edge e = (u, v) is now a multiple of 6.0. We put new vertices on every integer-coordinate point between u and v. So, the edge e = (u, v) is replaced by a path $P^e = u = v_0, v_1, \dots, v_{6k-1}, v_{6k} = v$ (notice that the distance between v_i and v_{i+1} is exactly one).

We color vertices $u = v_3, v_6, ..., v_{3i}, ..., v_{6k-3}, v_{6k} = v$ as black and the remaining vertices as white. See the resulting graph G in Figure. 1(d) (in this figure we have scaled up everything by 3 and not 6 for clarity and better understanding).

Since the degree of each vertex in the resulting UDG is at most 3, we can attach a new leaf to each black vertex via a unit length vertical or horizontal edge. We color these new leaves as gray and call the resulting new UDG G'.

Finally, we place one pebble on s and t and each gray vertex of G'. We also place two pebbles on each black vertex of G' except s and t. We show that G has a hamiltonian path between s and t if and only if the answer of CONMAX on G' is 1. If there is a hamiltonian path between s and t in G, we can move the pebble on each gray vertex to its neighboring vertex in V(G') and move pebbles on each black vertex to its neighboring vertices along the path corresponding to G's hamiltonian path that induce a connected subgraph in G'. For the reverse side, we show that if G does not have a hamiltonian path between s and t then the value of CONMAX is at least 2.0.

We show that when G is not hamiltonian, then establishing connectivity in G' requires a pebble in a gray vertex to move to a white vertex which requires a movement of 2.0.

Figure 2: The 3-degree vertex in minimum maximum degree spanning tree viewed in G'.

Consider the optimal connectivity establishment in G'. This induces a connected subgraph of G which is not a hamiltonian path and, therefore, has a maximum degree at least 3. Let u be a vertex with degree 3. It has only 2 pebbles. So, one of its neighboring white vertices, say v, can not be covered by the pebbles on it. If we remove the edge between u and v, we would have a subtree T' in which we need at least 2 moves to connect its pebbles to u's pebbles. This completes the proof. It is clear that nothing would be changed in this proof if we replace general UDG graphs with their specific type grids because we used only vertical/horizontal edges and integer coordinated vertices.

We can also use the above proof for the $(2 - \frac{\sqrt{2}}{2})$ inapproximability of CONMAX on geometrical grounds.

Proof. (of Theorem 1) The proof structure is almost identical to the proof of Theorem 5. In Figure. 2, the distance between vertex u and vertex v is 3. In proof of Theorem 1 we had to move pebbles only in integer units of length and the uv path was not covered by the pebbles placed on u. So to connect T''s pebbles to u's pebbles, we had to move them 2 units and the approximation factor was at least 2.

This is different on geometrical ground as u's two pebbles can move to every point of the plane without any limitation. So, there would be a movement of them in which the minimum coverage of these pebbles over all 3 outgoing paths of u is $\frac{\sqrt{2}}{2}$ (For example when they move in north-west and south-east direction with 45 degree slope). So the maximum of minimum coverage over these 3 paths by u's pebbles is at most $\frac{\sqrt{2}}{2}$ and again suppose that this minimum coverage is being happened for uv path. This completes the proof because in this situation the movement of T''s pebbles would be at least $3-1-\frac{\sqrt{2}}{2}=2-\frac{\sqrt{2}}{2}$ and this leads to the approximation factor $2-\frac{\sqrt{2}}{2}$.

2.2 O(m) approximation for CONMAX

If two pebbles are adjacent at the end then their original distance should be at most $\lambda = 2OPT + 1$. This means that if we scale down all distances by a factor of $\frac{1}{\lambda}$ then the corresponding UDG would be connected. This is the idea behind the algorithm: centered at one of the points, scale down all distances by a factor $\frac{1}{\lambda}$ and move every point to its new location after scaling. This yields an O(m)-factor approximation. The rest of details is left to the journal version of this paper.

2.3 O(m) approximation for CONSUM

We construct a complete weighted graph in which the weight of (i, j) is defined as $\max(0, (|p_i - p_j|_2 - 1)/2)$. Then we find a Minimum Spanning Tree (MST) of this graph. It can be shown that two fifths the weight of the MST is a lower-bound for the optimum solution.

Next, we do the following operation for each edge (i, j) of the MST: Removing the edge gives us two connected components. We translate each connected component along the edge $p_i p_j$ by a distance of $\max(0, (|p_i - p_j|_2 - 1)/2)$. Note that among the edges of the tree, only the distance between p_i and p_j is changed.

After all these operations, all edges of the MST become present in the resulting UDG. The total sum of movements is at most m times the weight of the original MST, hence an $\mathcal{O}(m)$ -approximation. More details are left to the journal version of this paper.

3 Predetermined Topology

Assume that we are given m different points $p_1, \ldots, p_m \in \mathbb{R}^2$. The goal is to make the UDG defined on these points have certain properties. One of the properties that might be desirable for the UDG to have, is to have it contain a certain predetermined graph.

Clearly one can assume that the given topology E is connected; otherwise, the problem can be solved for each connected component separately, and the solutions can be combined together. Hence, from now on we will assume that E is connected.

The main result we obtain is that there is a FPTAS for each one of these problems. Our FPTAS's use the ELLIPSOID method as a blackbox.

Remark 1 The ELLIPSOID method works with a separation oracle defined on a convex set; that is an oracle which when given a point p determines whether it's inside the convex set, and if the answer is false, returns a hyperplane separating the point and the convex set. Given a convex body $C \subset \mathbb{R}^n$ and an initial ellipsoid E_0 containing C, and an arbitrary positive number \underline{V} , the ELLIPSOID method either finds a point in C, or finds out that the volume of C is less than \underline{V} . The time it takes for the ELLIPSOID method to run is bounded by a polynomial in n and $\log(Vol(C)/\underline{V})$.

3.1 FPTAS for TOPOLMAX

In this section we will show how TOPOLMAX can be approximated using the ELLIPSOID method.

Our algorithm uses some of the results and tools from the $\mathcal{O}(m)$ approximation algorithm for CONMAX, including the definition and properties of the geometrical transformation homothety. For details refer to the journal version of this paper.

For two given points p_i and p_j to become at most 1 unit apart (in the Euclidean metric), one should be moved by at least $e_{ij} = \max(0, (|p_i - p_j|_2 - 1)/2)$.

Now given an instance of TOPOLMAX define \underline{O} to be $\max_{\{i,j\}\in E} e_{ij}$. Clearly \underline{O} is a lower-bound for OPT.

Lemma 4 An instance of TOPOLMAX can be solved by a sum of displacements of $2(m-1)\underline{O}$.

The main idea used behind the proof is exactly the same as the one used in CONMAX, namely the use of homotheties.

Let's formulate TOPOLMAX as a linear program. This linear program is exact, but unfortunately has infinitely many constraints. The following simple lemma forms the basis of this linear program.

Lemma 5 For a vector $v \in \mathbb{R}^2$, the inequality $|v|_2 \leq d$ holds if and only if for each unit vector $u \in S^1$ (S^1 is the unit circle), the inequality $u \cdot v \leq d$ holds.

Now let's formulate our problem as a non-linear program, and then convert it to a linear program. We can define the variables x_1, \ldots, x_m and y_1, \ldots, y_m to be the final coordinates of the points; i.e. $p_i^* = (x_i, y_i)$. Our problem can be formulated like the following

Note that this formulation can be completely written in terms of x_1, \ldots, x_m and y_1, \ldots, y_m ; we can simply replace each p_i^* by (x_i, y_i) . Now applying the previous lemma to this formulation, we can rewrite it like the following

Minimize sSubject To $(p_i^* - p_i) \cdot u \le s \quad \forall i \in \{1, \dots, m\}, u \in S^1$ $(p_i^* - p_i^*) \cdot u \le 1 \quad \forall \{i, j\} \in E, u \in S^1$

The new formulation is a linear program (although, with infinitely many constraints), since inner product is a bilinear operator. To use the ELLIPSOID method on this new formulation, we should first remove s. For

each $s \in \mathbb{R}^{\geq 0}$, define L_s to be the convex set in \mathbb{R}^{2m} defined by the constraints

$$(p_i^* - p_i) \cdot u \le s \quad \forall i \in \{1, \dots, m\}, u \in S^1$$
$$(p_i^* - p_i^*) \cdot u \le 1 \quad \forall \{i, j\} \in E, u \in S^1$$

 L_s is the intersection of infinitely many half-planes. Hence, it is convex. The optimum solution of TOPOL-MAX is the minimum s for which L_s is nonempty.

Because of the constraints $(p_i^* - p_i) \cdot u \leq s$, we can find a sphere surrounding L_s . This sphere is centered at the point $(p_1, \ldots, p_m) \in \mathbb{R}^{2m}$, and its radius is \sqrt{ms} . That is because

$$|(p_1^*, \dots, p_n^*) - (p_1, \dots, p_m)|_2 = \sqrt{\sum_{i \in \{1, \dots, n\}} |p_i^* - p_i|_2^2} \le \sqrt{ms^2} = \sqrt{ms}$$

Since this sphere can be surrounded by a hypercube with a side length of $2\sqrt{ms}$, the volume of this sphere is at most $(2\sqrt{ms})^{2m}$.

Note that using the previous lemma, existence of a separation oracle for L_s becomes obvious. In fact, we just have to check the unit vectors u which are parallel to the vectors $(p_i^* - p_i)$ and the vectors $(p_i^* - p_i^*)$.

We have all of the things we need for the ELLIPSOID method, except \underline{V} , the lower-bound on the volume of L_s . Note that $L_s \subseteq L_t$ for $s \leq t$. So if we obtain a lower-bound on the volume of L_s for one s, that lowerbound also works for every L_t for which $t \geq s$. Let OPT denote the optimum solution of TOPOLMAX. Let $s^* = (1 + \delta)OPT$. Our goal is to derive a lower-bound on the volume of L_{s^*} .

Lemma 6 The volume of L_{s^*} is greater than or equal to $(\frac{\delta OPT}{2m})^{2m}$.

Proof. Since L_{OPT} is nonempty, one can find a point $(q_1, \ldots, q_m) \in L_{OPT} \subseteq L_{s^*} \subset \mathbb{R}^{2m}$.

Let H^{α} denote the α -homothety with respect to q_1 .

Consider the points $H^{\alpha}(q_1), \ldots, H^{\alpha}(q_m)$. Since each q_i can be reached from q_1 by a path consisting only of the edges in E, we have $|q_i - q_1|_2 \le m - 1$. So

$$|H^{\alpha}(q_i) - q_i|_2 = (1 - \alpha)|q_i - q_1|_2 \le (1 - \alpha)(n - 1)$$

Since $|q_i - p_i| \leq OPT$, we have $|H^{\alpha}(q_i) - p_i| \leq OPT + (1 - \alpha)(m - 1)$.

Because of the properties of homotheties, for each $\{i, j\} \in E$, we have $|H^{\alpha}(q_i) - H^{\alpha}(q_j)|_2 \leq \alpha$. Now let r_1, \ldots, r_m be some arbitrary points for which we have $|r_i - H^{\alpha}(q_i)|_2 \leq (1 - \alpha)/2$. For each $\{i, j\} \in E$, we have

$$\begin{aligned} |r_i - r_j|_2 &\leq & |H^{\alpha}(q_i) - H^{\alpha}(q_j)|_2 + |r_i - H^{\alpha}(q_i)|_2 \\ &+ |r_j - H^{\alpha}(q_j)|_2 \\ &\leq & \alpha + \frac{1 - \alpha}{2} + \frac{1 - \alpha}{2} = 1 \end{aligned}$$

We also have

$$\begin{aligned} |r_i - p_i| &\leq |r_i - H^{\alpha}(q_i)| + |H^{\alpha}(q_i) - p_i| \\ &\leq \frac{1 - \alpha}{2} + OPT + (1 - \alpha)(n - 1) \\ &\leq OPT + (1 - \alpha)m \end{aligned}$$

This shows that there is a copy of $\underbrace{B_{(1-\alpha)/2} \times \cdots \times B_{(1-\alpha)/2}}_{m}$ inside $L_{OPT+(1-\alpha)m}$, where B_x shows a 2-dimensional ball of radius x. Since $\operatorname{Vol}(B_x) = \pi x^2 \ge x^2$, we have

$$\mathrm{Vol}(L_{OPT+(1-\alpha)m}) \geq (\frac{1-\alpha}{2})^{2m}$$

We want α to be chosen in such a way that $OPT + (1 - \alpha)m \leq s^*$. This can be obtained by setting $\alpha = 1 - (s^* - OPT)/m$. For this α , we have $1 - \alpha = (s^* - OPT)/m = \delta OPT/m$. Therefore

$$\operatorname{Vol}(L_{s^*}) \ge (\frac{\delta OPT}{2m})^{2m}$$

Using the lower-bound $(\frac{\delta OPT}{2m})^{2m}$ as the parameter <u>V</u> of ELLIPSOID, one can see that the ellipsoid method is able to find a point inside L_{s^*} in time bounded by a polynomial of m and

$$\log \frac{(2\sqrt{m}(1+\delta)OPT)^{2m}}{(\frac{\delta OPT}{2m})^{2m}} = \log(4m\sqrt{m}\frac{1+\delta}{\delta})^{2m}$$
$$= 2m\log(4m\sqrt{n}\frac{1+\delta}{\delta}) = \mathcal{O}\left(\operatorname{poly}(m,1/\delta)\right)$$

We don't know OPT, so we can't actually set the parameter \underline{V} of ELLIPSOID to the above lower bound; this is not a problem, as we can just run the ELLIPSOID method for the time bound we have obtained (which depends only on m and δ).

Using the previous lemmas it's easy to see that Algorithm 3.1 is a $(1 + \epsilon)$ -approximation for TOPOLMAX. If OPT resides in an interval $[(1 + \delta)^i \underline{O}, (1 + \delta)^{i+1}\underline{O}]$, then $\overline{s} = (1 + \delta)^{i+2}\underline{O}$ is definitely larger than $s^* = (1 + \delta)O$. Hence, ELLIPSOID finds a point of $L_{\overline{s}}$ in the time limit given. But we have the following inequality (we're assuming without loss of generality that $\epsilon \leq 1$)

$$\overline{s} \leq (1+\delta)^2 (1+\delta)^* \underline{O} \leq (1+\delta)^2 OPT = (1+2\delta+\delta^2) OPT \leq (1+2\delta+\delta) OPT = (1+\epsilon) OPT$$

So the solution found by Algorithm 3.1 is a $(1 + \epsilon)$ -approximation.

Algorithm 1 TOPOLMAX

- 1: Calculate <u>O</u> using the formula $\max\{(|p_i p_j|_2 1)/2 \mid \{i, j\} \in E\}.$
- 2: Let $\delta = \epsilon/3$. Divide the interval $[\underline{O}, 2(m-1)\underline{O}]$ into $\mathcal{O}(\log m/\log(1+\epsilon))$ intervals of the form $[(1+\delta)^{i}\underline{O}, (1+\delta)^{i+1}\underline{O}]$. Sort the interval endpoints in an increasing order.
- 3: for each interval endpoint like a do
- 4: Run the ELLIPSOID method on L_a using the initial bounding sphere of radius \sqrt{ma} around the origin. Run this method until it finds an answer or the upper-bound on the execution time we found earlier passes.
- 5: If the ELLIPSOID method finds a solution point, then stop the algorithm and return that solution.
- 6: **end for**
- 7: If no solution is found, return the solution found from our previous $\mathcal{O}(m)$ -approximation algorithm.

3.2 TOPOLSUM

The same method used in the previous section can be slightly modified to work for TOPOLSUM. One can again find similar bounds on the volume of the convex body and again show that the ELLIPSOID method works in polynomial time.

4 Concluding Remarks

In this paper we showed that FPTAS exists once the target UDG is known, i.e. adjacent vertices are specified. Therefore the hardness of CONMAX, CONSUM and similar movement problems lie in finding the topology of the target UDG. We know some good heuristic ways of guessing the target topology but have little theoretical justification for their behavior.

Considering other types of properties such as obtaining an independent set of a given size or considering a bigger class of graphs like disc graphs are good research directions to follow. Directly related to our work one can narrow the hardness and approximability gap by improving one or both. We conjecture that both CON-MAX and CONSUM are approximable within constant factors.

References

- E. Demaine, M. Hajiaghayi, H. Mahini, S. Oveisgharan, A. Sayedi, and M. Zadimoghaddam. Minimizing movement, In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007.
- [2] E. Demaine, M. Hajiaghayi, H. Mahini, S. Oveisgharan, A. Sayedi, and M. Zadimoghaddam. Minimizing Movement, ACM Transactions on Algorithms, volume 5, number 3, July 2009, Article 30.

- [3] E.Demaine, M.Hajiaghayi, D.Marx, Minimizing Movement: Fixed-Parameter Tractability, In Proceedings of the 17th Annual European Symposium on Algorithms, 2009.
- [4] MA.Fazli, Movement Minimization in Euclidean Plane, BSc Thesis, Sharif University of Technology, 2009.
- [5] N.Ahmadipour, Movement Minimization for Network Design in Geometric Spaces, BSc Thesis, Sharif University of Technology, 2010
- [6] L.G. Valiant, University considerations in VLSI circuits, IEEE Trans. Computers 30 (1981) 135-140.
- [7] M.R. Garey, D.S. Johnson and R.E. Tarjan, The Planar Hamiltonian Problems is NP-complete, SIAM Journal on Computing, Vol. 5(1976), pp 704-714.
- [8] A. Itai, C.H. Papadimitriou and J.L.Szwarcfiter, Hamilton paths in grid graphs, SIAM Journale of Computing. 11 (1982) 676-686.
- [9] M.Mahdian, Y.Ye and J.Zhang, Approximation algorithms for metric facility location problems, SIAM Journal on Computing, 2006, p411-432.

A Randomly Embedded Random Graph is Not a Spanner

Abbas Mehrabian*

Abstract

Select n points uniformly at random from a unit square, and then form a random graph on these points by adding an edge joining each pair independently with probability p. We show that for every fixed $\epsilon > 0$, if $p < 1 - \epsilon$, then with probability approaching 1 as n becomes large, the resulting embedded graph has unbounded stretch factor.

1 Introduction

Select *n* points uniformly at random from a unit square, and then form a random graph *G* on these points by joining each pair independently with probability p = p(n). This is not a "random geometric graph" in the usual sense of that term, because points are connected without regard to their geometric distance. For every two points *u* and *v*, let d(u, v) denote their Euclidean distance. Make *G* weighted by putting weight d(u, v) on every edge *uv*. For two vertices *u* and *v*, let $d_G(u, v)$ denote their shortest-path distance on (weighted) *G*, and let $d_G(u, v) = \infty$ if there is no (u, v)-path in *G*. The *stretch factor* of *G* is defined as

$$\max\frac{d_G(u,v)}{d(u,v)}$$

where the maximum is taken over all vertices u, v. In the open problem session of CCCG 2009 [1], O'Rourke asked if for $p > \ln n/n$, the resulting graph has a bounded stretch factor. We give a negative answer to this question. More precisely, we show that for every fixed $\epsilon > 0$, if $p < 1-\epsilon$, then with probability approaching 1 as n becomes large, the resulting graph has unbounded stretch factor.

2 Proof of the Main Result

Assume that n points are chosen independently and uniformly from a unit square, where n is sufficiently large. Let $p < 1 - \epsilon$ for some fixed $\epsilon > 0$, and build the graph G as in the introduction. In the following, with high probability means with probability 1 - o(1), where the asymptotics is with respect to n. Fix a positive λ . We will show that with high probability the graph G has stretch factor larger than λ .

Let m be a positive integer satisfying $m^2 \leq n/2 < 2m^2$. Partition the unit square into m^2 squares of side $\frac{1}{m}$ by drawing m-1 equally spaced vertical lines and m-1 equally spaced horizontal lines. We will call the generated squares of side $\frac{1}{m}$ the *small* squares. Let K be the number of small squares that contain exactly two points. The probability that some point lies on the boundary of some small square is zero, and we will assume that this does not happen.

Lemma 1 With high probability $K \ge e^{-8}n$.

Proof. Number the small squares arbitrarily from 1 to m^2 . We have

$$K = K_1 + K_2 + \dots + K_{m^2},$$

where K_i is the indicator variable for the event that the *i*-th small square contains exactly two points. Hence $\mathbb{E}K_i$ is the probability of this event. Let $1 \leq i \leq m^2$ be arbitrary. The probability that a random point lies in the *i*-th small square is $1/m^2$. So the probability that exactly two of the *n* random points are in this square is

$$\mathbb{E}K_i = \left(\frac{n(n-1)}{2}\right) \left(\frac{1}{m^2}\right)^2 \left(1 - \frac{1}{m^2}\right)^{n-2}.$$

By the choice of m, we have $m^4 \le n^2/4$ and $m^2 \ge n/4$. These bounds together with the fact that for large n, $\exp(-5/n) \le 1 - 4/n$ give

$$\mathbb{E}K_i \ge \left(\frac{n^2}{4}\right) \left(\frac{4}{n^2}\right) \left(1 - \frac{4}{n}\right)^n \ge e^{-5}.$$

Thus by linearity of expectation,

$$\mathbb{E}K = \mathbb{E}K_1 + \mathbb{E}K_2 + \dots + \mathbb{E}K_{m^2} \ge m^2 e^{-5} \ge ne^{-7}.$$

Now, we estimate $\operatorname{Var}(K)$ and show that $\operatorname{Var}(K) = O(n)$. Let i, j be arbitrary, with $1 \leq i < j \leq m^2$. The probability that both the *i*-th square and the *j*-th square contain exactly two points is

$$\mathbb{E}[K_i K_j] = \binom{n}{2} \binom{n-2}{2} \left(\frac{1}{m^2}\right)^4 \left(1 - \frac{2}{m^2}\right)^{n-4},$$

and we have

$$\mathbb{E}K_i \mathbb{E}K_j = (\mathbb{E}K_i)^2 = {\binom{n}{2}}^2 \left(\frac{1}{m^2}\right)^4 \left(1 - \frac{1}{m^2}\right)^{2(n-2)}.$$

^{*}Department of Combinatorics and Optimization, University of Waterloo, amehrabi@uwaterloo.ca

Thus,

$$\operatorname{Cov}(K_i, K_j) = \mathbb{E}[K_i K_j] - \mathbb{E}K_i \mathbb{E}K_j$$
$$\leq {\binom{n}{2}}^2 \left(\frac{1}{m^2}\right)^4 \left[\left(1 - \frac{2}{m^2}\right)^{n-4} - \left(1 - \frac{1}{m^2}\right)^{2(n-2)} \right].$$

Moreover, since $m^2 = \Theta(n)$,

$$\left(1 - \frac{2}{m^2}\right)^{n-4} - \left(1 - \frac{1}{m^2}\right)^{2(n-2)}$$

$$= \exp\left(\frac{-2(n-4)}{m^2}\right) - \exp\left(\frac{-2n+4}{m^2}\right) + O(1/m^2)$$

$$= \exp\left(-\frac{2n}{m^2}\right) \left(e^{8/m^2} - e^{4/m^2}\right) + O(1/m^2)$$

$$= e^{\Theta(1)}O(1/m^2) + O(1/m^2) = O(1/m^2).$$

Thus,

$$Cov(K_i, K_j) \le {\binom{n}{2}}^2 \left(\frac{1}{m^2}\right)^4 \left[\left(1 - \frac{2}{m^2}\right)^{n-4} - \left(1 - \frac{1}{m^2}\right)^{2(n-2)} \right]$$
$$= O(n^4)O(1/m^8)O(1/m^2) = O(1/m^2).$$

Consequently, since $\operatorname{Var}(K_i) = \mathbb{E}K_i(1 - \mathbb{E}K_i) \le 1/4$,

$$Var(K) = \sum_{i} Var(K_{i}) + \sum_{i \neq j} Cov(K_{i}, K_{j})$$
$$\leq m^{2}/4 + 2\binom{m^{2}}{2}O(1/m^{2}) = O(m^{2}) = O(n)$$

Let $t = (e^{-7} - e^{-8}) n$. Then Chebyshev's inequality gives

$$\begin{aligned} \mathbf{Pr}\left[K < ne^{-8}\right] &\leq \mathbf{Pr}\left[|K - \mathbb{E}K| \geq t\right] \\ &\leq \frac{\operatorname{Var}(K)}{t^2} = o(1). \end{aligned}$$

Thus, with high probability, $K \ge e^{-8}n$.

A small square S with exactly two points u and v is called *nice* if the following statements are true.

- 1. Points u and v lie in the circle which is co-centric with S and has radius $r = (7m\lambda)^{-1}$.
- 2. The points u and v are nonadjacent in graph G.

We claim that the existence of a nice square S implies that the stretch factor of G is larger than λ . In fact, the (weighted) distance between u and v in G is at least $\frac{1}{2m} - r$, since any (u, v)-path in G should go out of Sat the very first step. However, the Euclidean distance between u and v is at most 2r, and we have

$$\left(\frac{1}{2m} - r\right) > \lambda(2r).$$

Let A be the (random) set of small squares that contain exactly 2 points. Let $S \in A$ with points u and v inside it. Then for S to be nice, u and v should lie in the co-centric circle with radius r, and u and v should be nonadjacent in G. The probability of the former is $(\pi r^2 m^2)^2 = (\pi/7\lambda)^2$, and the probability of the latter is 1-p. These two events are independent, so the probability that S is not nice is $1 - (\pi/7\lambda)^2(1-p)$.

Let A_0 be a fixed set of small squares, and assume that we condition on A being equal to A_0 . Then the events happening inside each square of A_0 are independent of the others. In particular, the events

$$\{S \text{ is nice} : S \in A_0\}$$

are mutually independent, hence the probability that no nice square exists is equal to

$$\left[1 - (\pi/7\lambda)^2(1-p)\right]^{|A_0|}$$
.

Therefore, conditioned on the event $|A| \ge e^{-8}n$, the probability that no nice square exists is at most

$$[1 - (\pi/7\lambda)^2(1-p)]^{e^{-8}n}$$

which, since $p < 1-\epsilon$, approaches 0 as n becomes large. By Lemma 1, with high probability the size of A is at least $e^{-8}n$, i.e. the event $|A| \ge e^{-8}n$ happens with probability 1-o(1). Thus with high probability a nice square exists and the stretch factor is larger than λ .

Acknowledgement. The author thanks Nick Wormald for suggesting an alternative step in the proof of Lemma 1.

References

 Erik D. Demaine and Joseph O'Rourke. Open Problems from CCCG 2009. In Proceedings of the 22nd Canadian Conference on Computational Geoemtry (CCCG 2010), Winnipeg, Manitoba, Canada, August 9–11, 2010, 83–86.

Approximation Algorithms for the Discrete Piercing Set Problem for Unit Disks

Minati De^{*†}

Gautam K. Das[‡]

Subhas C. Nandy*

Abstract

In this note, we shall consider constant factor approximation algorithms for a variation of the discrete piercing set problem for unit disks. Here a set of points Pis given; the objective is to choose minimum number of points in P to pierce all the disks of unit radius centered at the points in P. We first propose a very simple algorithm that produces a 14-factor approximation result in $O(n \log n)$ time. Next, we improve the approximation factor to 4 and then to 3. Both algorithms run in polynomial time.

1 Introduction

The piercing set of a set of objects S in \mathbb{R}^2 is a set of points Q such that each object in S contains at least one point in Q. Here the problem is, given the set S, compute a piercing set of minimum size. Let us consider the intersection graph G = (V, E) of the objects in S. Its nodes V correspond to the members in S, and an edge $e = (u, v) \in E$, for a pair of vertices $u, v \in V$ implies that the two objects corresponding to the nodes u and v intersect. A clique C in the graph G implies that each pair of objects corresponding to the nodes in C are intersecting. But, it does not imply that all of them have a non-empty common intersection region. In other words, a clique C in G does not imply that the objects corresponding to the members in C can be pierced by a single point. However, if S consists of a set of axis-parallel rectangles, then the minimum piercing set corresponds to the minimum clique cover 1 of the intersection graph of the members in S.

The minimum clique cover problem for a set of axisparallel unit squares in $\mathbb{I}\!\!R^2$ is known to be NP-hard [17]. Hochbaum and Maass [16] proposed a PTAS for the minimum clique cover problem for a set of axis-parallel unit squares with time complexity $n^{O(1/\epsilon^2)}$. The time complexity was later improved to $n^{O(1/\epsilon)}$ by Feder and Greene [13], and by Gonzalez [14]. Chan [5] proposed a PTAS for squares of arbitrary size with time complexity $n^{O(1/\epsilon^2)}$. In fact, this algorithm works for any collection of fat objects. Chan and Mahmood [6] considered the problem for a set of axis-parallel rectangles of fixed height (but of arbitrary width), and proposed a PTAS with $n^{O(1/\epsilon^2)}$ time complexity.

The minimum clique cover problem for unit disk graph also has a long history. The problem is known to be NP-hard [9], and a 3-factor approximation algorithm is easy to obtain [19]. Recently, Dumitrescu and Pach [12] proposed an $O(n^2)$ time randomized algorithm for the minimum clique cover problem with approximation ratio 2.16. They also proposed a polynomial time approximation scheme (PTAS) for this problem that runs in $O(n^{1/\epsilon^2})$ time. It is an improvement on a previous PTAS with $O(n^{1/\epsilon^4})$ running time [22].

Since, the disks do not satisfy the Helly's property², the minimum piercing set problem for unit disks is different from the minimum clique cover problem for unit disk graph. The minimum piercing set problem for disks has a lot of applications in wireless communication where the objective is to place the base stations to cover a set of radio terminals (sensors) distributed in a region. The minimum piercing set problem for unit disks is also NP-hard [3, 12]. Carmi et. al [3] proposed an approximation algorithm for this problem where the approximation factor is 38. In particular, if the points are distributed below a straight line L, and the base stations (of same range) are allowed to be installed on or above L only then a 4-factor approximation algorithm can be obtained provided all the points lie within an unit distance from at least one base station.

In the discrete version of the minimum piercing set problem for unit disks, two sets of points P and Q are given. The unit disks are centered at the points of P, and the piercing points need to be chosen from Q. The objective is to choose minimum number of points from Q to pierce all the disks in P. The problem is known to be NP-hard [18]. The first constant factor approximation result on this problem is proposed by Calinescu et al. [2]. It uses linear programming relaxation method to produce an 108-factor approximation result. The approximation re-

^{*}Indian Statistical Institute, Kolkata, India

[†]Visiting Carleton University, Canada, during April 1, 2011 -August 27, 2011. minati.isi@gmail.com

[‡]Indian Institute of Technology Guwahati, India

¹The minimum clique cover problem for a graph G = (V, E) is partitioning the vertex set V into minimum number of subsets such that the subgraph induced by each subset is a clique.

 $^{^2\}mathrm{A}$ set of object has the Helly property if each intersecting family has a non-empty intersection.

sult is then improved to 72 in [21], 38 in [3], and 22 in [8]. Finally, Das et al. [10] proposed an 18-factor approximation algorithm that runs in $O(n \log n + m \log m + mn)$ time, where |P| = n and |Q| = m.

Another variation of the discrete piercing set problem for unit disks assumes Q = P. In other words, the unit disks corresponding to the points in P need to be pierced by choosing a minimum number points from Pitself. In the literature, the problem is referred to as the minimum dominating set problem for the unit disk graph (or MDS problem in short). Here, an undirected graph is constructed with nodes corresponding to the points in P. Between a pair of nodes there is an edge if the distance between the two points is less than or equal to their common radius. A vertex in the graph dominates itself and all its neighbors. The objective is to choose minimum number of vertices to dominate all the vertices in the graph.

The problem is known to be NP-hard [7]. Ambuhl et al. [1] first proposed an approximation algorithm for this problem. They considered the weighted version of the problem where each node is attached with a positive weight. The objective is to find the minimum weight dominating set of the nodes in the graph. The approximation factor of their proposed algorithm is 72. Huang et al. [15] improved the approximation factor of the same problem to $6 + \epsilon$. Dai and Yu [11] further improved the approximation factor to $5 + \epsilon$. Though they have not analyzed the time complexity of their proposed algorithm, their algorithm needs $O(\frac{n^9}{\epsilon^2})$ time. Recently, Zou et al. [23] proposed a polynomial time $4 + \epsilon$ factor approximation algorithm. Nieberg and Hurink [20] proposed an $O(n^c)$ time PTAS for computing the minimum dominating set for unit disk graphs, where $c = (2r + 1)^2$, and r is an integer satisfying $(2r + 1)^2 < (1 + \epsilon)^{r/2}$. It accepts any undirected graph as input, and returns a $(1 + \epsilon)$ factor approximation solution for the dominating set problem, or a certificate indicating that the input graph is not a unit disk graph. For a 2-factor approximation result, the worstcase running time is obtained by setting $\epsilon = 1$; in that case, r will be equal to 22. Thus, the running time is $O(n^{(2r+1)^2}) = O(n^{(2\times 22+1)^2}) = O(n^{2025})$. Even for a 3-factor approximation result, the worst case time complexity (by putting $\epsilon = 2$) becomes $O(n^{625})$. Thus, this algorithm is not at all tractable from a practical point of view. Our present work is directed towards finding a tractable algorithm with a guaranteed constant factor approximation result. For the unweighted version of the discrete piercing set problem, the best known result is a 5-factor approximation algorithm proposed in [4], and it works for disks of arbitrary radii. This result is then used for the h-piercing problem, where the objective is to choose minimum number of points in P to pierce each disk by at least h points. The proposed approximation

factor was $5(2^h - 1)$.

We propose three methods that use almost similar type approach for the discrete piercing problem with Q = P. The first one produces a 14-factor approximation result in $O(n \log n)$ time. The second one produces a 4-factor solution in $O(n^9)$ time, and the last one produces a 3factor solution in $O(n^{18})$ time. Recall that the running time of the existing algorithm for producing a 3-factor approximation solution is $O(n^{625})$ [20]. Thus, our algorithm is a substantial improvement over the existing results in the literature. We can use this result to improve the approximation factor for the *h*-piercing problem [4] of constant radius disks to $3(2^h - 1)$ from $5(2^h - 1)$.

2 Approximation Algorithms

We are given a set of points P, where each point corresponds to a unit disk centered at that point. The objective is to choose a subset $P' \subseteq P$ of minimum cardinality such that the disk corresponding to each point in P contains at least one member of P'.

2.1 A simple 14-factor approximation algorithm

Consider a partitioning of the plane into a grid whose each cell is of size $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$. Since the maximum distance between any two points in a grid cell is less than or equal to 1, we can pierce all the disks centered at points of Pin a particular cell by choosing any one member $p \in P$ lying in that cell. In other words, if we draw a disk of unit radius, and centered at p, it covers all the points lying inside that cell. Note that, it may cover point(s) in the other cell(s). But, we show that a disk centered at a point $p \in P$ inside a grid cell may cover (some or all) points in at most 14 other grid cells.

1	2	3		4	5
6	7	8		9	10
11	12	A C	B D	14	15
16	17	18		19	20
21	22	23		24	25

Figure 1: Discrete piercing set for unit disks

Consider the 5×5 grid structure as shown in Figure 1. The length of each side of a cell is $\frac{1}{\sqrt{2}}$. The cells are numbered as $1, 2, \ldots, 25$. The cell 13 is split into four parts, namely A, B, C and D. Observe that, a disk of radius 1 centered at any point in sub-cell A may cover (some or all) points in only 15 cells, numbered 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18 and 19. The same fact can be observed for the sub-cells B, C and D. We can further tighten the observation as stated below.

Observation 1 A single unit disk centered at a point inside a cell can not cover points in more than 14 cells simultaneously.

Proof. First we prove that a single unit disk centered at a point p in the cell A can not cover points in cell number 4 and 16 simultaneously (see Figure 1).

Let u and v be the bottom-left and top-right corners of the cells 4 and 16 respectively. Thus, dist(u, v) = 2, where dist(.,.) denotes the Euclidean distance between a pair of points. Let p be a point properly inside cell A. Therefore, dist(u, p) + dist(p, v) > 2. This implies that at least one of dist(u, p) and dist(p, v) is greater than 1. Therefore, the point p can not cover a point inside cell 4 and a point inside cell 16 simultaneously. Thus, a single unit disk at a point $p \in A$ can cover (some or all) points in cells numbered 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18 and 19, but it can not cover a point in cell 4 and a point in cell 16 simultaneously.

Similarly, it can be shown that

- a single unit disk centered at a point $p \in B$ can cover (some or all) points in cells numbered 2, 3, 4, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19 and 20, but it can not cover a point in cell 2 and a point in cell 20 simultaneously.
- a single unit disk centered at a point $p \in C$ can cover (some or all) points in cells numbered 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 22, 23 and 24, but it can not cover a point in cell 6 and a point in cell 24 simultaneously.
- a single unit disk centered at a point p ∈ D can cover (some or all) points in cells numbered 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19, 20, 22, 23 and 24 but it can not cover a point in cell 10 and a point in cell 22 simultaneously.

Thus, the observation follows. $\hfill \Box$

In our approximation algorithm, we select one point from each cell that contains at least one point. The stepwise description of the proposed method is given in Algorithm 1.

Theorem 1 The approximation factor of our algorithm is 14, and its running time is $O(n \log n)$.

Proof. Consider a disk in the optimum solution. By Observation 1, it can cover points in at most 14 cells.

Algorithm 1 MDS_14-FACTOR(P)

- 1: Input: Set *P* of points in a 2-dimensional plane.
- 2: **Output:** A Set $P^* \subseteq P$ such that the unit disks centered at points in P^* cover all the points in P.
- 3: Set $P^* \leftarrow \emptyset$.
- 4: Consider a partitioning of the plane into a grid whose each cell is of size $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$.

/* A grid cell (α, β) is said to be less than another grid cell (γ, δ) if and only if either $\alpha < \gamma$ or $\alpha = \gamma$ and $\beta < \delta^*/$

- 5: Consider a height balanced binary tree \mathcal{T} for storing the non-empty grid cells. Each element of \mathcal{T} is a tuple (α, β) indicating the indices of a non-empty cell. It is attached with any point $p_i \in P$ that lies in that cell (as the piercing point). For each point $p_i = (x_i, y_i) \in P$, we compute the indices of the grid cell $\alpha = \lceil \frac{x_i}{\sqrt{2}} \rceil$ and $\beta = \lceil \frac{y_i}{\sqrt{2}} \rceil$. If the tuple (α, β) is not in \mathcal{T} , we store it in \mathcal{T} and attach p_i with it. Otherwise (i.e., if (α, β) is in \mathcal{T}), we have nothing to do.
- 6: for (each node v of \mathcal{T}) do
- 7: Let p be the point attached to the node v. Set $P^* \leftarrow p$

8: end for

9: return P^*

But, we have chosen at most 14 different disks to cover those points. Thus, the approximation factor follows.

In order to justify the time complexity, we shall not construct the grid explicitly. We maintain a height balanced binary tree \mathcal{T} for storing the non-empty grid cells. The processing of each point requires only the checking of the corresponding grid cell in \mathcal{T} . After processing all the points in P, we need to visit \mathcal{T} for reporting the piercing points. Thus the time complexity result follows.

2.2 Improving the approximation factor to 4

We now show that we can have a 4-factor approximation algorithm by increasing the worst case running time. We partition the plane into a grid whose each cell is of size $\frac{3}{\sqrt{2}} \times \frac{3}{\sqrt{2}}$ as shown in Figure 2(a).

Lemma 2 The minimum piercing set of the unit disks centered at the points inside a grid cell of size $\frac{3}{\sqrt{2}} \times \frac{3}{\sqrt{2}}$ can be computed in $O(n^9)$ time.

Proof. Let us consider a grid cell of size $\frac{3}{\sqrt{2}} \times \frac{3}{\sqrt{2}}$. We use χ to denote the cell, and P_{χ} to denote the set of points inside this cell. We split χ into 9 subcells of size $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$ (in Figure 2(b) it is shown separately.). In order to get the minimum cardinality subset of P for



Figure 2: Proof of Lemma 2

piercing the disks centered at the points in P_{χ} , we need to identify the minimum number of points in P such that the disks centered at those points can cover all the points in P_{χ} . Note that, if all the 9 cells are non-empty, we need at most 9 disks to cover all the points in P_{χ} . The reasons are (i) the disk centered at any point inside a subcell covers all the points inside that subcell, and (ii) each non-empty subcell of χ can contribute one such point.

In order to cover the points P_{χ} , we need to consider the disks centered at the points in P that lie in χ and the shaded region around χ as shown in Figure 2(c). Let this set of points be Q_{χ} . We choose every point of Q_{χ} , and check whether the disk centered at that point covers all the points in P_{χ} . If it fails for all the points in Q_{χ} , then we choose each pair of points $p, q \in Q_{\chi}$ and test whether each point in P_{χ} lies inside at least one disk centered at p and q. If it fails again, we need to choose each triple of points of Q_{χ} and so on. Finally, we need to choose each tuple of 8 points from Q_{χ} and test whether each point in P_{χ} lies inside one of the disks centered to those 8 points. In each step, the checking needs O(n)time. Thus, these 8 steps needs in total $O(n^9)$ time in the worst case. If the 8-th step also fails, we choose one point in each cell arbitrarily, and put a disk centered at those 9 points. Thus, the time complexity of this optimal algorithm follows. The stepwise description of the method described in Lemma 2 is given in Algorithm 2. Next, we use the Algorithm 2 for designing Algorithm 3 for getting a 4-factor approximation result for the discrete piercing problem.

Algorithm	2	$OPT(\chi$	(P_{χ}, Q_{χ}))
-----------	----------	------------	------------------------	---

- 1: **Input:** The cell χ of size $\frac{3}{\sqrt{2}} \times \frac{3}{\sqrt{2}}$, set $P_{\chi} \subseteq P$ of points inside cell χ , and set $Q_{\chi} \subseteq P$ of points such that each unit disk centered at the points in Q_{χ} covers at least one point in P_{χ} .
- 2: **Output:** Set $P^* \subseteq Q_{\chi}$ such that the unit disks centered at points in P^* cover all the points in P_{χ} .
- 3: Set $P^* \leftarrow \emptyset$ and $flag \leftarrow false$.
- 4: for (k = 1, 2, ..., 8) do
- 5: Choose each k points from each Q_{χ} , and check whether the disks centered at these k points cover all the points in P_{χ} .
- 6: If the answer of the above step is true, then add these k points in the set P^* , set flag = true, and break the *for* loop.
- 7: end for
- 8: if (flag = false) then
- 9: Divide χ into 9 subcells of size ¹/_{√2} × ¹/_{√2}. Choose one point of P_χ from each of these 9 subcells and add these 9 points to the set P*.

10: **end if**

11: return P^*

Observe that a unit disk in the optimum solution of a cell χ does not cover any point of some other cell ψ unless ψ is one of the eight neighboring cells of χ . We color the cells with minimum number of colors such that the unit disks placed in the cells of same color are non-overlapping irrespective of which point is chosen (as the center of the disk) in those cells. Thus, if we color cell number 1 (top-left cell) of the grid by A, we need to assign three different colors, say B, C and D to its three neighboring cells numbered 2, 7 and 8, which in turn are neighbors to each other (see Figure 2). But, we can again assign color A to cell 3. Thus, we have the following result.

Lemma 3 The minimum number of colors required to color the cells of the grid is 4.

Proof. We assign color to the cells in the grid from top row to the bottom row, and the cells in each row are colored from left to right. While assigning color to a cell, at most three of its neighbors are already colored. These are all different since the corresponding cells are neighbor to each other. So, we may assign the remaining fourth color to it. \Box

Theorem 4 A 4-factor approximation algorithm for

the minimum discrete piercing set problem for unit disk exists with time complexity $O(n^9)$.

Proof. Consider the cells colored by A. Since the distance between any two cells with color A is at least $\frac{3}{\sqrt{2}}(>2)$, a unit disk can covers only the points in a single cell with color A. Therefore, the set of disks in the optimum solution of one cell colored with A do not cover any point in any other cell colored with A. Thus, the optimum solution of the points of all the cells colored with A can be computed by choosing each Acolored cell independently, and computing its optimum solution. Let us denote this solution by OPT_A . Surely $|OPT_A| \leq |OPT|$, where OPT is the optimum solution for the point set P distributed on the plane. Similarly OPT_B, OPT_C and OPT_D denote the optimum solution of the cells colored as B, C and D. The approximation factor of our algorithm follows from the fact that $|OPT_A| + |OPT_B| + |OPT_C| + |OPT_D| \le 4|OPT|$, and our reported answer is $OPT_A \cup OPT_B \cup OPT_C \cup OPT_D$. The time complexity follows from the fact that we are using at most O(n) points while computing the opti-

mum solution of a cell, and we are computing the optimum solution for only non-empty cells, which may be O(n) in the worst case.

2.3 Improving the approximation factor to 3

We now improve the previous method by reducing dependency between cells. As in the earlier sections, here also we need to partition the region into cells as follows. We split the plane into horizontal strip of width $\frac{3}{\sqrt{2}}$. Each odd numbered strip is divided into equal sized cells of width $\frac{6}{\sqrt{2}}$. The horizontal width of the last cell may be less than $\frac{6}{\sqrt{2}}$, depending on the horizontal width of the region. Each even numbered strip is divided into cells such that the first cell is of width $\frac{3}{\sqrt{2}}$, and the other cells are of width $\frac{6}{\sqrt{2}}$, excepting the last cell as mentioned for odd numbered strips. Next, we assign color to the cells of the odd numbered strips using three colors A, B and C as shown in Figure 3. Now consider the cells in the even numbered strips, say strip 2. Cell 8 shares sides of two cells 1 and 2, which are already colored by A and B. So, we can color cell 8 by C. By the same reason, cells 7 and 9 are colored by C and A. Thus, the cells of each odd numbered strips are colored using the sequence B, C, A, B, C, A, \ldots Such a coloring admits that no part of the disk centered at a point inside a cell of a particular color $i \in \{A, B, C\}$ will lie in another cell of the same color i.

The maximum number of disks (centered at points in P) required to cover all the points is a cell of size $\frac{3}{\sqrt{2}} \times \frac{6}{\sqrt{2}}$ is 18. Arguing as in Subsection 2.2, the worst case time

Algorithm 3 MDS_4 -FACTOR(P)

- 1: Input: Set P of points in a 2-dimensional plane.
- 2: **Output:** Set $P^* \subseteq P$ such that the unit disks centered at points in P^* cover all the points in P.
- 3: Set $P^* \leftarrow \emptyset$.
- 4: Consider a partitioning of the plane into a grid whose each cell is of size $\frac{3}{\sqrt{2}} \times \frac{3}{\sqrt{2}}$. /* A grid cell (α, β) is said to be less than another grid cell (γ, δ) if and only if either $\alpha < \gamma$ or $\alpha = \gamma$ and $\beta < \delta^*/$
- 5: Consider an height balanced binary tree \mathcal{T} for storing the non-empty grid cells. Each element of \mathcal{T} is a tuple $\chi = (\alpha, \beta)$ indicating the indices of a nonempty cell. It is attached with two sets namely, P_{χ} and Q_{χ} where $P_{\chi} \subseteq P$ is the set of points inside the cell χ and $Q_{\chi} \subseteq P$ is the set of points such that the disk centered at the points in Q_{χ} covers at least one point in P_{χ} . For each point $p_i = (x_i, y_i) \in P$, we compute the indices of the grid cell $\alpha = \lceil \frac{3x_i}{\sqrt{2}} \rceil$ and $\beta = \left\lceil \frac{3y_i}{\sqrt{2}} \right\rceil$. If the tuple (α, β) is not in \mathcal{T} , we store it in $\check{\mathcal{T}}$ with corresponding P_{χ} and Q_{χ} . Otherwise (i.e., if (α, β) is in \mathcal{T}), we just modify the sets P_{χ} and Q_{γ} .
- 6: for (each node v of \mathcal{T}) do
- 7:
- Run $\frac{3}{\sqrt{2}}$ -X $\frac{3}{\sqrt{2}}$ -OPT $(\chi, P_{\chi}, Q_{\chi})$ /* Algorithm 2 */ Let P_1^* be the output of the above algorithm. Set 8: $P^* = P^* \cup P_1^*$

9: end for

10: return P^*



Figure 3: Coloring of cells for 3-factor approximation algorithm

needed for computing the minimum number of disks required to cover the points of P in such a cell is $O(n^{18})$. Thus, we have the following result:

Theorem 5 A 3-factor approximation algorithm for the minimum discrete piercing set problem for unit disks exists with time complexity $O(n^{18})$.

3 Conclusion

We proposed constant factor approximation algorithms for a variation of the discrete piercing set problem for unit disks, where the points chosen for piercing the disks will be from the set of center points of the disks given for piercing. The most simple algorithm produces 14factor approximation result in $O(n \log n)$ time. We then improve the approximation factor to 4. Finally, we propose a 3-factor approximation algorithm, which is an improvement of the existing result by a factor of $\frac{5}{3}$ [20]. Though, the time complexity of our proposed 4- and 3-factor approximation algorithms are high, in actual scenario, they terminate very fast.

Finally, our algorithm can also be used to solve the *h*piercing problem for unit disks as defined in [20]. Following the same method as in [20], it can be shown that the result obtained using our method is no worse than $3(2^{h} - 1)$ -factor of the optimum solution of *h*-piercing problem. Thus, the result produced by our algorithm for the *h*-piercing problem is an improvement by a factor $\frac{5}{3}$ over the existing best known result [20].

References

- C. Ambuhl, T. Erlebach, M. Mihalak and M. Numkesser. Constant-factor approximation for minimum-weight (connect) dominating sets in unit disk graphs. Proc. of the APPROX-RANDOM, pp. 3–14, 2006.
- [2] G. Calinescu, I Mandoiu, P. J. Wan and A. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mobile Network Applications*, 9(2):101–111, 2004.
- [3] P. Carmi, M. Katz and N. Lev-Tov. Covering points by unit disks of fixed location. Proc. of the 18th. International Symposium on Algorithms and Computation, pp. 644–655, 2007.
- [4] P. Carmi, M. J. Katz and N. Lev-Tov. Polynomial-time approximation schemes for piercing and covering with applications in wireless networks. *Computational Geometry: Theory and Applications*, 39(3):209–218, 2008.
- [5] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003.
- [6] T. M. Chan and A. -A. Mahmood. Approximating the piercing umber of unit height rectangles. Proc. of the 17th. Canadian Conference on Computational Geometry, pp. 15–18, 2005.
- [7] B. N. Clark, C. J. Colbourn and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1):165–177, 1990.
- [8] F. Claude, G. K. Das, R. Dorrigiv, S. Durochar, R. Fraser, A. Lopez-Ortiz and B. G. Nickerson. An improved line-separable algorithm for discrete unit disk cover. *Discrete Mathematics, Algorithms and Applications*, 2(1):77–87, 2010.

- [9] M. R. Cerioli, L. Faria, T. O. Ferreira and F. Protti. On minimum clique partition and maximum independent set on unit disk graphs and penny graphs: complexity and approximation. *Electronic Notes in Discrete Mathematics*, 18:73–79, 2004.
- [10] G. K. Das, R. Fraser, A. Lopez-Ortiz and B. G. Nickerson. On the discrete unit disk cover problem. Proc. of the 5th. Workshop on Algorithm and Computation, LNCS-6552, pp. 146–157, 2011.
- [11] D. Dai and C. Yu. A $(5 + \epsilon)$ -approximation algorithm for minimum weighted dominating set in unit disk graph. Theoretical Computer Science, 410(8-10):756-765, 2009.
- [12] A. Dumitrescu and J. Pach. Minimum clique partition in unit disk graphs. Graphs and Combinatorics, 27(3):399–411, 2011.
- [13] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. Proc. of the 20th ACM Symposium on Theory of Computing, pp. 434–444, 1988.
- [14] T. F. Gonzalez. Covering a set of points in multidimensional space. *Information Processing Letters*, 40(4):181– 188, 1991.
- [15] Y. Huang, X. Gao, Z. Zhang and W. Wu. A better constant-factor approximation for weighted dominating set in unit disk graph. *Journal of Combinatorial Optimization*, 18(2):179–194, 2008.
- [16] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130– 136, 1985.
- [17] M. R. Garey and D. S. Johnson. Computers and Intractability – A Guide to the Theory of NPcompleteness. Freeman, New York, 1979.
- [18] D. Johnson. The NP-completeness column: an ongoing guide. Journal of Algorithms, 3(2):182–195, 1982.
- [19] M. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.
- [20] T. Nieberg and J. Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. Proc. of the 3rd International Workshop on Approximation and Online Algorithms, LNCS - 3879, pp. 296–306, 2006.
- [21] S. Narayanappa and P. Voytechovsky. An improved approximation factor for the unit disk covering problem. Proc. of the 18th Canadian Conference on Computational Geometry, 2006.
- [22] I. Pirwani and M. Salavatipour. A weakly robust PTAS for minimum clique partition in unit disk graphs. Proc. of the 12th Scandinavian Symposium and Workshops on Algorithm Theory, LNCS-6139, pp. 188–199, 2010.
- [23] F. Zou, Y. Wang, X. -H. Xu, X. Li, H. Du, P. Wan and W. Wu. New approximations for minimumweighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. *Theoretical Computer Science*, 412(3):198–208, 2011.

New Lower Bounds for the Three-dimensional Orthogonal Bin Packing Problem^{*}

Chia-Hong Hsu[†]

Chung-Shou Liao^{§,†}

Abstract

In this paper, we consider the three-dimensional orthogonal bin packing problem, which is a generalization of the well-known bin packing problem. We present new lower bounds for the problem and demonstrate that they improve the best previous results.

1 Introduction

The bin packing problem (abbreviated as 1D-BP) is one of the classic NP-hard combinatorial optimization problems. Given a set of n items with positive sizes $v_1, v_2, \ldots, v_n \leq B$, the objective is to find a packing in bins of equal capacity B to minimize the number of bins required. The problem finds obvious practical relevance in many industrial applications, such as the container loading problem and the cutting stock problem.

The bin packing problem is strongly NP-hard. Furthermore, it does not admit a $(\frac{3}{2} - \epsilon)$ -factor approximation algorithm unless P=NP [10]. On the other hand, it has been shown that the simple *First Fit* approach can obtain a $\frac{17}{10}$ -factor approximation algorithm, and the *First Fit Decreasing* algorithm can approximate within an asymptotic $\frac{11}{9}$ -factor [11]. Subsequently, Fernandez de la Vega and Lueker [9] proposed an asymptotic polynomial time approximation scheme (PTAS), and Karmarkar and Karp [12] presented an improved asymptotic fully PTAS. For further results on approximation algorithms, readers may refer to Coffman, Garey, and Johnson's survey [6].

There are many variations of the bin packing problem, such as the strip packing, square packing, and rectangular box packing problems. In this paper, we consider the three-dimensional orthogonal bin packing problem (abbreviated as 3D-BP). Given an instance I of n 3D rectangular items I_1, I_2, \ldots, I_n , each item I_i is characterized by its width w_i , height h_i , depth d_i , and volume $v_i = w_i h_i d_i$. The goal is to determine a non-overlapping axis-parallel packing in identical 3D rectangular bins with width W, height H, depth D, and size B = WHDthat minimizes the number of bins required. We assume that the orientation of the given items is fixed; that is, the items cannot be rotated and they are packed with each side parallel to the corresponding bin side.

A considerable amount of research has been devoted to the design and analysis of lower bounds for the bin packing problem [4, 16, 22]. Martello and Toth [19, 20] and Labbé *et al.* [14] proposed lower bounds for 1D-BP, and then extended the concept to multi-dimensional models [17, 18]. Fekete and Schepers [7, 8] devised lower bounds based on *dual feasible functions* (please see the Appendix) and several related results were presented [3, 5]. Boschetti [1] combined Martello and Toth's work with the above dual feasible functions and proposed the best lower bound for 3D-BP; i.e., the lower bound *dominates*¹ all the previous results for 3D-BP.

In the following sections, we first review the previously proposed lower bounds and integrate the best of them for 1D-BP and 3D-BP to obtain a new lower bound for 3D-BP. Then, we propose another novel lower bound for 3D-BP and show that it dominates all the previous results.

2 Lower bounds for 1D-BP revisited

An obvious lower bound for 1D-BP, called the *continu*ous lower bound, can be computed as follows:

$$L_0 = \left\lceil \frac{\sum_{i=1}^n v_i}{B} \right\rceil$$

It is known that the asymptotic worst-case performance ratio of the continuous lower bound L_0 is $\frac{1}{2}$ for 1D-BP [19]. The lower bound can be easily extended to 3D-BP by considering the volume v_i of each item I_i . Martello *et al.* [17] showed that, for 3D-BP, the worstcase performance ratio of L_0 is $\frac{1}{8}$.

Subsequently, the bound was improved by Martello and Toth [20]. Under the new bound denoted by L_1 , the set of items is partitioned into two subsets such that one contains the items whose size is larger than B/2, and the other contains the remainder. Since each item in the former subset needs one bin, at least $|V(B/2, B)|^2$

^{*}Supported by the National Science Council of Taiwan under Grant NSC99-2218-E-007-010.

[†]Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 300, Taiwan. [§]Corresponding Author: csliao@ie.nthu.edu.tw

¹For two lower bounds L_1 and L_2 of a minimization problem, L_2 is said to *dominate* L_1 , denoted by $L_1 \leq L_2$, if for any instance I, $L_1(I) \leq L_2(I)$, where L(I) is the value provided by a lower bound L for an instance I.

²For convenience, we define $V(a, b] = \{I_i \mid a < v_i \leq b\}$ and its cardinality as $\mid V(a, b] \mid$.

bins are required. In addition, only items of size v_i , $p \leq v_i \leq B - p$ are considered, where p is an integer with $1 \leq p \leq B/2$, because an item of size p cannot be placed in the same bin as an item whose size is greater than B - p. Hence, a valid lower bound L_1 can be computed if we allow the rest of the items (i.e., the items in V[p, B/2]) to be split. (The other rounding scheme of L_1 , denoted by $L'_1(p)$, is described in the Appendix.)

Labbé et al. [14] further improved L_1 , denoted as L_2 , by partitioning the set of items into three subsets (V(B/2, B], V(B/3, B/2], and V[p, B/3], where1) and applying the First Fit Decreasingalgorithm [6, 11, 13]. The procedure is implemented as follows. The items in V(B/2, B] are assigned to separate bins as L_1 . It may be possible to assign some of the items in V(B/3, B/2) to the open bins, and at most one item in V(B/3, B/2) can fit in any of the open bins. Thus, the open bins are sorted in nondecreasing order based on their residual space, and the items in V(B/3, B/2) are assigned in non-decreasing order of their size. The procedure proves that the items in V(B/2, B] and V(B/3, B/2] can be matched optimally in a pairwise manner. Let K be the subset of items in V(B/3, B/2) that cannot be matched through the above procedure. The items in K can be paired, so at least $\lceil K/2 \rceil$ bins are required. It follows that a valid lower bound L_2 can be obtained by allowing the items in V[p, B/3] to be split as follows.

$$L_{2} = |V(B/2, B]| + \lceil K/2 \rceil + \max_{1 \le p \le B/3} \{0, L_{2}(p)\}, \text{ where}$$
$$L_{2}(p) = \left\lceil \frac{\sum_{v_{i} \in V[p, B-p]} v_{i}}{B} - |V(B/2, B-p]| - \lceil K/2 \rceil \right\rceil$$

The lower bound L_2 can be obtained in O(n) time provided that the sizes of the items are given sorted. Bourjolly and Rebetez [2] proved that $L_1 \leq L_2$ (excluding the rounding scheme $L'_1(p)$), and that the asymptotic worst-case performance ratio of L_2 for 1D-BP is $\frac{3}{4}$. Note that the primal concept of Labbé *et al.* cannot be easily extended to a new lower bound L_{m-1} for 1D-BP by partitioning the set of items into m subsets³, even by using a brute-force approach.

3 Lower bounds for 3D-BP revisited

For 3D-BP, Boschetti [1] proposed a lower bound, denoted by L_B , which actually consists of three types of lower bounds: $L_B(p,q,r)$, $L'_B(p,q,r)$, and $L''_B(p,q,r)$. We discuss them in detail below. Note that no dominance relations hold between the three bounds.

$$L_B = \max_{\substack{1 \le p \le W/2 \\ 1 \le q \le H/2 \\ 1 \le r \le D/2}} \{L_B(p,q,r), L'_B(p,q,r), L''_B(p,q,r)\}$$

Boschetti [1] proved that L_B is currently the best lower bound for 3D-BP by applying L_1 to $L_B(p,q,r)$ and $L'_B(p,q,r)$, denoted by $L_{B,1}$. In this section, we first review $L_{B,1}$ by applying L_1 to $L_B(p,q,r)$ and $L'_B(p,q,r)$. Then, based on the proofs in [2] and [3], which show, respectively, that $L_1 \leq L_2$ and $L'_1(p) \leq f_2^p$ (the dual feasible function f_2^p is discussed in the Appendix), we integrate L_2 in [14] and f_2^p in [3] with L_B to obtain a better lower bound for 3D-BP, denoted by $L_{B,2}$, and show that $L_{B,1} \leq L_{B,2}$.

The lower bound $L_{B,2}(p,q,r)$. First, we consider the lower bound $L_B(p,q,r)$. Given an item $I_i = (w_i, h_i, d_i)$ for every i, we let $I^{W}(W-p,W) = \{I_i \mid W-p < V\}$ $w_i \leq W$, $I^H(H-q, H] = \{I_i \mid H-q < h_i \leq H\}$, $I^{D}(D-r,D] = \{I_i \mid D-r < d_i \le D\}, \text{ and } I[p,q,r] =$ $\{I_i \mid w_i \geq p, h_i \geq q, d_i \geq r\}$. The objective of $L_B(p,q,r)$ is to compute a valid lower bound for 1D-BP by using a simple rounding technique when considering the volume of each item in I[p, q, r]. For example, if $I_i \in I^W(W - p, W]$ is placed in a bin, then it will occupy a volume equal to $Wh_i d_i$ since no items in I[p,q,r] can be packed side by side parallel to the width. Hence, let B = WHD and $L_B(p,q,r)$ be computed as a continuous lower bound by rounding the volume of each item v_i for every *i* to $v_i(p,q,r) = w_i(p)h_i(q)d_i(r)$ such that if $I_i \in I^W(W - p, W]$, i.e., $w_i > W - p$, then $w_i(p) = W$; otherwise, $w_i(p) = w_i$. If $I_i \in I^H(H-q, H]$, then $h_i(q) = H$; otherwise, $h_i(q) = h_i$. Similarly, if $I_i \in I^D(D-r, D]$, then $d_i(r) = D$; otherwise, $d_i(r) = d_i$. Note that it can be proved that this rounding technique is a dual feasible function [3, 8] (the so-called classic dual feasible function f_0^p ; please see the Appendix).

$$L_B(p,q,r) = \left\lceil \frac{\sum_{i=1}^n v_i(p,q,r)}{B} \right\rceil$$

Since $L_B(p,q,r)$ can be computed as a continuous lower bound for 1D-BP by considering the volume of each item, L_1 can be applied to $L_B(p,q,r)$ to obtain a valid lower bound, denoted by $L_{B,1}(p,q,r)$. By contrast, we apply L_2 and the dual feasible function f_2^p to $L_B(p,q,r)$ separately. We select the maximum of the two refined lower bounds, denoted by $L_{B,2}(p,q,r)$, and show that it is a valid lower bound and that it is not smaller than $L_{B,1}(p,q,r)$.

Lemma 1 $L_{B,2}(p,q,r)$ is a valid lower bound for 3D-BP, and it dominates $L_{B,1}(p,q,r)$.

Proof. Based on the above rounding scheme, each item in V(B/2, B] that is rounded, say w_i is rounded to W if

³Scholl *et al.* [21] showed that the lower bound L_2 can be extended by considering the items in V(B/4, B/3], but the process is quite complicated and it does not have any obvious extension.

 $w_i > W - p$, has two other dimensions larger than H/2and D/2; otherwise, $v_i(p,q,r) \leq B/2$. Hence, the items in V(B/2, B] are assigned to separate bins.

Consider the items in V(B/3, B/2]. Assume we fit item I_i in V(B/3, B/2] in an open bin, and item I_j in V(B/2, B] is placed in the same bin. In addition, suppose the original dimensions of I_j are $w_j > W$ $p, h_j > H/2$, and $d_j > D/2$. Then, we only need to determine the height and depth of I_i since only the items in I[p, q, r] are considered. We have $h_i > H/3$ and $d_i >$ D/3 because $v_i(p, q, r) > B/3$. If $h_i < H/2$, it implies that $d_i > 2D/3$; similarly, if $d_i < D/2$, it implies that $h_i > 2H/3$. Thus, at most one item in V(B/3, B/2]can fit in any of the open bins. The rounded items in V(B/3, B/2] that can not be matched could not be matched originally either. Moreover, based on the above discussion, at most two items in V(B/3, B/2] can be paired. Hence, it is valid to apply L_2 to $L_B(p, q, r)$.

Furthermore, $L_{B,2}(p,q,r)$ is a valid lower bound for 3D-BP because f_2^p is a dual feasible function, where $1 \leq p \leq B/2$ and it can be applied directly to $L_B(p,q,r)$. Because $L_1 \leq L_2$ and $L'_1(p) \leq f_2^p$, $L_{B,2}(p,q,r)$ dominates $L_{B,1}(p,q,r)$.

The lower bound $L'_{B,2}(p,q,r)$. Regarding the lower bound $L'_B(p,q,r)$, as above, only the items in I[p,q,r]are considered. Let $I(W - p, H - q, D - r) = I^W(W - p, W] \cap I^H(H - q, H] \cap I^D(D - r, D]$. Obviously, |I(W - p, H - q, D - r)| is a valid lower bound and no items in I[p,q,r] can be placed in the open bins. Next, the items in $I[p,q,r] \setminus I(W - p, H - q, D - r)$, denoted by I'[p,q,r] are considered. The objective of $L'_B(p,q,r)$ is to consider items in terms of their width, height, and depth. Let the respective subsets be:

$$\begin{split} &I(p,H-q,D-r) = I^{H}(H-q,H] \cap I^{D}(D-r,D] \cap I'[p,q,r]; \\ &I(W-p,q,D-r) = I^{W}(W-p,W] \cap I^{D}(D-r,D] \cap I'[p,q,r]. \\ &I(W-p,H-q,r) = I^{W}(W-p,W] \cap I^{H}(H-q,H] \cap I'[p,q,r]; \end{split}$$

Any two items from the different subsets above can not be matched in the same bin. That is, the items in I(p, H - q, D - r), I(W - p, q, D - r), and I(W - p, H - q, r) can only be packed in separate bins. Thus, the items in I(p, H - q, D - r), I(W - p, q, D - r), and I(W - p, H - q, r) can be considered separately. For each dimension, a continuous lower bound of 1D-BP can be computed similarly. It follows that a valid lower bound $L'_B(p,q,r)$ can be derived as follows:

$$\begin{split} L'_B(p,q,r) &= \mid I(W-p,H-q,D-r) \mid + \\ & \left\lceil \frac{\sum_{I_i \in I(p,H-q,D-r)} w_i}{W} \right\rceil + \left\lceil \frac{\sum_{I_i \in I(W-p,q,D-r)} h_i}{H} \right\rceil + \\ & \left\lceil \frac{\sum_{I_i \in I(W-p,H-q,r)} d_i}{D} \right\rceil \end{split}$$

Since a continuous lower bound of 1D-BP can be computed for each dimension, Boschetti [1] applied L_1 to the lower bound $L'_B(p,q,r)$, denoted by $L'_{B,1}(p,q,r)$, in terms of the width, height, and depth. Our lower bound, denoted as $L'_{B,2}(p,q,r)$, is obtained by applying L_2 and f_2^p to $L'_B(p,q,r)$ separately and selecting the maximum of the two refined lower bounds. We show that $L'_{B,2}(p,q,r)$ is still a valid lower bound.

Lemma 2 $L'_{B,2}(p,q,r)$ is a valid lower bound for 3D-BP, and it dominates $L_{B,1}(p,q,r)$.

Proof. Without loss of generality, we consider the depth of each item in $I(W - p, H - q, r) = I^W(W - p, W] \cap I^H(H-q, H] \cap I'[p, q, r]$. Because $r \leq d_i < D-r$, the lower bound L_2 for 1D-BP can be used directly in terms of the depth of these items. This is similar to the width and height of the items in I(p, H - q, D - r) and I(W - p, q, D - r). Moreover, the dual feasible function f_2^p can be used directly for each dimension of the items. Hence, $L'_{B,2}(p,q,r)$ is a valid lower bound for 3D-BP. Because $L_1 \leq L_2$ and $L'_1(p) \leq f_2^p$, $L'_{B,2}(p,q,r)$ dominates $L'_{B,1}(p,q,r)$.

The lower bound $L''_{B,2}(p,q,r)$. The lower bound $L''_B(p,q,r)$, which is conceptually similar to $L_B(p,q,r)$, can be obtained by using another rounding technique proposed in [18]. The objective is to pack items into a bin like small rectangular boxes whose dimensions are p, q, and r, where $1 \leq p \leq W/2, 1 \leq q \leq$ H/2, and $1 \leq r \leq D/2$. The maximum number of small rectangular boxes that can be placed in a bin is |W/p||H/q||D/r|. Besides, every item is represented by small rectangular boxes whose dimensions are p, q, and r. Thus, for every i, the volume of each item v_i , can be rounded to $v'_i(p,q,r) = w'_i(p)h'_i(q)d'_i(r)$ such that, if $I_i \in I^W(W/2, W]$, then $w'_i(p) = \lfloor W/p \rfloor - \lfloor (W-w_i)/p \rfloor$; otherwise, $w'_i(p) = \lfloor w_i/p \rfloor$. If $I_i \in I^H(H/2, H]$, then $h'_{i}(q) = |H/q| - |(H-h_{i})/q|$; otherwise, $h'_{i}(q) = |h_{i}/q|$. Similarly, if $I_i \in I^D(D/2, D]$, then $d'_i(r) = |D/r| |(D-d_i)/r|$; otherwise, $d'_i(r) = |d_i/r|$. For each dimension, it can be proved that the rounding technique is a dual feasible function [3, 8]. More precisely, it is similar to the dual feasible function f_2^p except that $w_i = W/2$, $h_i = H/2$, and $d_i = D/2$. $L''_B(p,q,r)$ can be computed as a continuous lower bound as follows:

$$L_B''(p,q,r) = \max_{\substack{1 \le p \le W/2 \\ 1 \le q \le H/2 \\ 1 \le r \le D/2}} \left\{ \left\lceil \frac{\sum_{i=1}^n v_i'(p,q,r)}{\lfloor W/p \rfloor \lfloor H/q \rfloor \lfloor D/r \rfloor} \right\rceil \right\}$$

We let the size of a bin B be equal to $\lfloor W/p \rfloor \lfloor H/q \rfloor \lfloor D/r \rfloor$ and apply L_2 to $L''_B(p,q,r)$, denoted by $L''_{B,2}(p,q,r)$, and show that it is also a valid lower bound.

Lemma 3 $L''_{B,2}(p,q,r)$ is a valid lower bound for 3D-BP, and it dominates $L''_B(p,q,r)$. **Proof.** When L_2 is applied to $L''_B(p,q,r)$, the dimensions of the items in V(B/2, B] are larger than $\frac{1}{2}\lfloor W/p \rfloor$, $\frac{1}{2}\lfloor H/q \rfloor$, and $\frac{1}{2}\lfloor D/r \rfloor$. The width w_i of each item I_i with $w'_i(p) > \frac{1}{2}\lfloor W/p \rfloor$ is larger than W/2 originally. Similarly, $h_i > H/2$ and $d_i > D/2$. Hence, the items in V(B/2, B] are assigned to separate bins.

Consider each item I_i in V(B/3, B/2]. We have $w'_i(p) > \frac{1}{3} \lfloor W/p \rfloor$, $h'_i(q) > \frac{1}{3} \lfloor H/q \rfloor$, and $d'_i(r) > \frac{1}{3} \lfloor D/r \rfloor$, which implies that $w_i > W/3$, $h_i > H/3$, and $d_i > D/3$, because $v'_i(p,q,r) > B/3$. If item I_i can fit in an open bin, without loss of generality, there is one dimension of I_i , say $d'_i(r)$, that satisfies the condition $\frac{1}{2} \lfloor D/r \rfloor > d'_i(r) > \frac{1}{3} \lfloor D/r \rfloor$, which implies that $D/2 \ge d_i > D/3$. Furthermore, if $\frac{1}{2} \lfloor D/r \rfloor > d'_i(r)$, we have $w'_i(p) > \frac{2}{3} \lfloor W/p \rfloor$ and $h'_i(q) > \frac{2}{3} \lfloor H/q \rfloor$, which implies that $w_i > 2W/3$ and $h_i > 2H/3$ because $v'_i(p,q,r) > B/3$. Thus, at most one item in V(B/3, B/2] can fit in any of the open bins; and at most two items in V(B/3, B/2] can be paired.

On the other hand, we claim that if we can not fit I_i in some open bin, in which I_j in V(B/2, B] is placed, then I_j was not matched with I_i originally. More precisely, if $d'_j(r) + d'_i(r) > \lfloor D/r \rfloor$, then $d_j + d_i > D$. We know that $d'_j(r) = \lfloor D/r \rfloor - \lfloor (D - d_j)/r \rfloor$. Suppose that $d_i \leq D/2$. Then, we have:

$$\left\lfloor \frac{D}{r} \right\rfloor - \left\lfloor \frac{D - d_j}{r} \right\rfloor + \left\lfloor \frac{d_i}{r} \right\rfloor > \left\lfloor \frac{D}{r} \right\rfloor$$
$$\Rightarrow \left\lfloor \frac{d_i}{r} \right\rfloor > \left\lfloor \frac{D - d_j}{r} \right\rfloor$$
$$\Rightarrow \frac{d_i}{r} \ge \left\lfloor \frac{d_i}{r} \right\rfloor \ge \left\lfloor \frac{D - d_j}{r} \right\rfloor + 1 > \left(\frac{D - d_j}{r} - 1 \right) + 1$$
$$\Rightarrow d_j + d_i > D$$

We also know that $d_j + d_i > D$ if $d_i > D/2$; therefore, $L''_{B,2}(p,q,r)$ is a valid lower bound. Because $L_0 \leq L_2$, $L''_{B,2}(p,q,r)$ dominates $L''_B(p,q,r)$. \Box

Thus, we have the following new lower bound $L_{B,2}$ for 3D-BP:

$$L_{B,2} = \max_{\substack{1 \le p \le W/3 \\ 1 \le q \le H/3 \\ 1 \le r \le D/3}} \{ L_{B,2}(p,q,r), L'_{B,2}(p,q,r), L''_{B,2}(p,q,r) \}$$

The theorem follows immediately.

Theorem 4 $L_B \leq L_{B,1} \leq L_{B,2}$.

4 A new lower bound for 3D-BP

In this section, we extend the approach in [14] to 3D-BP and propose a novel lower bound, denoted by L_B^* . First

of all, we define some notations. Let $I^{W}(W/2, W] = \{I_i \mid W/2 < w_i \leq W\}, I^{H}(H/2, H] = \{I_i \mid H/2 < h_i \leq H\}$, and $I^{D}(D/2, D] = \{I_i \mid D/2 < d_i \leq D\}$. Similarly, let $I^{W}(W/3, W/2] = \{I_i \mid W/3 < w_i \leq W/2\}, I^{H}(H/3, H/2] = \{I_i \mid H/3 < h_i \leq H/2\}$, and $I^{D}(D/3, D/2] = \{I_i \mid D/3 < d_i \leq D/2\}$.

$$\begin{split} &I(W/2, H/2, D/2) = I^{W}(W/2, W] \cap I^{H}(H/2, H] \cap I^{D}(D/2, D]; \\ &I(W/3, H/2, D/2) = I^{H}(H/2, H] \cap I^{D}(D/2, D] \cap I^{W}(W/3, W/2]; \\ &I(W/2, H/3, D/2) = I^{W}(W/2, W] \cap I^{D}(D/2, D] \cap I^{H}(H/3, H/2]; \\ &I(W/2, H/2, D/3) = I^{W}(W/2, W] \cap I^{H}(H/2, H] \cap I^{D}(D/3, D/2]; \end{split}$$

We compute the new lower bound as follows. The items in $I(W/2, H/2, D/2) \cap V(B/3, B]$ are assigned to separate bins because each dimension of the items in I(W/2, H/2, D/2) is larger than half the size of its corresponding bin side. It may be possible to assign some of the items in $I(W/3, H/2, D/2) \cap V(B/3, B]$, $I(W/2, H/3, D/2) \cap V(B/3, B]$, and $I(W/2, H/2, D/3) \cap V(B/3, B]$ to the open bins; however, at most one item can fit in any of the open bins because only the items in V(B/3, B] are considered.

In addition, an item from the above three subsets can only fit in the open bins if one of its dimensions is smaller than half the size of the corresponding bin side; i.e., such an item can be only packed in the open bins in terms of its width, height, and depth. We then partition the $| I(W/2, H/2, D/2) \cap V(B/3, B) |$ open bins into two subsets so that one subset contains the open bins whose residual space is smaller than B/2, and the other contains the remaining bins. Note that the items in the first subset have at least two dimensions that are more than 2/3 of the size of the corresponding bin sides. Thus, the open bins in that subset can be divided into three parts based on the smallest dimension of their included items. Moreover, the bins in each part are sorted in non-increasing order based on the corresponding dimension. Therefore, the items in $I(W/3, H/2, D/2) \cap V(B/3, B], I(W/2, H/3, D/2) \cap$ V(B/3, B], and $I(W/2, H/2, D/3) \cap V(B/3, B]$ must be assigned in non-decreasing order separately in terms of their width, height, and depth. Similar to the proof of Labbé et al. [14], the procedure proves that the items are matched optimally in a pairwise manner.

Next, the second subset of open bins are sorted in non-decreasing order based on their residual space, and the items that cannot be matched are mixed and assigned in non-decreasing order according to their volume. Let $K^{HD} \subseteq I(W/3, H/2, D/2) \cap$ V(B/3, B] be the subset of items that cannot be matched through the above process. Similarly, let $K^{WD} \subseteq I(W/2, H/3, D/2) \cap V(B/3, B]$ and $K^{WH} \subseteq$ $I(W/2, H/2, D/3) \cap V(B/3, B]$ be the subsets of items that cannot be matched either. Note that any two items from the different subsets above can not be matched in the same bin because, without loss of generality, one dimension of each item I_i , say w_i , no larger than W/2 implies that $h_i > 2H/3$ and $d_i > 2D/3$. Hence, the items in K^{HD} , K^{WD} , and K^{WH} can only be paired separately, and at least $\lceil K^{HD}/2 \rceil + \lceil K^{WD}/2 \rceil + \lceil K^{WH}/2 \rceil$ bins are required.

Then, we consider the remaining items in I(p, H - q, D-r), I(W-p, q, D-r), and I(W-p, H-q, r). Similarly, any two items from the different subsets can not be matched in the same bin. Thus, the items can be only packed in terms of each dimension. First, the items are assigned to the above open bins by allowing the items to be split. Let I'(p, H-q, D-r), I'(W-p, q, D-r), and I'(W-p, H-q, r) be the subsets of items that cannot be packed in the $|I(W/2, H/2, D/2) \cap V(B/3, B]| + \lceil K^{HD}/2 \rceil + \lceil K^{WD}/2 \rceil + \lceil K^{WH}/2 \rceil$ open bins respectively. We compute a continuous lower bound of 1D-BP for each dimension. Finally, a valid lower bound can be obtained by allowing the rest of the items to be split as follows:

$$L_B^* = |I(W/2, H/2, D/2) \cap V(B/3, B]| +$$
(1)

$$\left\lceil \frac{K^{HD}}{2} \right\rceil + \left\lceil \frac{K^{WD}}{2} \right\rceil + \left\lceil \frac{K^{WH}}{2} \right\rceil + (2)$$

$$\left\lceil \frac{\sum_{I_i \in I'(p, H-q, D-r)} w_i}{W} \right\rceil + \tag{3}$$

$$\left\lceil \frac{\sum_{I_i \in I'(W-p,q,D-r)} h_i}{H} \right\rceil + \tag{4}$$

$$\left\lceil \frac{\sum_{I_i \in I'(W-p, H-q, r)} d_i}{D} \right\rceil + \tag{5}$$

$$\max_{\substack{1 \le p \le W/3 \\ 1 \le q \le H/3 \\ 1 \le r \le D/3}} \{0, L_B^*(p, q, r)\}, \text{ where } L_B^*(p, q, r) = \\ \left[\frac{\sum_{I_i \in I'[p, q, r]} v_i}{B} - \alpha + |I(W - p, H - q, D - r)|\right]$$

and $\alpha = (1) + (2) + (3) + (4) + (5)$.

We use the rounding scheme, i.e., the dual feasible function f_0^p for each dimension of every item I_i , to derive a rounded volume $v_i(p,q,r) = w_i(p)h_i(q)d_i(r)$. Next, we show that 1) L_B^* is a valid lower bound; and 2) after applying the rounding scheme f_0^p , L_B^* dominates $\max_{1 \le p \le W/3, 1 \le q \le H/3, 1 \le r \le D/3} \{L_{B,2}(p,q,r), L'_{B,2}(p,q,r)\}$.

Lemma 5 L_B^* is a valid lower bound.

Proof. The dimensions of each item in I(W/2, H/2, D/2) are more than half the size of the corresponding bin sides even if the item is rounded. Hence, the items in I(W/2, H/2, D/2) are assigned to separate bins.

Consider the items in $I(W/3, H/2, D/2) \cap V(B/3, B]$, $I(W/2, H/3, D/2) \cap V(B/3, B]$, and $I(W/2, H/2, D/3) \cap V(B/3, B]$. Without loss of generality, say we fit item

$$\begin{split} &I_i \in I(W/3, H/2, D/2) \cap V(B/3, B] \text{ into an open bin,} \\ &\text{and item } I_j \text{ in } I(W/2, H/2, D/2) \cap V(B/3, B] \text{ is placed} \\ &\text{in the same bin. } I_i \text{ may fit with respect to the width} \\ &\text{because } h_i(q) > H/2 \text{ and } d_i(r) > D/2 \text{ imply that } h_i > \\ &H/2 \text{ and } d_i > D/2. \text{ Besides, } w_i = w_i(p) \text{ because } W/2 \geq \\ &w_i > W/3. W/2 \geq w_i \text{ also implies that } h_i(q) > 2H/3 \\ &\text{and } d_i(r) > 2D/3 \text{ because } v_i(p,q,r) > B/3. \text{ Thus, if} \\ &h_i \text{ is rounded, then } h_i > H-q; \text{ otherwise, } h_i > 2H/3. \\ &\text{Similarly, } d_i > \min\{D-r, 2D/3\}. \text{ Because only the} \\ &\text{items in } I[p,q,r] \text{ are considered, at most one item in} \\ &\text{the above three subsets (every item } I_k \text{ in the subsets} \\ &\text{has } w_k > W/3, h_k > H/3, \text{ and } d_k > D/3) \text{ can fit in any} \\ &\text{of the open bins.} \end{split}$$

On the other hand, since I_i may fit (in terms of the width) into the bin in which I_j is placed, we need to consider if w_j is rounded (because $w_i = w_i(p)$). We know that the rounded w_j that can not be matched was not matched originally either. In addition, based on the above discussion, for item $I_i \in K^{HD}$, $W/2 \ge w_i > W/3$ implies that $h_i > \min\{H - q, 2H/3\}$ and $d_i > \min\{D - r, 2D/3\}$. Thus, two items from any two of K^{HD} , K^{WD} , and K^{WH} cannot be matched in the same bin; and at most two items from each subset can be paired.

Finally, similar to the lower bound $L'_B(p,q,r)$, we consider the remaining items in I(W - p, H - q, r), I(p, H - q, D - r), and I(W - p, q, D - r). The items are first assigned to the above open bins by allowing the items to be split. Then, we compute a continuous lower bound of 1D-BP for each dimension of the remainder. Thus, f_0^p can be applied to L_B^* , and L_B^* becomes a valid lower bound for 3D-BP by allowing the rest of the items to be split. \Box

Lemma 6 For each $1 \le p \le W/3$, $1 \le q \le H/3$, $1 \le r \le D/3$, L_B^* dominates $L_{B,2}(p,q,r)$ and $L'_{B,2}(p,q,r)$.

Proof. First we consider $L_{B,2}(p,q,r)$. Since f_0^p is applied to both $L_{B,2}(p,q,r)$ and our new lower bound L_B^* , we claim that the new partition scheme is better than Labbé *et al.*'s method. For the first part, we have $I(W/2, H/2, D/2) \cap V(B/3, B]$ open bins compared to V(B/2, B] bins. Every item $I_k \in V(B/2, B]$ has $w_k(p) > W/2$, $h_k(q) > H/2$, and $d_k(r) > D/2$; thus, $I_k \in I(W/2, H/2, D/2) \cap V(B/3, B]$. We have $V(B/2, B] \subseteq I(W/2, H/2, D/2) \cap V(B/3, B]$.

For the second part, each item $I_k \in V(B/3, B]$ has $w_k(p) > W/3$, $h_k(q) > H/3$, and $d_k(r) > D/3$. Besides, if one of the item's dimensions, say the width $w_k(p) \leq W/2$, it implies that $h_i(q) > 2H/3$ and $d_i(r) > 2D/3$. We have $V(B/3, B] \subseteq I(W/2, H/2, D/2) \cup I(W/3, H/2, D/2) \cup$ $I(W/2, H/3, D/2) \cup I(W/2, H/2, D/3)$. Therefore, $|V(B/2, B] | + \lceil K/2 \rceil \leq |I(W/2, H/2, D/2) \cap$ $V(B/3, B] | + \lceil K^{WH}/2 \rceil + \lceil K^{HD}/2 \rceil + \lceil K^{WD}/2 \rceil$. It is obvious that the remainder of $L_{B,2}(p, q, r)$ is no larger than the remainder of L_B^* . Thus, L_B^* dominates $L_{B,2}(p,q,r)$.

Consider the lower bound $L'_{B,2}(p,q,r)$. For the first part, since f_0^p is applied to L_B^* , we have $I(W - p, H - q, D - r) \subseteq I(W/2, H/2, D/2) \cap V(B/3, B]$. Regarding the second part, without loss of generality, say I(W - p, H - q, r) is considered in $L'_{B,2}(p,q,r)$. We explore the possibility of placing the items in $I(W/2, H/2, D/2) \cup I(W/2, H/2, D/3) \cup I(W - p, H - q, r)$ for the new lower bound L_B^* . Clearly, by considering each dimension, L_B^* dominates $L'_{B,2}(p,q,r)$.

Finally, similar to $L_{B,2}'(p,q,r)$, we apply the dual feasible function f_2^p to each dimension of all the items instead. Then, we compute the summation of the rounded volume of each item, and a continuous lower bound can be obtained by letting the size of a bin $B = \lfloor W/p \rfloor \lfloor H/q \rfloor \lfloor D/r \rfloor$. It is also valid to apply L_2 to this continuous lower bound, denoted by $L_{DF}^*(p,q,r)$. Then, we have:

$$L_{B,DF}^* = \max\{L_B^*, L_{DF}^*(p, q, r)\}$$

Because $L_{B,2}''(p,q,r) \leq L_{DF}^*(p,q,r)$, the next theorem follows immediately.

Theorem 7 $L_{B,2} \leq L_{B,DF}^*$.

5 Concluding remarks

We have considered the 3D-BP problem and proposed two new lower bounds $L_{B,2}$ and $L_{B,DF}^*$. In addition, we have demonstrated that the lower bounds improve the best previous results, and that $L_{B,DF}^*$ dominates all the other lower bounds for 3D-BP proposed in the literature. In our future research, we will continue to improve the non-oriented model, which allows items to be rotated.

References

- M.A. Boschetti. New lower bounds for the threedimensional finite bin packing problem. *Discrete Applied Mathematics*, 140:241–58, 2004.
- [2] JM. Bourjolly, V. Rebetez. An analysis of lower bounds procedures for the bin packing problem. *Computers & Operations Research*, 32:395–405, 2005.
- [3] J. Carlier, F. Clautiaux, A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers* & Operations Research, 34:2223-2250, 2007.
- [4] F. Clautiaux, C. Alves, J.V. de Carvalho. A survey of dual-feasible and superadditive functions. Annals of Operations Research, 179:317–342, 2010.
- [5] F. Clautiaux, A. Moukrim, J. Carlier. New datadependent dual-feasible functions and lower bounds for a two dimensional bin-packing problem. *International Journal of Production Research*, 47(2):537–560, 2009.

- [6] E.G. Coffman Jr., M.R. Garey, D.S. Johnson. Approximation algorithms for bin-packing: a survey. In D.S. Hochbaum (ed.) Approximation algorithms for NP-hard problems, 46–93, PWS Publishing, Boston MA, 1997.
- [7] S.P. Fekete, J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematics Pro*gramming, 91:11–31, 2001.
- [8] S.P. Fekete, J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60:311–329, 2004.
- [9] W. Fernandez de la Vega, G.S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355. 1981.
- [10] M.R. Garey, D.S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Co., New York, 1979.
- [11] D.S. Johnson. Near-optimal bin packing algorithms. Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.
- [12] N. Karmarkar, R.M. Karp. An efficient approximation scheme for the one-dimensional bin packing problem. In Proc. 23rd IEEE Annu. Found. Comp. Sci. (FOCS 82), 312–320, 1982.
- [13] B.H. Korte, J. Vygen. Combinatorial Optimization Theory and Algorithms (Chapter 18). Springer-Verlag, 2008.
- [14] M. Labbé, G. Laporte, H. Mercure. Capacitated vehicle routing on trees. Operations Research, 39:616–622, 1991.
- [15] G.S. Lueker. Bin packing with items uniformly distributed over intervals [a, b]. In Proc. 24th IEEE Annu. Found. Comp. Sci. (FOCS 83), 289–297, 1983.
- [16] A. Lodi, S. Martello, M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Op*erational Research, 141:241–252, 2002.
- [17] S. Martello, D. Pisinger, D. Vigo. The threedimensional bin packing problem. Operations Research, 48(2):256-267, 2000.
- [18] S. Martello, D. Vigo. Exact solution of the twodimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.
- [19] S. Martello, P. Toth. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Chichester, U.K., 1990.
- [20] S. Martello, P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.
- [21] A. Scholl, R. Klein, C. Jürgens. BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24:627–645, 1997.
- [22] S.S. Seiden, R. van Stee New bounds for multidimensional packing. Algorithmica, 36:261–293, 2003.

The 2×2 Simple Packing Problem

André van Renssen*

Bettina Speckmann[†]

Abstract

We significantly extend the class of polygons for which the 2×2 simple packing problem can be solved in polynomial time.

1 Introduction

We study the 2×2 simple packing problem: Given a simple rectilinear grid polygon P consisting of n edges and containing N grid cells, determine the maximum number of non-overlapping, axis-aligned 2×2 squares that can be packed into P [4]. The optimal solution is not necessarily unique with respect to the placement of the squares (Fig. 1). We, however, are interested only in the optimal number of squares.



Figure 1: A polygon and one of its optimal solutions.

El-Khechen [6] proved that there always exists an optimal solution such that all packed squares are aligned on the grid. Hence we consider only optimal solutions of this type. We say that placing a square in a certain location is optimal, if there exists an optimal solution such that a square is placed in that location. Similarly, we say that not placing a square in a certain location does not affect optimality, if there is an optimal solution, such that no square is placed in that location.

Previous work. For polygons with holes, the 2×2 simple packing problem is known to be NP-hard [1]. Recently, it was proven to be NP-complete [5]. When

the polygon does not contain holes it is not known whether the problem is NP-complete. An O(N) time 1/2-approximation algorithm is known to exist [2]. This algorithm can be modified to run in $O(n^2)$ time by performing a line sweep from bottom to top [8].

For polygons without holes there exists a PTAS [7] that runs in $O(k^2 N^{\frac{1}{\epsilon^2}}/\epsilon^2)$ time, where k is the size of the square (in our case k = 2) and $\epsilon > 0$. The algorithm has an error ratio of at most $(1+\epsilon)^2$. A faster PTAS [3] runs in $O(\hat{N} \log \hat{N} + \hat{N}\Delta^{\frac{1}{\epsilon}-1})$ time, where \hat{N} is the number of possible locations to place squares, Δ is the number of squares any point in the 2-dimensional plane can be in (in our case $\Delta = 4$), and $\epsilon > 0$. The algorithm has an error ratio of at most $1 + \epsilon$.

For a few classes of polygons, namely staircases, pyramids, and Manhattan skyline polygons, the problem is solvable in O(n) time [6]. These classes of polygons can be solved by filling them from bottom to top with squares placed in the corners.

Results. We describe a different approach: instead of placing squares, we determine where *not* to place squares. This allows us to solve a number of classes of polygons in $O(n^2)$ time. Our polygon classes are significantly larger than those previously solvable and include staircases, pyramids, and Manhattan skyline polygons.

In Section 2 we present some observations on parts of polygons that cannot contain squares. In Section 3 we consider the intersection graph of all possible squares and we present a number of rules to reduce this graph. In Section 4 we describe a class of polygons that can be solved based on these reduction rules. In Section 5 we describe a second class of polygons which can be solved more efficiently, without considering the intersection graph. Neither of these polygons classes is contained in the other. Additional details and extensions can be found in the Master's thesis of the first author [8].

2 Tight Corridors and Tails

The dual graph of a grid polygon P has a vertex for each cell of P and an edge between two vertices iff their corresponding cells share a border. A *tight corridor* of P corresponds to a path in the dual graph, where each vertex has degree at most 2 and no vertex is part of a 4-cycle. A *tail* is a tight corridor in which at least one vertex has degree 1. We prove in [8] that no cell of a tight corridor can be part of a square. We also show

^{*}School of Computer Science, Carleton University, Canada, avrensse@connect.carleton.ca This research was partially supported by NSERC.

[†]Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands, **speckman@win.tue.nl** Bettina Speckmann is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

how to remove these cells in O(n) time. Every solvable class of polygons can be extended by attaching tails.

3 The Intersection Graph

Instead of placing squares, we determine where not to place squares. For this, we use the *intersection graph* G = (V, E) of all possible squares (see Fig. 2). Each square is represented in G by a vertex located at the position of its center on the grid. Two vertices are connected by an edge iff their corresponding squares overlap. Clearly every vertex of G has degree at most 8. El-Khechen [6] already observed that solving the Maximum Independent Set Problem on G results in an optimal non-overlapping placement of squares.



Figure 2: The intersection graph G.

The number of vertices \widehat{N} of G is upper bounded by N, the number of cells in P. However, N can be far larger than \widehat{N} (for example, if P is a single tail). G can be constructed in $O(n \log n + \widehat{N})$ time using a simple sweepline algorithm.

3.1 Graph Reduction

Placing a square in a corner where both edges of the polygon P have length at least 2 is known to be optimal (for example, placing the square represented by v' in Fig. 2). We generalize this result, by noting that it does not depend on the edges of P, but rather on the sets of neighbors of the vertices of G. Each of the neighbors of v' excludes more vertices from the maximum independent set than v' does.

We define the *conflict set* of a vertex v, denoted by C(v), as all neighbors of v and v itself:

$$C(v) = \{u \mid (u, v) \in E\} \cup \{v\}$$

The conflict set of a vertex changes when one of its neighbors is removed from G. A vertex v is called *removable* when there exists a vertex v', a neighbor of v,

such that $C(v') \subseteq C(v)$. In other words, the conflict set of v is a superset of the conflict set of v'. In Fig. 2, the conflict set of vertex v consists of v_1 to v_4 , v', and v itself. The conflict set of v' consists of v_1 , v_2 , v, and v'. Since $C(v') \subseteq C(v)$, v is removable.

Lemma 1 Removing a removable vertex v does not affect optimality.

We refer to removing a removable vertex as the *superset rule*. Each vertex has degree at most 8, so we can check in constant time whether a vertex is removable. Since removing vertices from G changes the conflict set of neighboring vertices, vertices that were not removable at first can become removable later.

Next, we take a closer look at which vertices are affected when a removable vertex is removed. We already saw that only the neighbors of a removed vertex and their neighbors can become removable. A removable vertex v' remains removable when a removable vertex v is removed, unless C(v') = C(v).

Lemma 2 Given two removable vertices v and v' with $C(v') \neq C(v), v'$ remains removable after the removal of v.

To efficiently remove all removable vertices, we maintain them in a doubly-linked list L. A vertex can be marked as removable and it stores a pointer to its location in L. At each step, we remove the first vertex in L from G and update its neighbors and the neighbors of its neighbors: unmarked vertices that become removable are added to L and marked vertices that become not removable are removed from L. This way, all removable vertices can be removed in $O(\hat{N})$ time.

3.2 Cycles

Next, we discuss cycles of degree 2 vertices which might also contains some higher degree vertices. If a cycle Cconsists only of degree 2 vertices (i.e. it is not connected to the rest of G), the optimal solution contains exactly $\lfloor |C|/2 \rfloor$ vertices. For the optimality of the solution it does not matter which vertices we pick, as long as no two vertices are neighbors.

If a cycle C is connected to the rest of the graph $G \setminus C$, we can deal only with two special cases (see Fig. 3): (a) $G \setminus C$ is connected to a single vertex of C and (b) $G \setminus C$ is connected to two vertices go C, forming a triangle. If $G \setminus C$ is connected to only one vertex of C, we can remove that vertex, since for any pair of neighbors we can pick only one vertex. Note that there may be multiple edges between $G \setminus C$ and v, as long as no other vertices of C are connected to $G \setminus C$. Such single connecting vertices can also be avoided when there are multiple such vertices in C, as long as there is an odd number of vertices of Cbetween them.



Figure 3: Cycles that can be removed.

Next, we look at connections that form triangles. In Fig. 3 (b) vertices v_1 , v_2 , and v_3 form a triangle and vertices v_2 and v_3 are part of the cycle C. If C has odd length, at some point in C, there must be two adjacent vertices that are both not part of the solution. In this case, these vertices can be chosen to be v_2 and v_3 , without losing optimality.

If C has even length, we cannot apply this method, since there are no two adjacent vertices that are both not part of the solution. We note, however, that since vertices v_1 , v_2 , and v_3 form a triangle, picking any of them excludes the other two from the solution. Now, assume we pick vertex v_1 to be part of the solution. This excludes both v_2 and v_3 . Since C has even length, there exists an optimal solution such that one of the neighbors of v_2 and v_3 is not used. Without loss of generality, we assume that the remaining neighbor of v_2 (i.e. not v_1 or v_3) is not used. Now vertex v_1 can be replaced by vertex v_2 , without violating the property that no two adjacent vertices are part of the solution. Thus, vertex v_1 can be removed without affecting the optimality of the solution. Moreover, since v_1 may be connected to other vertices in G, not picking it is potentially better.

3.3 Diamonds

A *diamond* is a rectangle having one cell removed from each of its corners. Its corresponding graph is a rectangular graph having a vertex removed from each corner (see Fig. 4). If the height of a diamond is odd, the diamond can be removed using the superset rule and the rules for removing cycles.

We now look at a special case: a diamond consisting of four vertices having other parts of the graph attached



Figure 4: The graph of a diamond.

to two opposite vertices. This configuration is shown in Fig. 5: vertices v_2 , v_3 , v_5 , and v_6 form a diamond and the other parts of the graph are attached to two opposite vertices v_2 and v_6 . The rule states that v_3 and v_5 can always be picked without affecting optimality. In other words, v_2 , v_4 , and v_6 can always be removed. For this rule to be applicable v_4 may be present, but this is not required. Also, there may be other vertices connected to v_2 to v_6 , as long as they are not connected to v_3 , v_4 , and v_5 . It is also possible that there are only vertices connected to v_2 (or only to v_6).



Figure 5: The configuration of a diamond.

Lemma 3 Picking vertices v_3 and v_5 from a diamond is optimal.

Note that we do not require that there is only one vertex on the chains between v_2 and v_6 . We require only that there is an odd number of vertices on these chains. Hence, we can generalize the rule to include every diamond that satisfies this property.

3.4 Remarks

The reduction rules described in this section require that specific configurations are found in the graph G. All configurations can be found in $O(\hat{N})$ time. Since resolving any configuration removes vertices, these rules can be applied at most a linear (in \hat{N}) number of times, hence applying all rules (including the superset rule) until no rule is applicable takes $O(\hat{N}^2)$ time.

Finally, note that no rule uses the fact that the polygon P is simple. Hence all reduction rules can also be applied to graphs constructed from polygons with holes.

4 Polygons Solvable by Graph Reduction

In the previous section, we described a number of reduction rules. However, we did not relate these rules directly to polygons. In this section we characterize the polygons that can be solved using these rules.

To construct a solvable polygon, we start out with one of four classes of *initial polygons*: The first class is the class of staircases, pyramids, and Manhattan skyline polygons. The second class consists of polygons constructed by stacking the blocks shown in Fig. 6 (see Fig. 7 (a)). The blocks may be mirrored horizontally.



Figure 6: The various blocks. Each block starts out as an even height rectangle. Blocks (b) and (f) are required to have odd width. (a) A rectangle. (b)–(f) A single cell is removed from: (b) the lower right corner, (c) the upper right corner, (d) the upper and lower right corners, (e) the upper left and right corners, (f) the upper and lower right corners and an odd height column is removed from the upper left corner.

Note that it is allowed to stack multiple blocks next to each other on top of a wider block and that it is allowed to stack a wider block on top of multiple blocks. The third class consists of diamonds: rectangles having a single cell removed from each corner. The final class is the class of polygons constructed by starting with an arbitrary rectangle and attaching rectangles to its corners (see Fig. 7 (b)). Note that the attached rectangles may not cover an entire side of a rectangle and no two rectangles may be attached to the same corner.

Initial polygons of even height are *non-cascading poly*gons. A polygon is called non-cascading if its optimal placement can be constructed regardless of the polygons that are connected to it. In other words, its optimal solution does not influence and is not influenced by the polygons connected to it. We have to restrict the type of possible connections for some non-cascading initial polygons in order for them to remain non-cascading. In the case of non-cascading staircases, pyramids, and Manhattan skyline polygons, a polygon may be attached to any horizontal edge whose height (measured from the base) is even. For non-cascading diamonds, the horizontal edge to which the other polygon is connected may not be covered entirely unless this edge has length 2.



Figure 7: Constructing a polygon by: (a) stacking blocks, (b) attaching rectangles to the corners of rectangles.

Non-cascading polygons can be attached to any constructible polygon.

Rectangles of even height and width are special case of non-cascading polygons. Such rectangles can be attached anywhere. In particular, they can be attached to corners (see Fig. 8). The height and width of the rectangle need not be greater than the length of the edges of the corner, hence any corner can be extended this way.



Figure 8: A corner extension. Figure 9: A universal connector polygon.

We define an *universal connector polygon* to be a polygon that can be solved without affecting the connected polygons in any way. Any two polygons may be connected by means of a universal connector polygon. We distinguish two types of universal connector polygons. The first type is the tight corridor described in Section 2. Two polygons connected by a tight corridor correspond to two disconnected graphs that can be solved individually. The second type is a variation on the diamond: an even width rectangle of height 4 having even width rows of cells removed from each corner. The rectangles that are removed from the left corners must have the same width. The same must hold for the rectangles that are removed from the right corners (see Fig. 9). Universal connector polygons can be extended by attaching non-cascading polygons to any edge or by attaching even height and width rectangles to corners.

Any polygon can be extended by attaching tails to it (see Section 2).

Theorem 4 Every polygon constructed by the construction scheme of Section 4 can be solved using the graph reduction rules.

The class of polygons constructible using the above construction scheme is significantly larger than the class



Figure 10: A polygon constructed using the construction scheme. Dashed lines represent the borders of the building blocks.

of previously solvable polygons. An example polygon is shown in Fig. 10.

5 Polygons Solvable in Quadratic Time

In the previous section, we presented a construction scheme for polygons solvable in time polynomial in \hat{N} (the number of possible square locations) using the graph reduction method. In this section, we present a different class of polygons which does not require the construction of the intersection graph; the constructed polygons can be solved in $O(n^2)$ time. Though this new class has some overlap with the class of the previous section, neither is contained in the other.

The new construction scheme is very similar to the previous scheme. There are, however, some differences. Diamonds are replaced by diamonds having an arbitrary number of steps and the non-cascading diamonds are replaced by even height diamonds having an arbitrary number of steps. Furthermore, the extension of rectangles placed in corners has an additional requirement: the height and width of the rectangle may not be equal to the length of the edges of the corner.

To solve these polygons, we find specific configurations of edges in the polygon. These configurations will correspond to (parts of) the polygons constructed using the construction scheme. Since we also need to know which edge of the polygon is closest to another edge, we first define the distance between two horizontal edges (two vertical edges are treated analogously). Since we do not need the distance between a horizontal and a vertical edge, we define this to be infinite.

We define the *slab* of an edge e, denoted by slab(e), as the region bounded by e and two half-lines orthogonal to e starting at the two endpoints of e, such that the interior of the polygon intersects slab(e) in the immediate neighborhood of e. A slab contains an edge f if one of the endpoints of f lies in the interior or on the boundary of the slab or f intersects the slab.

Given two horizontal edges e and f of a rectilinear

grid polygon, the distance between these two edges is defined to be infinite when slab(e) does not contain fand slab(f) does not contain e, or when the two edges are connected by means of a single vertical edge. Otherwise, the distance is the difference between their respective y-coordinates.

We now define the edge e closest to another edge f as the edge that has the smallest distance to edge f. An edge can have multiple closest edges: if multiple edges have the smallest distance to an edge e, they are all part of the set of closest edges of e. The closest edges are needed to efficiently check whether we need to split the polygon during the removal of configurations.

The configurations are shown in Fig. 11. Here we describe two of these configurations in detail and analyze the number of squares that can be placed when removing them. Full details can be found in [8]. Some configurations can be solved by using other configurations in combination with tail removal. These configurations are shown to make it easier to see the correlation between the construction scheme and the configurations. When describing the configurations, we say that certain rectangles do not contain edges in their interior. Here a rectangle contains an edge if (a part of) the edge lies in the interior of the rectangle. Edges lying on the boundary of the rectangle are allowed. All configurations may be mirrored and rotated.

Configuration (a): A rectangular configuration C. The rectangle defined by the horizontal edge and the shortest vertical edge does not contain any edges of the polygon. The height h of C is the length of the shortest of the two vertical edges and the width w of C is the length of the horizontal edge. The height h needs to be strictly greater than 1. When C is removed, we add $\lfloor h/2 \rfloor \cdot \lfloor w/2 \rfloor$ squares. Note that if the shortest vertical edge has odd length, C is not removed completely: it is reduced to a single row.



Figure 11: The configurations.

Configuration (1): A universal connector configuration C. The two vertical edges have the same even length l and the y-coordinates of their endpoints are the same. The distance between the two edges is 4. Both edges are connected to two edges of length 1 that are directed towards the opposite edge. The rectangle defined by the two vertical edges does not contain any edges of the polygon. When C is removed, we add lsquares.

Next, we sketch how to use the configurations shown in Fig. 11 to solve the polygons constructed by the construction scheme. Staircases, pyramids, and Manhattan skyline polygons can be solved by repeatedly using configurations (a) and (g) on the base and the two vertical edges connected to it. The blocks shown in Fig. 6 can be solved by using the corresponding configurations: block (a) can be solved using configuration (a), block (b) using configuration (b), and so on. Diamonds having an arbitrary number of steps can be solved by using configuration (h). Polygons constructed by starting with an arbitrary rectangle and attaching rectangles to its corners can be solved by repeatedly using configuration (a) in combination with tail removal. For details see [8].

Since non-cascading polygons are special cases of the above polygons, the configurations used to solve them are the same. The even height and width rectangles that can be placed in corners can be solved using configurations (i), (j), and (k), depending on whether the edges of the corner are longer or shorter than the edges of the rectangle.

The tight corridors that can be used as universal connectors will be removed before the algorithm is applied to the polygon. Tight corridors formed during the removal of configurations will be removed as soon as they are formed. The diamonds that are used as universal connector polygons can be solved using configuration (l). Finally, tails will also be removed before and while the algorithm is applied to the polygon.

Solving the polygons constructed by this scheme now becomes quite simple: we find one of the configurations, fill it, and continue with the remainder of the polygon. To keep the algorithm this simple, we need to ensure that the remaining part of the polygon is described by its edges. Hence we remove all tight corridors (causing the polygon to be split) and tails after each step.

Lemma 5 The polygon can be split at most n/4 - 1 times.

Lemma 6 A polygon can be split in O(n) time, while updating the closest edges.

Lemma 7 While solving the polygon, configurations that do not reduce the complexity of the polygon are used at most a linear number of times.

From Lemmas 5, 6, and 7 it follows that the configuration removal algorithm runs in $O(n^2)$ time.

Theorem 8 Every polygon constructed by the construction scheme of Section 5 can be solved in $O(n^2)$ time using the configuration removal algorithm.

6 Conclusion and Open Problems

We described a number of techniques that can be used to solve certain instances of the 2×2 simple packing problem on simple polygons. Our methods significantly extend the class of polygons for which the 2×2 simple packing problem is solvable in polynomial time. The graph reduction technique is polynomial in \hat{N} and the configuration removal technique runs in $O(n^2)$ time. Both techniques return the optimal number of squares for polygons constructed using their respective construction schemes.

The complexity status of the 2×2 simple packing problem remains open. Nevertheless, it is an interesting open question if a PTAS that runs in time polynomial in n (not just polynomial in \hat{N}) exists. Another challenging problem is to find an exact algorithm for classes of polygons that do not require construction schemes to describe them, such as the class of rectilinear convex simple polygons.

Acknowledgements. The authors would like to thank Prosenjit Bose for helpful comments.

References

- F. Berman, D. Johnson, T. Leighton, P. Shor, and L. Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2):153–184, 1990.
- [2] F. Berman, F. Leighton, and L. Snyder. Optimal tile salvage. Technical Report ADA119117, Purdue University, May 1982.
- [3] T. Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Let*ters, 89(1):19–23, 2004.
- [4] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problem project. http://maven.smith.edu/~orourke/TOPP/.
- [5] M. Dulieu, D. El-Khechen, J. Iacono, and N. van Omme. Packing 2×2 unit squares into grid polygons is NPcomplete. In Proc. 21st Canadian Conference on Computational Geometry, pages 33–36, August 2009.
- [6] D. El-Khechen. Decomposing and Packing Polygons. PhD thesis, Concordia University, April 2009.
- [7] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.
- [8] A. van Renssen. The 2×2 simple packing problem. Master's thesis, Technische Universiteit Eindhoven, 2010.

On covering of any point configuration by disjoint unit disks

Yosuke Okayama

Masashi Kiyomi

Ryuhei Uehara*

Abstract

We give a configuration of 53 points that cannot be covered by disjoint unit disks. This improves the previously known configuration of 55 points.

1 Introduction

In 2008, Japanese puzzle designer Naoki Inaba proposed an interesting question [3]: "Given any configuration of 10 points, prove that you can cover all the points by coins. You can use any number of coins, but coins cannot overlap." That is, he proved the following theorem:

Theorem 1 Any configuration of 10 points can be covered by disjoint coins.

Inaba gave an interesting proof of this theorem based on the probabilistic method. (See appendix; this proof is essentially the same in [4] written in Japanese. The proof can be found in [6] also.) As he mentioned in the answer page [4], this theorem also derives another natural question: How many points arranged appropriately cannot be covered by disjoint coins? Let k be the maximum number of points such that any configuration of k points can be covered by coins. (We note that k points can be covered by at most k coins.) Inaba's theorem shows that $10 \leq k$, and trivially there is an upper bound of k; if we put sufficiently many points on a fine lattice, disjoint coins cannot cover all of them (Figure 1). This problem spread over the puzzle society in 2010 (at the 9th Gathering 4 Gardner). Peter Winkler took up this problem in his column [5], and he gave a configuration of 60 points that cannot be covered by disjoint coins. Moreover, Peter Winkler improved the lower bound from 10 to 12 [6, 7]. That is, $12 \le k \le 59$. Recently, Veit Elser improved the upper bound to 54 in 2011 [2]. In this paper, we further improve the upper bound of k to 52. That is, we give a configuration of 53 points that cannot be covered by disjoint coins. The main theorem is summarized as follows.

Theorem 2 Let k be the maximum number such that any configuration of k points can be covered by disjoint coins. Then $12 \le k \le 52$.



Figure 1: Points cannot be covered by disjoint coins

Hereafter, we assume that each coin is a unit disk of radius 1. To simplify the argument, each unit disk is an open disk. That is, a point on the edge of a unit disk is not covered by the disk. (Using the perturbation technique, our results can be applied to closed disks.) Let L_3 , L_4 , and L_6 be a triangular, square, and hexagonal lattice, respectively. The *size* of a lattice is defined by the shortest distance between any pair of two points in L_i for i = 3, 4, 6 (Figure 2). We sometimes abuse L_i as a set of lattice points for i = 3, 4, 6. Our construction of the point configuration consists of two phases.

2 Configuration of the points in a circle

We first consider point configurations in a large circle. We denote by x a circle of radius $r = 2\sqrt{3}/3 - 1 = 0.1547...$ For the circle x, we have the following lemma:

Lemma 3 Let C_1 and C_2 be disjoint two unit disks. We suppose that a circle x circumscribes both of C_1 and C_2 . Then we cannot arrange any unit disk C_3 with $C_3 \cap x \neq \emptyset$ that is disjoint from C_1 and C_2 .

Proof. Since $r = 2\sqrt{3}/3 - 1$, when C_1 , C_2 , C_3 touch with each other, the circle x also touches all of them (Figure 3). Since C_1 , C_2 , C_3 are disjoint, the lemma follows immediately.

Using the circle x of radius r, we give the key idea of our point configuration:

^{*}School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan. {mkiyomi, uehara}@jaist.ac.jp



Figure 2: Triangular lattice, square lattice, and hexagonal lattice. Each size is given by the length of the arrow.



Figure 3: The circle x in the space surrounded by three unit disks



Figure 4: The size of each lattice



Figure 5: Proof of Lemma 4

Lemma 4 Let C^+ be a disk of radius 1 + 2r. For i = 3, 4, 6, let L_i be the lattice of size $\sqrt{3}r$, $\sqrt{2}r$, and r, respectively (Figure 4). (That is, we make x the largest empty circle of each L_i .) Then any point configuration in $L_i \cap C^+$ cannot be covered by disjoint unit disks.

Proof. We first observe that when we put a closed disk x' of radius r in $L_i \cap C^+$, x' should contain at least one point in $L_i \cap C^+$ because of the size of L_i .

Now in order to derive a contradiction, we assume that all the points in $L_i \cap C^+$ are covered by disjoint unit disks C_1, C_2, \ldots . Without loss of generality, $C_1 \cap C^+$ contains the largest number of points in C^+ among $C_i \cap C^+$ (Figure 5). Then we can put a circle x_1 of radius r in $C^+ \setminus C_1$ such that x_1 inscribes C^+ and circumscribes C_1 . Then, by the observation, x_1 contains at least one



Figure 6: Enlargement of the lattices L_4 and L_6

point p_1 in $L_i \cap C^+$. By the assumption, there is a disk C_2 covering the point p_1 in x_1 . Then we can put again a circle x_2 of radius r in $C^+ \setminus C_1$ such that x_2 circumscribes C_1 and C_2 , and x_2 contains a point p_2 in $L_i \cap C^+$. (Note that x_1 and x_2 may overlap.) Then, by Lemma 3, we cannot cover p_2 by the other unit disks C_3, \ldots This is a contradiction. Thus the lemma follows.

By Lemma 4, we can use $L_3 \cap C^+$ of size $\sqrt{3}r$, $L_4 \cap C^+$ of size $\sqrt{2}r$, and $L_6 \cap C^+$ of size r, where $r = 2\sqrt{3}/3 - 1$, as point configurations that give upper bounds of k, respectively. Among them, the upper bound k < 82 given by $L_3 \cap C^+$ is much better than the others (the leftmost one in Figure 2). For L_4 and L_6 , we can slightly enlarge the size of the lattices than that of Lemma 4 with careful analyses. For L_4 , when four points around on xin Figure 4, at most one point on x touches surrounding unit disks. Hence we can enlarge L_4 until at most three points of the square touch surrounding unit disks (Figure 6). (More precisely, we can enlarge to the minimum square of all the squares of which three points of it touch the surrounding disks.) For L_6 , we can enlarge L_6 in Figure 4 until all of 6 points are on surrounding unit disks as in Figure 6. However, these enlargements cannot catch up with the case of L_3 at all. Even using the enlargement technique, our best achievements of the cases of L_4 and L_6 are k < 102 and k < 119, respectively. (The point configurations after enlargements are given in Figure 2.) Hence we omit the details of these enlargements.

3 Improvement of the point configuration

Hereafter, we fix the lattice L_3 of size $\sqrt{3}r$. Carefully checking the proof of Lemma 4, we can see that C^+ is redundant. We first cut off the top and the bottom of C^+ as in Figure 7. More precisely, the lines ABand EF are straight line segments in parallel, and the distance between AB and the center of C^+ is equal to the distance between EF and the center of C^+ . The distance between AB and EF is 1 + 3r. The curves HA, BC, DE, and FG are arcs of the circles of radius r. The curves CD and GH are arcs of the circle C^+



Figure 7: The oval-like form Θ



Figure 8: Proof of Lemma 5

of radius 1 + 2r. Let Θ be the closed area surrounded by the resulting oval-like form ABCDEFGH. We now refine Lemma 4:

Lemma 5 Let Θ be the closed area given by the oval in Figure 7. Let L_3 be the lattice of size $\sqrt{3}r$. Then any point configuration in $L_3 \cap \Theta$ cannot be covered by disjoint unit disks.

Proof. In order to derive a contradiction, we assume that all points in $L_3 \cap \Theta$ are covered by disjoint unit disks C_1, C_2, \ldots . Without loss of generality, $C_1 \cap \Theta$ contains the largest number of points in $L_3 \cap \Theta$ among $C_i \cap \Theta$. Then we can put a circle x_1 of radius r in $\Theta \setminus C_1$ such that x_1 inscribes Θ and circumscribes C_1 (Figure 8). Then x_1 contains at least one point p_1 in $L_3 \cap \Theta$. By the assumption, there is a disk C_2 covering



Figure 9: A point configuration in Θ ; the circled points are in Θ .

the point p_1 . Then we can put again a circle x_2 of radius r in $\Theta \setminus C_1$ such that x_2 circumscribes C_1 and C_2 , and x_2 contains a point p_2 in $L_3 \cap \Theta$. Then, by Lemma 3, we cannot put any unit disk that covers p_2 . This is a contradiction. Hence the lemma follows.

Now we minimize the number of points in $L_3 \cap \Theta$, where L_3 has size $\sqrt{3}r$. Our best achievement is given in Figure 9. In this point configuration, we have two criteria for the points p_1, p_2, \ldots, p_6 in Figure 9.

- 1. The line ℓ_1 joining p_1 and p_2 and the line ℓ_2 joining p_3 and p_4 have enough distance to put Θ between them; the distance between ℓ_1 and ℓ_2 is equal to $5.5\sqrt{3}r = 5.5\sqrt{3}(2\sqrt{3}/3 1) = 5.5(2 \sqrt{3}) = 1.4737...$ On the other hand, the corresponding width of Θ is equal to $1 + 3r = 1 + 3(2\sqrt{3}/3 1) = 2\sqrt{3} 2 = 1.4641...$ Hence we can put Θ between ℓ_1 and ℓ_2 such that all the points on ℓ_1 or ℓ_2 are outside of Θ .
- 2. In Figure 9, the closest points on the right and left sides of Θ are p_5 and p_6 , respectively. We show that we can put Θ between them. To simplify the argument, we assume that we put Θ on the line ℓ_2 (joining p_3 and p_4) as in Figure 9, and we take the coordinate with the center O = (0,0) of the Θ . Let $p_5 = (x_5, y_5)$ and $p_6 = (x_6, y_6)$. Then we have $p_5 = (x_5, -7\sqrt{3}r/4)$, $p_6 = (x_6, -\sqrt{3}r/4)$, and $|x_5 x_6| = 33r/2 = 11\sqrt{3} 33/2 = 2.5525 \dots$ Let p'_5 be the point on the edge of Θ such that p'_5 has the same height of p_5 (and closest one of two such points). Let p'_6 be the point on the edge of Θ defined similarly for p_6 . That is, we can let $p'_5 = (x'_5, -7\sqrt{3}r/4)$, and $p'_6 = (x'_6, -\sqrt{3}r/4)$. Since $x'_i^2 + y'_i^2 = (1 + 2r)^2$ for i = 5, 6, we can obtain $|x'_5 x'_6| = \sqrt{115/(4\sqrt{3}) 725/48} + \frac{1}{2}$

 $\sqrt{283/48 - 29/(4\sqrt{3})} = 2.5302...$ Therefore, we can put Θ between p_5 and p_6 such that they are outside of Θ .

Based on these criteria, we can put Θ as in Figure 9, and the only circled points are in Θ . The number of the circled points is 53, and that concludes the proof of Theorem 2.

4 Concluding remarks

We give an upper bound of 52 for the maximum number k such that any configuration of k points can be covered by disjoint coins. In the oval Θ , it is essentially required that the radius of the largest empty circle is bounded by $r = 2\sqrt{3}/3 - 1$. Hence some computational power may improve the upper bound. But smart proofs seem to be better; recently, Aloupis developed another technique, and gave a better upper bound [1]. Applying his technique to the point configuration in Figure 9, it seems that we can remove a few more points. Our idea is based on the uniform point configurations. The upper bound based on some nonuniform point configurations would be interesting.

We still have a big gap between 12 and 52. Improvement of the lower bound is also interesting. In the appendix, we give the proof of the lower bound 10 by the probabilistic method. Indeed, the proof states a stronger result: any configuration of 10 points can be covered by the sheet in Figure 10. That is, the arrangement of the coins is fixed. Moreover, the bound given by the probabilistic method does not seem to be tight. Hence the gap between the lower bound and the real value seems to be larger than the gap between the upper bound and the real value.

Acknowledgements

The authors are grateful to Hirokazu Iwasawa and Naoki Inaba for their fruitful discussion on this topic. The authors also thank Peter Winkler, Veit Elser, János Pach, and Joseph Mitchell for their helpful comments.

References

- [1] G. Aloupis. Personal communication. 2011.
- [2] V. Elser. Packing-constrained point coverings. Geombinatorics, to appear.
- [3] N. Inaba. http://inabapuzzle.com/hirameki/ suuri_4.html. (in Japanese), 2008.
- [4] N. Inaba. http://inabapuzzle.com/hirameki/ suuri_ans4.html. (in Japanese), 2008.


Figure 10: A sheet of infinitely many coins

- [5] P. Winkler. Puzzled: Figures on a Plane. Communications of the ACM, 53(8):128, August 2010.
- [6] P. Winkler. Puzzled: Solutions and Sources. Communications of the ACM, 53(9):110, September 2010.
- [7] P. Winkler. Personal communication. 2011.

A Proof of Inaba's theorem by the probabilistic method

Let P be any configuration of 10 points p_1, p_2, \ldots, p_{10} . We put randomly a sheet of infinitely many coins arranged like Figure 10 on P. For $i = 1, 2, \ldots 10$, let A_i be the event that the point p_i is covered by a coin. Then, $\Pr\{A_i\} = (\sqrt{3} - \pi/2)/\sqrt{3} > 0.093$ by a simple calculation of ratios of areas of coins and the background. Hence the probability that all points are covered is given as follows:

$$\begin{aligned} \Pr\{A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_{10}\} \\ &= 1 - \left(\Pr\{\overline{A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_{10}}\}\right) \\ &= 1 - \left(\Pr\{\overline{A_1} \vee \overline{A_2} \vee \overline{A_3} \vee \dots \vee \overline{A_{10}}\}\right) \\ &\geq 1 - \left(\Pr\{\overline{A_1}\} + \Pr\{\overline{A_2}\} + \Pr\{\overline{A_3}\} + \dots + \Pr\{\overline{A_{10}}\}\right) \\ &> 1 - 10 \cdot 0.093 = 0.07 > 0. \end{aligned}$$

Since the all points are covered with positive probability, there exists a way to put the sheet to cover all the points.

Improving Accuracy of GNSS Devices in Urban Canyons^{*}

Boaz Ben-Moshe[†] Elazar Elkin[‡] Harel Levi[§] Ayal Weissman[¶]

Abstract

This paper addresses the problem of calculating the accurate position of a GNSS device operating in an urban canyon, where lines of sight (LOS) with navigation satellites are too few for accurate trilateration calculation. We introduce a post-processing refinement algorithm, which makes use of a 3D map of the city buildings as well as captured signals from all traceable navigation satellites. This includes weak signals originating from satellites with no line of sight (NLOS) with the device. We also address the dual problem - computing a 3D map of the city buildings when the position of the device is given. This is achieved by storing LOS/NLOS rays to all navigation satellites sampled at multiple locations within a region of interest (ROI). These rays are then used to compute the 3D shapes of buildings in the ROI.

A series of field experiments confirm that both algorithms are applicative. The position refinement algorithm significantly improves the device's accuracy and the mapping algorithm allows few users to map a complex urban region simply by walking through it.

1 Introduction

Receivers in Global Navigation Satellite Systems (GNSS) such as GPS, GLONASS or GALILEO tend to output inaccurate location estimations while operating in urban regions, mostly due to the density of tall buildings, which often block a receiver's line of sight (LOS) to the navigation satellites. Modern GNSS receivers are sensitive enough to receive the indirect signal reflected from the buildings. This *multi-path* effect is the major factor of poor performance of GNSS in urban canyons. A GNSS receiver approximates its position by interpolating the signal from each navigation satellite into a *pseudorange*, an approximation of the distance between

the receiver and the navigation satellite, obtained by multiplying the speed of light by the time needed for the signal to travel the distance. Using four *pseudoranges* and their associated satellite locations, the *GNSS* receiver location can be computed simply by intersecting the four spheres (see [2, 4] for more information regarding *GNSS* principles). Figure 5 presents an actual positioning error caused by wrong *pseudoranges* in an urban region.

In all but rare cases, a rule of thumb, which correlates a strong signal to the existence of LOS is proven very effective. Therefore, a GNSS device operating in a non-urban area would simply sort captured signals according to their strength, then use four (or more) strong enough signals to compute its location. Since a receiver wandering around at the country-side typically has LOSwith more than four satellites for most of its journey, the decisive majority of location computations in such areas are typically based on signals originating from LOSsatellites, for which *pseudoranges* tend to be accurate (the error range is typically within 2-5 meters).

In urban canyons, however, the situation is fundamentally different. It is very common for a GNSS device operating in an area of this sort (e.g. downtown Manhattan) to be surrounded by obstacles such as tall buildings, which block LOS with most, and infrequently all, otherwise available satellites. Since at least four strongenough signals, equivalent to four LOS satellites, are required for accurate positioning, the outcome of a narrowly available sky is inevitably a skewed computation, up to the point where the device is unable to perform its task.

Prior attempts to address limited LOS in urban areas, all of the while succeed to present reasonably-accurate results where satellites' signals are scarce and weak, are mostly based on approaches such as Map Matching (MM) [7, 10, 17, 18] and Dead Reckoning (DR) [4, 13]. A certain degree of improvement could arguably be obtained by assuming the GNSS receiver is located inside a car, which drives at some estimated speed on top of a road with a known path. The fact of the matter, however, is that most of these methods are evidently not sufficient in rough urban canyons, where lines of sight (LOS) can and do deteriorate up to the point where a receiver only captures multipath indirect reflections (zero LOS). In such circumstances, the decisive majority of GNSS devices become incompetent and cannot improve

^{*}This research was partially supported by the MAGNET program of the Israel Ministry of Industry and Trade - RESCUE consortium (patent pending 61/426,541).

[†]Department of Computer Science, Ariel University Center, Ariel 40700, Israel. benmo@g.ariel.ac.il

[‡]Department of Computer Science, Ariel University Center, Ariel 40700, Israel

[§]Department of Computer Science, Ariel University Center, Ariel 40700, Israel

[¶]Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900 Israel

accuracy using these methods.

The novelty of the GNSS - refinement method presented in this paper is based on two core concepts. The first is that unlike existing methods, which mostly rely on information external to the line of sight (LOS) problem, such as vehicle speed and road location, the discussed improvement is confronting the LOS problem in a more direct manner, by applying LOS-based algorithms. The second inventive aspect is an effective leverage of supposedly-useless weak signals originating from no line of sight (NLOS) satellites. By combining captured signals' strength with shading algorithms on a region of interest's 3D map, our improved GNSS device is able to determine with a high degree of assurance in which parts of the region of interest (ROI) it could potentially be, and likewise, in which parts of the ROI it is certain not to be, thus significantly narrowing the problem's error range.

The above, however, merely segments a ROI into "can-be" and "cannot-be" partial regions. Therefore, to address the general case, in which intersecting the satellites' binary LOS maps yields more than one "can-be" region, we multiply each binary map with a "likelihood weight". These weights are from a continuous range, where each derives from the captured signal's strength of the respective satellite. We later discuss how summing weighted LOS maps for all satellites usually converges to a single "highest likelihood" location. We also explain the heuristics we use in case there are still several candidate locations subsequent to that summing procedure.



Figure 1: The Urban Canyon effect: In red, the *GPS* captured path. In blue, the actual path.

1.1 Related Work

Prior studies have shown that longer integration times and data wipe-off enable High Sensitivity GPS (HS-GPS) receivers to acquire and track signals at lower signal strengths [14, 9]. This increases satellite availability in weak signal environments, but in an urban canyon comes with a baggage of positioning errors resulting from signal cross-correlation, multipath and echo-only signals [14, 8]. Most attempts to improve GNSS devices' accuracy in urban canyons consider the typical in-vehicle situation. This narrows the estimation problem, since vehicles are generally restricted to travel on roads. Nevertheless, GNSS and other absolute positioning systems do not inherently locate vehicles onto roads [12, 15]. The process of coinciding the output of a sensor such as GPS with a road network map is called Map Matching (MM) and is often integrated with Dead Reckoning (DR), which is the process of estimating one's current position based upon a previously determined position [3, 19].

Unfortunately, the problem's narrowing achieved by MM techniques is not sufficient in complex urban canyons. This is mainly because limited LOS in such areas frequently causes initial location estimates that are off by tens of meters. Such deviations are too large for MM techniques, which often leads to placements onto wrong distant roads.

In this paper we demonstrate how 3D maps of an area can be leveraged to acquire more information out of captured (both LOS and NLOS) GNSS signals. This additional information can then be used in conjunction with existing MM techniques, or as an alternative to such methods. Moreover, the concepts employed to narrow the estimation problem also form the basis of our algorithm for the dual 3D modeling problem (see section 3).

1.2 Paper Structure

Following an introduction and a related work review, we turn to a detailed discussion of the GNSS refinement algorithm. We then present a framework algorithm to the dual (inverse) problem, namely the computing of the city buildings' 3D maps by capturing the signal strength of all navigation satellites. Subsequently, we put the presented algorithms to the test and discuss results from field experiments conducted in rough urban canyons. We conclude with suggested future work.

2 GNSS Refinement Algorithm

2.1 Overview and Definitions

In this section we present the main algorithm for improving the GNSS receiver's accuracy in urban canyons. The algorithm transforms the signal strength of each traceable navigation satellite into a LOS/NLOS value. This value is not boolean but a continuous value in the range of [0, 1], representing the LOS clearance.

The algorithm's detailed description begins with formalization of (i) input parameters available from the GNSS device; (ii) some pre-defined constants (thresholds) and (iii) functions and data structures used throughout the refinement process (see Table 1). We then describe the mechanism by which the algorithm determines LOS status with each satellite. Subsequently, we explain the concepts behind LOS/NLOS partial maps and discuss the formation of an aggregated likelihood map out of them. We conclude the section with a high-level pseudo-code, which encapsulates the entire algorithm.

GNSS Algorithm Definitions						
	DeviceOutput: The current non-refined location					
Device	estimation, position error range, and the set					
	(S(t)) of all traceable satellite signals.					
	LastPosition: The last recorded refined loca-					
	tion, and the corresponding error ratio and con-					
	fidence level.					
	SigBench: A signal-strength threshold which					
Constants	determines visibility (assume LOS if higher:					
Competition	NLOS if lower).					
	MarSia: A surely visible signal-strength thresh-					
	old (used for linear transformation to [0.1]					
	range)					
	MinorErr: An error estimation threshold (don					
	correct if smaller)					
	$S(t) = S_1 \dots S_n$: The set of all satellite signals as					
	captured by the $GNSS$ receiver in time t.					
	$S_n(t) \subset S(t)$: The set of visible satellites (subset					
	of $S(t)$ obtained using SiaBench).					
Functions	$S_{u}(t) \subset S(t)$: The set of invisible satellites (sub-					
	set of $S(t)$ obtained using SigBench).					
	Map: A 2.5D representation of the terrain - in-					
	cluding both earth surface and buildings on top					
	of it.					
	<i>ROI</i> : The region of interest; a minimal polygon					
	which contains both: (i) all buildings which may					
	affect the user.(ii) all possible locations in which					
the user could potentially be.						
	$LOS(S_i, ROI)$: A binary shading function from					
each point of the ROI to a LOS. NLOS						
	S_i location.					
	$SIG(S_i, ROI, W_i)$: A refinement of $LOS(Si,$					
	ROI) to a continuous range using a signal					
	strength $(W_i \in [0, 1])$ weight.					
Approximate: The proposed GNSS refi						
	algorithm (i) Creates an aggregated likelihood					
	map from all $SIG(Si, ROI, W_i)$ (ii) Picks most					
likely location with respect to the par						
DeviceOutput and LastPosition. Aggregated: An aggregation ad-hoc 2D n with ROI's boundaries.						

Table 1: Formal definitions of parameters used throughout the proposed GNSS refinement algorithm.

2.2 Approximating Satellites' LOS Status

The strength of a signal as captured by GNSS-receiver depends on several factors ([5]):

- Global parameters: transmission frequency, transmission power these parameters are mostly fixed.
- Position and time: atmosphere and ionosphere condition, the angle between the satellite and the receiver.
- LOS and multi-path status: the nature of propagate signal with respect to the possible "radio path" to the receiver [16, 5, 6].

For a given GNSS (e.g. GPS L1, L2), the global parameters are fixed. The position and time parameters can be approximated within a small error range, usually smaller then 5 dB. The typical LOS signal strength is at least 10dB stronger than the signal strength of a reflected signal (NLOS). It is therefore rather simple to classify captured signals into $S_v(t)(LOS)$ and $S_u(t)(NLOS)$ subsets. Moreover, the field experiments we conducted (see section 4) demonstrate that determining a signal's LOS/NLOS status is applicable even in highly complex urban regions.

2.3 Partial LOS Map

Computing a shading map of a city building map (ROI)w.r.t. a satellite position can be done by projecting the buildings on the surface (the satellite position can be thought as in infinity). Our implementation encapsulates the LOS map of each captured signal S_i in S(t) as a 2D matrix filled with (0 and 1) binary values, each indicating whether the receiver is likely to have LOS with the corresponding satellite within a one square meter spot. The computation of the map's values is a relatively straightforward shading algorithm, which makes use of the satellite's position and the 2.5D Map of the area.

A fundamental, somewhat tricky, feature of the proposed algorithm concerns the leverage of weak multipath signals captured by the receiver to obtain meaningful information. A weak captured signal almost always indicates the absence of LOS with the respective satellite and is therefore classified (using SigBench) into the subset of invisible satellites $S_{\mu}(t)$. Knowing the receiver cannot see the satellite from its current location, we can conclude it is certainly not positioned in spots where this satellite can be seen. Derives from this observation is the applicability of using the complementary LOS maps (switching 1 and 0 values) of invisible satellites belonging to the subset $S_u(t)$ as likelihood layers, which are as informative as likelihood layers generated from visible satellite signals belonging to the $S_{\nu}(t)$ subset. Furthermore, since the discussed algorithm is targeted at urban canyons scenarios where LOS may deteriorate even to an empty $S_v(t)$ subset (all captured signals are weak), the complementary LOS maps of invisible satellites belonging to $S_u(t)$ become an essential source of information for the formation of the aggregated likelihood map discussed below.

2.4 Likelihood Weights

Likelihood weightening is a mechanism employed by the algorithm to improve determinism. In the absence of weights, which transform a LOS map from a binary LOS/NLOS representation into a more refined likelihood map, the algorithm could very well narrow the

problem by segmenting the *ROI* into "can be" and "cannot be" regions, but would not be able to resolve a scenario of multiple "can be" spots, and pick a single location to be presented on the *GNSS* device's screen. By introducing signal-strength derived weights, the *LOS* maps become differentiated from one another (each map is multiplied by a weight from a continuous range). This, in turn, results in (i) a further segmented aggregated likelihood map, which serves the algorithm's purpose of picking a single spot; (ii) improved accuracy and reliability, since greater importance is granted to stronger signals.

2.5 Aggregated Likelihood Map

The likelihood map is implemented as a 2D matrix filled with real numbers, each representing the likelihood of the receiver to be located within a location (e.g., within 1 square meter). The matrix is constructed by (a) multiplying each partial LOS matrix with a weight, which signifies the signal's strength of the corresponding satellite and (b) summing all partial weighted matrices. The outcome of this matrix addition is a single 2D matrix, where each point represents a likelihood, and the matrix's highest value(s) is the most likely spot. Since this likelihood matrix is being constructed by summing a considerable number (roughly 8 to 18) of partial (already differentiated by weights) matrices, the max value of the matrix tends to have a relatively small number of appearances. In the empirically common case of a single unique max value, the algorithm concludes and the spot represented by that max value is being presented. Otherwise, if several max values are encountered, the algorithm's Approximate function is employing heuristic methods (e.g. nearest point to the non-intervened receiver's output - DeviceOutput) to choose the point to be presented.



Figure 2: Construction of a likelihood map from LOS maps. In this example there are three satellites $(S_1 - S_3)$ and four points $(p_1 - p_4)$, p_1 and p_4 have the same aggregated shading maps (they both can see S_1 , and S_2). Yet p_2 and p_3 aggregated shading maps differ from the aggregated shading map of p_1 (or p_4).

2.6 Algorithm Formalization

Using the definitions at the beginning of this section (see Table 1), the refinement algorithm's high-level pseudo-

code would be:

Algorithm 1: High-Level pseudo-code for <i>GNSS</i> re-				
finement algorithm				
Result : RefinedPosition				
if PositionError < MinorErr then				
$_$ return <i>DevicePosition</i> ;				
Let $S(t)$ be the set of all satellite signals as				
captured by the $GNSS$ receiver in time t .				
Let $Aggregated$ be a $2D$ matrix with ROI 's				
boundaries (initialized with 0 values).				
for each S_i in $S(t)$ do				
Use SigBench to determine whether $S_i \in S_v$ or				
$S_i \in S_u$.				
Let W_i be S_i 's signal strength weight ¹				
$L_i \leftarrow SIG(S_i, ROI, W_i)^2$				
if $S_i \in S_u$ then				
L Inverse L_i ³				
$Aggregated \leftarrow Aggregated + L_i^4$				
Let <i>LikelyPoints</i> be the set of max values in				
Aggregated.				
Refined Position = Nearest point in $LikelyPoints$				
to DevicePosition.				
return Refined Position.				
${}^{1}W_{i}$ is transformed into [0, 1] range using MaxSig constant.				

 1W_i is transformed into [0,1] range using MaxSig constant. 2Compute partial weighted map.

³For an *NLOS* satellite : x = (1 - x) to each x in the matrix. ⁴Add partial weighted map to aggregated likelihood map.

3 3D Mapping Algorithm

3.1 Overview

In this section we address the problem of constructing a 3D building map using the strength of the signals from navigation satellites. This task is the dual problem to the improved accuracy: instead of using the 3D buildings' map to improve the receiver's position, we use its position to approximate the buildings' 3D map.

Most GNSS devices are able to keep detailed log files, which contain information about captured satellites' signals along a device's journey. Thereafter, it is a straightforward process to track down a device's path and the satellites' position respective to that device at each point in time during the journey (in sampling rate of 1-10Hz). This ability to track down signals, when magnified by a number of GNSS devices covering a subjected urban canyon, is a preliminary enabler of our novel framework algorithm to the dual problem, which is the generation of buildings' 3D models out of captured satellites signals.

The 3D mapping problem can be defined as follows: given a GNSS log-file, which contains samples of: time,

position, accuracy and the signal strength to each traceable satellite, convert the log file into two sets of 3Dvectors: (i) Blue vectors: all LOS signals. (ii) Red vectors: all the NLOS signals. The goal is to construct the surface, which will block all the vectors in the red set and will not block the vectors from the blue set.

3.2 2.5D Mapping Heuristics

We limit this algorithm to compute a 2.5D map representation, a surface of a terrain representation in which each (X, Y) location has a single Z value associated with it. For simplicity, we divide the mapping algorithm into two sequential steps: (i) computing the buildings' contours. (ii) approximating the height of each contour.

For the algorithm's first step we traverse timeconsecutive samples of the satellite signals. We then compute distances between consecutive samples, using a weighted xor function (assuming LOS is 1 and NLOS is 0), a change in satellite status (LOS/NLOS) contributes to the distance function according to the satellite angle (high-angle satellites contribute more). If the distance is above some threshold, we consider this point to be an *edge-point*. *Edge-points* tend to appear in buildings' corners and next to buildings' walls (due to the amplifying distance of high-angle satellites). We then filter out *edge-points* with potential high position error, and aggregate all the relatively accurate *edge-points*. Lastly, we compute contours which are bounded by the *edgepoints.* These contours may be general polygons or some constraint shapes (see Figure 6).

The algorithm's second step computes the z-value of each contour. The height value of each contour is bounded by all the LOS rays going over it. Yet because the LOS/NLOS data is "noisy" by nature, the actual z-value is computed as the height for which maximal weighted-constraints (LOS/NLOS) are satisfied. As in the first stage, the higher the ray-angle is (LOS/NLOS) the larger its weight becomes. Noteworthy is that samples from the same spot taken at different times of the day are beneficial for the algorithm. This is because for each triplet (spot, building, satellite) if there's a (time dependent) LOS ray from the spot to the satellite, which goes over the building's contour, then there exists a time of the day, which minimizes the vertical distance between that ray and the building's roof.

During our initial field experiments, the contours of the builds were slightly larger and shorter than in reality. In order to fix this effect we modified the algorithm to keep refining the 2.5D map by updating the contours according to the building approximated height.

4 Experimental Results

We conducted a set of preliminary experiments to evaluate the suggested algorithms in practice. In order to



Figure 3: 3D mapping algorithm in action. In red are the rays which are blocked (NLOS), in blue are the rays with LOS to the corresponding navigation satellites.

evaluate the improved accuracy algorithm two types of experiments were conducted (see Figures 3-5): (i) locating a GPS receiver in a fixed position for few hours (at each position). (ii) walking along a fixed route. In both experiments the actual position was compared to the suggested location of both (a) the GPS device (b) the refinement algorithm. In order to evaluate the 2.5Dmapping algorithm we have walked through the university campus and computed the approximated building map. The algorithm was implemented in Java and tested on both Android and Linux. The algorithm was able to refine the location (in an area of 200*200 meters) in less than 1 second, which validated its applicability to run efficiently on mobile devices (equipped with a 1Hz GPS). The following GPS receivers were used: Fastrax 1Hz, Wintec G-Rays 10Hz and the internal GPS of the Android devices. All tests were made while walking. The GPS row data was accessed via the NMEA protocol.



Figure 4: Improved position algorithm in action: In both above examples the *GPS* suggested a position with 12-18 meters error ratio. The refinement algorithm was able to fix the position to an error of less than 1 meter.



Figure 5: Improved Position: Left: above view. Right: 3D perspective view. In red, the path computed by a 1Hz fastrax GPS with an average error of 31 meters and max error of 180 meters; In green, the refined position with an average error of 4 meters and max error of 11 meters; In yellow, the actual path.



Figure 6: 2.5D mapping example, Left: the actual buildings, Middle: the 2.5D map of the buildings using axis parallel rectangles contours. Right: the 2.5D map of the buildings with no contour constraints

5 Conclusion and Future Work

We have proposed a new framework for improving a GNSS-receiver accuracy in urban regions. We have also presented an algorithm for constructing a 2.5Dbuilding map - using the receiver's position and the LOS status to it from each navigation satellite. In practical implementations, both algorithms could be running together - improved position accuracy assists in improving the 2.5D map accuracy, which in turn further improves positioning accuracy. Such approach for Simultaneous Localization And Mapping (SLAM) is proven to be an efficient method in many navigation and mapping tasks [1, 11]. The experimental results presented above show that even a basic implementation of the algorithms helps improving the GNSS accuracy significantly in urban canyons. Using few walking clients equipped with standard GPS devices we were able to construct a 2.5D buildings map of a complex downtown area; this map was then sufficient as the input source for the accuracy improvement algorithm. For future work we intend to use *GNSS-pseudoranges* to compute a more accurate map of the buildings. In particular, we would like to generalize the 2.5Dmapping algorithm into a real 3D mapping method.

Acknowledgment The authors wish to thank Eliyahu Ariel and Prof. Avner Kidar for introducing us to the real world of *GPS*.

References

- J. Artieda, J. M. Sebastián, P. Campoy, J. F. Correa, I. F. Mondragón, C. Martínez, and M. Olivares. Visual 3-d slam from uavs. *Journal of Intelligent and Robotic Systems*, 55(4-5):299–321, 2009.
- [2] S. Gleason and D. Gebre-egiabher. GNSS Applications and Methods. 2009.
- [3] J. S. Greenfeld. Matching GPS observations to locations on a digital map. In Proceedings of the 81th Annual Meeting of the Transportation Research Board, Washington D. C. 2002.
- [4] P. D. Groves. Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems. 2008.
- [5] B. M. Hannah. Modelling and simulation of gps multipath propagation. 2001.

- [6] Y.-W. Lee, Y.-C. Suh, and R. Shibasaki. A simulation system for gnss multipath mitigation using spatial statistical methods. *Comput. Geosci.*, 34:1597–1609, November 2008.
- [7] S. Liu, Z. Shi, M. Zhao, W. Xu, and K. Zhang. An urban map matching algorithm using rough sensor data. *Power Electronics and Intelligent Transportation Sys*tem, Workshop on, 0:266–271, 2008.
- [8] G. D. Macgougan. High sensitivity GPS performance analysis in degraded signal environments. *M. Sc. The*sis, UCGE Report No, page 20176, 2003.
- [9] B. Peterson, D. Bruckner, and S. Heye. Measuring GPS Signals Indoors, Proceedings of ION GPS-1997, The Institute of Navigation, 16-19 September, Kansas City, Missouri, USA. pp 389-398, 1997.
- [10] M. A. Quddus, W. Y. Ochieng, and R. B. Noland. Integrity of map-matching algorithms. *Transportation Research Part C: Emerging Technologies*, 14(4):283 – 302, 2006.
- [11] D. Schleicher, L. M. Bergasa, M. Ocaña, R. Barea, and E. L. Guillén. Real-time hierarchical gps aided visual slam on urban environments. In *EUROCAST*, pages 326–333, 2009.
- [12] C. Scott. Improved GPS Positioning for Motor Vehicles Through Map Matching, Proceedings of ION GPS-1994, The Institute of Navigation, 20-23 September, Salt Lake City, Utah, USA. pp 1391-1400, 1994.
- [13] J. Stephen and J. Stephen. Development of a multisensor gnss based vehicle navigation system, 2000.
- [14] S. Syed. University of calgary development of map aided gps algorithms for vehicle navigation in urban canyons, 2005.
- [15] G. Taylor and G. Blewitt. Virtual Differential GPS and Reduction Filtering by Map Matching, Proceedings of ION GPS-1999, The Institute of Navigation, 20-23 September, Salt Lake City, Utah, USA. pp 114-120, 1999.
- [16] N. Viandier, D. F. Nahimana, J. Marais, and E. Duflos. Gnss performance enhancement in urban environment based on pseudo-range error model. In *Position*, *Location and Navigation Symposium*, 2008 IEEE/ION, pages 377–382, 2008.
- [17] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6):91 – 108, 2000.
- [18] Y. Zhang and Y. Gao. A fuzzy logic map matching algorithm. Fuzzy Systems and Knowledge Discovery, Fourth International Conference on, 3:132–136, 2008.
- [19] L. Zhao, W. Y. Ochieng, M. A. Quddus, Noland, and R. B. An Extended Kalman Filter algorithm for Integrating GPS and low-cost Dead reckoning system data for vehicle performance and emissions monitoring. *The Journal of Navigation*, 56:257–275, 2003.

Geometry-Free Polygon Splitting

Sherif Ghali*

Abstract

A polygon splitting algorithm is a combinatorial recipe. The description and the implementation of polygon splitting should not depend on the embedding geometry. Whether a polygon is being split in Euclidean, in spherical, in oriented projective, or in hyperbolic geometry should not be part of the description of the algorithm. The algorithm should be purely combinatorial, or geometry free.

The geometry ultimately needs to be specified, and the geometric predicates can only be implemented after specifying the coordinate type and the number type. But the geometry, along with the coordinates and number type in that geometry, remain a late "plug-in", to be added only to the finished algorithm.

We describe a kernel for hyperbolic geometry. Once classes and predicates in that geometry are developed, hyperbolic geometry can be used as a plug-in to polygon splitting alongside other geometries.

We also describe an algorithm for the splitting of a polygon represented using its bounding lines. The use of this dual representation ensures that all predicates are computed directly from input data. This remains the case even if the same polygon is split multiple times, as occurs in BSP tree construction.

1 Introduction

Polygon clipping and splitting algorithms are described in the literature for a specific geometry. An algorithm is described either for Euclidean geometry [10] or for oriented projective geometry [17, 2]. Initially, intersections in oriented projective space were performed by making observations about the homogenizing coordinate, w. As kernels for different geometries were developed, it became better understood that intersection operations can be performed while the coordinates remain invisible [2, 15, 6].

Yet there is no reason for clipping and splitting algorithms not to be designed and implemented as purely combinatorial algorithms. The geometry remains a variable, one that is bound to the algorithm at a late stage during compilation.

The art of geometric computing has been scattered, with computational geometry mainly seeking solutions in Euclidean spaces and with computer graphics seeking ones in oriented projective space. Recent work has shown that geometric algorithms can be made neutral [5, 4]. The same algorithm can be instantiated in either Euclidean or oriented projective geometry. We take at present another step and show that a kernel for hyperbolic geometry can also be defined. We show how a geometry kernel can be a late addition to an algorithm to produce a concrete algorithm in that geometry.

The problem addressed here is polygon splitting—two parts result from the split. If only one of the two parts is needed, the problem is termed polygon clipping instead. Given a splitting algorithm, regardless of whether it is geometry free, one can easily produce a clipping algorithm by removing the algorithm subset that generates the part that is not needed.

1.1 Number-Type, Coordinate, and Dimension Freedom

By liberating an algorithm from its number type, coordinates, dimension, or from geometry, an algorithm becomes number-type free, coordinate free, dimension free, or geometry free, respectively.

Number-type freedom refers to the ability to modify an implementation by changing as little as one program line, to make the implementation operate on one number type or another [13]. Minimal modification is important. Modifying an algorithm to use 'float' instead of 'double', for example, can in general not be performed simply by replacing one string with another. One must also confirm that each instance does indeed represent a coordinate in the geometric system.

Coordinate freedom [11] is as important to writing maintainable geometric systems as number-type freedom. A geometric system is said to be *coordinate free* if coordinate manipulation is restricted. Coordinates are needed in the input and output stages of an algorithm, but the intermediate stages of an algorithm are designed and implemented such that coordinates are not accessed. Aside from the objectives of generality and reuse, coordinate freedom promotes the use of a vectorial language to resolve geometric predicates [5, Chap. 17].

Dimension freedom involves defining a geometric algorithm that can operate in any dimension. The only algorithms that appear to be amenable to dimension freedom at this time are BSP algorithms.

^{*}shghali@gmail.com

1.2 Geometry Freedom

Geometric freedom proposes to turn a geometric algorithm into a purely combinatorial one [5, Chap. 29]. Figure 1 illustrates a few low-dimensional geometries.



Figure 1: Low-dimensional kernels for Euclidean, spherical, projective, oriented projective, and hyperbolic geometries.

Consider that we have defined classes (datatypes and functions) for objects in each geometry. In the real Euclidean plane $\mathbb{R}E^2$ we will define classes for a point, a line, and a polygon—called, respectively, Point_E2, Line_E2, and Polygon_E2. Likewise in the real oriented projective plane $\mathbb{R}T^2$ we will define the classes Point_T2, Line_T2, and Polygon_T2, and so on.

Even though in a geometric system we have no need for creating a concrete instance of Euclidean, spherical, or hyperbolic plane geometry, we define a datatype for each geometry [4]. The datatypes remain abstract no instance is ever created. They serve in acting as a parameter to a combinatorial algorithm. During compilation the generic geometry is replaced by a concrete one, and the resulting implementation is as efficient as one hand-tailored for a particular geometry.

If 'double' is chosen as the number type, the class for 2D Euclidean geometry becomes Geometry_E2<double>, that for 2D hyperbolic geometry Geometry_H2<double>, and so on.

Computational geometry often uses mapping in general and projection in particular to reduce one problem to another. It is clear that a Euclidean geometry cannot replace a different geometry everywhere, but even if the topology is identical, the mapping may be undesirable. It is possible, for instance, to use stereographic projection to define a bijection between points on the extended complex plane and the Riemann sphere [9]. It then becomes possible to claim that a problem on the sphere can be solved by invoking an algorithm on the complex plane, along with appropriate handling for the ideal point. This may be satisfying in synthetic geometry, but it is not a useful solution from an algebraic or a computing perspective. No numerical precision would be adequate to capture points in proximity of the north pole.

2 Kernel Support for Polygon Splitting

As with any instance of introducing modularity into a software system, one must define the interface between two or more components. In the case of raising the abstraction of polygon splitting, we need to define the classes and the functions provided by the kernel and used by the implementation of polygon splitting.

The following C++ code illustrates the implementation of a 2D Euclidean geometry class. Itself parameterized by a number type NT, the class also acts as a parameter for geometry-free algorithms.

template<typename NT> struct Geometry_E2

typedef NT NumberType;

typedef Point_E2<NT> Point; typedef Line_E2<NT> Hyperplane; typedef Polygon_E2<NT> Polytope;

L			
r	,		

ł

The code for other geometries is similar. The following code shows a class Geometry_H2 for 2D hyperbolic geometry.

```
template<typename NT>
struct Geometry_H2
```

typedef NT NumberType;

typedef Point_H2<NT> Point; typedef Line_H2<NT> Hyperplane; typedef Polygon_H2<NT> Polytope;

};

The hyperbolic geometry kernel represents points by those in the interior of the Poincaré disk, where lines are oriented circles orthogonal to the unit circle [8]. Line intersection results in either no (real) points or in two points. In the first case the lines have no intersection in the hyperbolic plane and in the second they have one intersection. One point will be inside the unit disk and its inversion will be outside. A line joining two points will pass by the two points as well as their inversions. Adopting the Poincaré disk rather than Weierstrass coordinates [3] means that we sacrifice homogeneity, which we leave as a second step.



Figure 2: Separability of hyperbolic geometry

The only property of hyperbolic geometry on which polygon splitting depends is separability, illustrated in Figure 2. We say that a geometry is *separable* if the removal of one hyperplane results in two disjoint sets. Separability is also the property of oriented projective geometry that is not satisfied by classical projective geometry and that makes it necessary to base geometric algorithms on the former. We have yet to identify a class of algorithms that can be naturally defined in classical projective geometry.

Each geometry in turn defines the concrete types for a point, a hyperplane, and a polytope in that geometry using traits [12]. A geometry-free algorithm is then written to use a point, a hyperplane, and a polytope without referring to a concrete type [4].

Traits are simply type mappings. A classical procedure performs mapping between objects. The procedure takes a set of parameters. When called, it evaluates a function and returns an object. Traits extend this notion to types. The 'typedef' statement in the C language already performs this mapping, although in the opposite order of what assignment statements in that language would suggest: the "l-type-value" appears on the right. The combination of type genericity and type mapping meant that traits have found wide applications in generic programming.

The polygon splitting implementation is a function split that is parameterized by the geometry.

```
template<typename Geometry>
void
```

```
split(const typename Geometry::Polytope & polytope,
const typename Geometry::Hyperplane & hyperplane,
typename Geometry::Polytope & positive_part,
typename Geometry::Polytope & negative_part);
```

To split in a concrete geometry, it suffices to instantiate the generic function with a concrete geometry.

```
\label{eq:split} $$ split<Geometry_E2<$ double > (...); $$ split<Geometry_S2<$ double > (...); $$ split<Geometry_H2<$ double > (...);
```

Type safety is guaranteed. The function for splitting in one geometry will only accept polytope objects and a hyperplane in that geometry. In this abstraction we refer to polygons using the more general term polytope to facilitate dimension freedom.

Only one predicate function is needed by split: linepoint sidedness. Visualization requires a second function: line-line intersection. Neither function is generic with respect to the geometry. To compile split in a given geometry, It is necessary to ensure that a concrete intersection function and sidedness predicate are available for that geometry. The declarations in the case of Euclidean geometry, for instance, are:

3 Geometry-Free Polygon Splitting

3.1 A Dual Representation for Polygons

Our main application for polygon splitting is the computation of Boolean operations. A polygon is recursively split by the partitioning hyperplanes in a binary tree. When a fragment of the polygon reaches a leaf node, the Boolean operation is evaluated and the fragment is either discarded or used to construct a subtree at a leaf [18].

Suppose that the operation we wish to compute is Boolean union, and that we have inserted into an initially empty tree the three dark-shaded polygons shown in Figure 3 (a). Our BSP tree will at this time include some leaf node N representing the light-shaded triangle in the center. Suppose that we then insert a fourth polygon defined by the three circular markers and the dashed lines. That polygon will be split by the interior nodes, and all fragments but one will be eliminated as redundant (because they will be already flagged as belonging to the point set). Only the fragment at the center will remain.



Figure 3: (a) Computing the union of four polygons; (b) the boundary of the fragment remaining of the fourth polygon; (c) the corresponding subtree

The traditional approach is to then construct a binary tree (Figure 3 (c)) to represent the remaining fragment of the fourth polygon (Figure 3 (b)—we use the reverse of the convention of a polygon's orientation to facilitate dimension freedom [5]). That binary tree is attached as a subtree at the leaf node N. Yet this seven-node subtree introduces six nodes that represent empty sets. This is the case in a binary tree whenever the key at an interior node matches the key at one of its ancestors. In this case all three interior nodes of the subtree would be present along the path to the root. Storing nodes that represent the empty set does not breach the BSP tree—the empty sets are convex—but it is not optimal.

The conclusion we make is that the recursive splitting of a polygon must maintain for each edge of the polygon whether the edge is the result of a cut. Only those edges that are not the result of a cut are used to construct the leaf subtree.

As is now well-understood, it is necessary to use ternary logic for the sidedness predicate. Figure 4 illustrates the issue in the present context. If two polygons have sides that coincide with a splitting line (perhaps because they have already been cut by precisely that line), then the act of folding the coincidence of sidedness with either the positive or the negative sides will result in a zero-area quadrangle.



Figure 4: Necessity of handling point-hyperplane incidence: Folding 0 into - means that polygon (a) is unnecessarily split, and likewise for polygon (b) if 0 is folded into +.

But how can we determine reliably whether a given vertex or edge is incident to a splitting line? One approach is to off-load the problem on the number type and assume that we have at our disposal an exotic number type capable of determining without failure the sign of a determinant. Yet a solution is possible that depends on no stronger than the built-in finite precision number types.

The solution we use is to define polygons by their bounding lines rather than their bounding vertices [16]. Sugihara and Iri have also shown how the sidedness predicates can be resolved directly from the coefficients of the (hyper-) planes and without appealing to duality.

In addition, we also store with each bounding line a flag describing whether the line is the result of a split. Storing these flags ensures that the BSP trees we construct contain no nodes representing empty sets.

3.2 Algorithm

Figure 5 shows a new polygon splitter with the following new features. It is generic with respect to the geometry. It avoids slivers by using the input lines to define all new fragments. It is suitable for concave polygons, and it maintains a flag for each polygon boundary to identify whether it is the result of a cut, which makes the resulting fragments suitable for processing in BSP trees.

The main operation in the graphics pipeline is to clip a polygon six times with the boundary of a cube in oriented projective space. In that setting the hyperplane is three-dimensional, but the object clipped is a twodimensional polygon lying in 3D.

The present algorithm is not identical to the one that appeared in the monograph [5, Page 263]. That work also represented polygons using their bounding hyperplanes, but vertex coordinates were computed to determine sidedness—an operation that is at present resolved by using hyperplane-based predicates [16].

Both preceding works on polygon clipping [5, 1] avert the construction of new vertices at the clip/split locations. They both do so by outputting hyperplanes instead of the classical method of outputting vertices during each iteration. The atomic test in Bernstein and Fussell's algorithm is based on four boundary edges and three vertices, for a total of 27 cases (each vertex may lie on either side or coincide with the splitting plane). In addition to being purely combinatorial or geometry free, the present algorithm (as well as the previous monograph presentation [5]) iterates instead over three hyperplanes and two vertices while handling 9 cases (three outcomes for each vertex). For completeness, we show the algorithm handling both positive and negative fragments. We also handle the coincidence of the polygon with the splitting plane—a case that can arise in 3D.

The implementation was invoked in spherical, Euclidean, and hyperbolic geometries. Examples of splitting a polygon on the sphere, in the Euclidean plane, and in the hyperbolic plane are shown in Figure 6.

4 Future Work

The trajectory we take is to define algorithms and a usable library for vector computer graphics that complements what can be done in raster computer graphics. Rather than represent an image of a 3D object as a uniformly sampled shading value (a raster image), we wish to capture an image as a planar graph on either a sphere or on a subset of the Euclidean plane. The extension of this work to 3D must proceed while satisfying the following two simultaneous objectives. The present splitting routine must be usable when splitting a 3D polygon embedded in a plane in space. But the splitting function must also robustly handle the case of a 3D polytope, which is necessary for dimension freedom in a BSP tree. A splitting routine has also been implemented for 1D in both spherical geometry and in Euclidean geometry [5]. Even though simple, the need arises in practice for the computation of Boolean operations on regular sets in 1D.

Line clipping and splitting is an equally important problem. In the context of BSP trees we often need to determine *sub-hyperplanes* [5, 1], the subset of a splitting line lying inside a convex region—an instance of line clipping. Figure 7 suggests that it may be possible to use classical duality to combine an implementation for line and polygon splitting [14]. But note that, as illustrated in the figure, the vertices to be addressed under line and polygon splitting are distinct. The vertices addressed under line splitting are the line's intersections



construct negative_polygon from negative_lines and flags_of_negative_lines

Figure 5: Geometry-free polygon splitting—Cases (f) and (h) in the algorithm handle the incidence of a vertex with the splitting line, case (a) handles the incidence of an edge with the splitting line, and cases (b) and (c) handle the cases when the polygon lies strictly on one side of the splitting line. Cases (f) and (h) are themselves special cases of (e) and (g). Cases (i) and (j) handle a proper (interior) intersection between the polygon's boundary and the splitting line. If, as iterative runs of the algorithm guarantee, the input polygon represents a regular set, case (d) cannot arise in 2D geometry. We include it to be able to handle 3D polygon splitting.



Figure 6: Polygon splitting in Euclidean, spherical, and hyperbolic geometries

with the polygon's bounding lines.



Figure 7: Combining polygon and line clipping

Genericity can also be used in the context of attributes [7]. A polygon's vertex will frequently carry along data such as texture coordinates, which will also need to be clipped along with the geometry. Yet clipping texture coordinates is not as straight-forward as it may seem because of the distinct metrics in each geometry. The appropriate solution is to devise an interpolation module that caters for distances, angles, and areas. Geometry and dimension freedom suggest that the solution should handle interpolants of an arbitrary function, not just linear interpolation, in an arbitrary geometry.

Acknowledgment

I am grateful for the helpful comments made by reviewer #3.

References

- G. Bernstein and D. Fussell. Fast, exact, linear booleans. Comput. Graph. Forum, 28(5):1269–1278, 2009.
- [2] J. Blinn and M. Newell. Clipping using homogeneous coordinates. *Comput. Graph.*, 12(3):245–251, Aug. 1978.
- [3] H. Buseman and P. Kelly. Projective Geometry and Projective Metrics. Academic Press, 1953.
- [4] S. Ghali. Geometry-free geometric computing—towards higher-order genericity through purely combinatorial geometric algorithms. to appear.

- [5] S. Ghali. Introduction to Geometric Computing. Springer, 2008.
- [6] S. Ghali. Sense and sidedness in the graphics pipeline via a passage through a separable space. *The Visual Computer*, 25(4):367–375, Apr. 2009.
- [7] P. Heckbert. Generic convex polygon scan conversion and clipping. In A. Glassner, editor, *Graphics Gems I*, pages 84–86. Academic Press, 1990.
- [8] M. Henle. Modern Geometries: Non-Euclidean, Projective, and Discrete. Prentice-Hall, 2nd edition, 2001.
- [9] P. Henrici. Applied and Computational Complex Analysis. Wiley, 1974.
- [10] Y. Liang and B. Barsky. An analysis and algorithm for polygon clipping. CACM, 26(11):868–876, 1983.
- [11] S. Mann, N. Litke, and T. DeRose. A coordinate free geometry ADT. Technical Report CS-97-15, University of Waterloo, July 1997.
- [12] N. Myers. Traits: A new and useful template technique. C++ Report, June 1995.
- [13] J. Nievergelt, P. Schorn, M. de Lorenzi, C. Ammann, and A. Brüngger. XYZ : Software for geometric computation. Report 163, ETH, Zürich, July 1991.
- [14] V. Skala. A new approach to line and line segment clipping in homogeneous coordinates. *The Visual Computer*, 21:905–914, 2005.
- [15] J. Stolfi. Oriented Projective Geometry: A Framework for Geometric Computations. Academic Press, 1991.
- [16] K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. J. Inform. Proc., 12(4):380–393, 1989.
- [17] I. Sutherland and G. Hodgman. Reentrant polygon clipping. CACM, 17:32–42, 1974.
- [18] W. Thibault and B. Naylor. Set operations on polyhedra using binary space partitioning trees. *Comput. Graph.*, 21(4):153–162, 1987.

Robustness of topology of digital images and point clouds

Peter Saveliev*

Abstract

Such modern applications of topology as digital image analysis and data analysis have to deal with noise and other uncertainty. In this environment, the data structures often appear "filtered" into a sequence of cell complexes. We introduce the homology group of the filtration as a generalization of the homology group of a single cell complex. It is the group of all possible homology classes of all elements of the filtration with a certain equivalence relation. This relation equates the classes that represent the same homology class of the original data structure. The persistent homology group of the filtration is obtained similarly with an equivalence relation that equates the classes the differences of which falls outside of user's choice of the acceptable level of noise.

1 Introduction

Since Poincaré, homology has been used as the main descriptor of the topology of geometric objects. In the classical context, however, all homology classes receive equal attention. Meanwhile, applications of topology in analysis of images and data have to deal with noise and other uncertainty. This uncertainty appears usually in the form of a real valued function defined on the topological space. Persistence is a measure of robustness of the homology classes of the lower level sets of this function [6], [2], [4], [3].

Since it's unknown beforehand what is or is not noise in the dataset, we need to capture all homology classes including those that may be deemed noise later. In this paper we introduce an algebraic structure that contains, without duplication, all these classes. Each of them is associated with its persistence and can be removed when the threshold for acceptable noise is set. The last step can be carried out repeatedly in order to find the best possible threshold. The construction follows the approach to analysis of digital images presented in [8].

2 Backgound

The topological spaces subject to such analysis are cell complexes. A *cell complex* is a combinatorial structure that describes how k-dimensional cells are attached to each other along (k - 1)-dimensional cells. Cell complexes come from the following two main sources.

First, a gray scale image is a real-valued function f defined on a rectangle. Given a threshold r, the lower level set $f^{-1}((-\infty, r))$ can be thought of as a binary image. Each black pixel of this image is treated as a square cell in the plane. These 2-dimensional cells have to be combined with their edges (1-cells) and vertices (0-cells) while in the *n*-dimensional case the image is decomposed into a combination of 0-, 1-, ..., *n*-cubes. This process is called *thresholding*. The result is a cell complex K for each r, see [7].

Second, a point cloud is a finite set S in some Euclidean space of dimension d. Given a threshold r, we deem any two points that lie within r from each other as "close". In this case, this pair of points is connected by an edge. Further, if three points are "close", pairwise, to each other, we add a face spanned by these points. If there are four, we add a tetrahedron, and, finally, any d + 1 "close" points create a d-cell. The process is called the *Vietoris-Rips construction*. The result is a cell complex K for each r [6].

Next, we would like to quantify the topology of the cell complex K. It is done via the *Betti numbers of* K: B_0 is the number of connected components in K; B_1 is the number of holes or tunnels (1 for letter O or the donut; 2 for letter B and the torus); B_2 is the number of voids or cavities (1 for both the sphere and the torus), etc.

The Betti numbers are computed via homology theory [1]. One starts by considering the collection $C_k(K)$ of all formal linear combinations (over a ring R) of kcells in K, called k-chains. Combined they form a finitely generated abelian group called the chain complex $C_k(K)$, or collectively $C_*(K)$. A k-chain can be recorded as an N_k -vector, where N_k is the total number of k-cells in K. The boundary of a k-chain is the chain comprised of all (k - 1)-faces of its cells taken with appropriate signs. Then the boundary operator $\partial : C_k(K) \to C_{k-1}(K), k = 0, 1, ...,$ acts on the chain complex and is represented by a $N_k \times N_{k-1}$ matrix.

From the chain complex $C_*(K)$, the homology group is constructed by means of the standard algebraic tools. To capture the topological features one concentrates on *cycles*, i.e., chains with zero boundary, $\partial A = 0$. Further, one can verify whether two given k-cycles A and B are *homologous*: the difference between them is the

^{*}Department of Mathematics, Marshall University, USA, $\verb"saveliev@marshall.edu"$

boundary of a (k + 1)-chain $T : A - B = \partial T$ (such as two meridians of the torus). In this case, A and B belong to the same homology class H = [A] = [B]. The totality of these equivalence classes in each dimension k is called the k-th homology group $H_k(K)$ of K, collectively $H_*(K)$. Then, Betti number B_k is the rank of $H_k(K)$.

3 Prior work and outline

The methods for computing homology groups are well developed. In real-life applications however both digital images and point clouds may be noisy and one needs to evaluate the significance of their homology classes. The approach to this problem has been the following. Instead of using a single threshold and studying a single cell complex, one considers all thresholds and all possible cell complexes. Since increasing threshold r enlarges the corresponding complex, we have a sequence of complexes:

$$K^1 \hookrightarrow K^2 \hookrightarrow K^3 \hookrightarrow K^4 \hookrightarrow \dots \hookrightarrow K^s,$$

where the arrows represent the inclusions: $i^{n,n+1}$: $K^n \hookrightarrow K^{n+1}$. Let $i^{nm} : K^n \hookrightarrow K^m, n \leq m$, also be the inclusion. This structure $\{K^n, i^{nm}\}$ is called a *filtration*.

Now, each of these inclusions generates a homomorphism $i_*^{nm} : H_*(K^n) \to H_*(K^m)$ called the *homology* map induced by i^{nm} . As a result, we have a sequence of homology groups connected by these homomorphisms:

$$H_*(K^1) \to H_*(K^2) \to \dots \to H_*(K^s) \longrightarrow 0.$$

These homomorphisms record how the homology changes as the complex grows at each step. For example, a component appears, grows, and then merges with another one, or a hole is formed, shrinks, and then is filled. We refer to these events as *birth and death* of the corresponding homology class.

In order to evaluate the robustness of an element of one of these groups the *persistence of a homology class* is defined as the number of steps in the homology sequence it takes for the class to end at 0. In other words,

persistence = death date - birth date.

The *p*-persistent homology group of K^i is defined as the image of $i_*^{i,i+p}$. It is what's left from $H_*(K^i)$ after *p* steps in the filtration. Now the robustness of the homology classes of the filtration is evaluated in terms of the set of intervals [*birth*, *death*] representing the life-spans, called *barcodes*, of the homology classes [5].

Our approach is similar but more algebraic. It consists of two steps.

First stage: we pool all possible homology classes in all elements of the filtration together in a single algebraic structure (Sections 4 and 5). The presence of noise at this point is ignored. The homology group $H_*(\{K^n\})$ of filtration $\{K^n\}$ captures all homology classes in the whole filtration – without double counting. The latter is achieved by an equivalence relation that equates the classes that, in a sense, represent the same homology class in the filtration: $y = i_*^{n,n+1}(x)$.

Second stage: for a given positive integer p, the pnoise group $N_*^p(\{K^n\})$ is comprised of the homology classes in $H_*(\{K^n\})$ with the persistence less than p. Next, we "remove" the noise from the homology group of filtration by using the quotient (Sections 6 and 7):

$$H^p_*(\{K^n\}) = H_*(\{K^n\}) / N^p_*(\{K^n\}).$$

In other words: if the difference between two homology classes is deemed noise, they are equivalent. This is the persistent homology group of filtration. The second stage can be repeated as needed.

The (persistent) homology group of filtration is a graded group and is intended to stand for **the homology group of the data set that is behind the filtration**.

The main contribution of the present paper is an algebraic treatment of persistence that is alternative to the persistence module [3]. In the case of image analysis, the homology group of the image, unlike the barcodes, captures only the topology independent from the gray levels. This is why one might say that our approach provides a coarser classification of the homology of filtrations.

We also discuss the computational aspects of this approach (Section 8) and multiparameter filtrations (Section 9).

4 Motivation: the homology of a gray scale image

In this section we will try to understand the meaning of the homology of the gray scale image in Figure 1. For simplicity we assume that there are only 2 levels of gray in addition to black and white. A visual inspection of the image suggests that it has three connected components each with a hole. Therefore, its 0- and 1-homology groups should have three generators each. We now develop an algebraic procedure to arrive at this result.



Figure 1: A gray scale image and the corresponding filtration

First the image is "thresholded". The lower level sets of the gray scale function of the image form a filtration: a sequence of three binary images, i.e., cell complexes: $K^1 \hookrightarrow K^2 \hookrightarrow K^3$, where the arrows represent the inclusions. Suppose A_i, B_i, C_i are the homology classes that represent the components of K^i and a_i, b_i, c_i are the holes, clockwise starting at the upper left corner. The homology groups of these images also form sequences – one for each dimension 0 and 1.

Suppose F_1, F_2 are the two homology maps, i.e., homomorphisms of the homology groups generated by the inclusions of the complexes, with $F_3 = 0$ included for convenience. These homomorphisms act on the generators, as follows:

$$\begin{aligned} A_1 &\to A_2 \to A_3 \to 0, B_1 \to B_2 \to B_3 \to 0, \\ C_2 &\to C_3 \to 0, a_1 \to a_2 \to a_3 \to 0, \\ b_1 &\to 0, c_3 \to 0. \end{aligned}$$

To avoid double counting, we want to count only the homology classes that don't reappear in the next homology group. As it turns out, a more algebraically convenient way to accomplish this is to count only the homology classes that go to 0 under these homomorphisms. These classes form the kernels of F_1, F_2, F_3 . Now, we choose the homology group of the original, gray scale image to be the direct sum of these kernels:

$$H_0(\{K^i\}) = \langle A_3, B_3, C_3 \rangle, \ H_1(\{K^i\}) = \langle b_1, a_3, c_3 \rangle.$$

Thus the image has three components and three holes, as expected.

5 Homology groups of filtrations

In the following sections we provide formal definitions. All cell complexes are finite.

Suppose we have a one-parameter filtration:

$$K^1 \hookrightarrow K^2 \hookrightarrow K^3 \hookrightarrow \ldots \ \hookrightarrow K^s$$

Here K^1, K^2, \ldots, K^s are cell complexes, the arrows represent the inclusions $i^{n,n+1} : K^n \hookrightarrow K^{n+1}$, and so do $i^{nm} : K^n \hookrightarrow K^m, n \leq m$. We will denote the filtration by $\{K^n, i^{nm} : n, m = 1, 2, ..., s, n \leq m\}$, or simply $\{K^n\}$. Next, homology generates a "direct system" of groups and homomorphisms:

$$H_*(K^1) \to H_*(K^2) \to \dots \to H_*(K^s) \longrightarrow 0.$$

We denote this direct system by $\{H_*(K^n), i_*^{nm} : n, m = 1, 2, ..., s, n \leq m\}$, or simply $\{H_*(K^n)\}$. The zero is added in the end for convenience.

Our goal is to define a single structure that captures all homology classes in the whole filtration without double counting. The rationale is that if $x \in H_*(K^n), y \in$ $H_*(K^m), y = i_*^{nm}(x)$, and there is no other x satisfying this condition, then x and y may be thought of as representing the same homology class of the geometric object behind the filtration.

The homology group of filtration $\{K^n\}$ is defined as the product of the kernels of the inclusions:

$$H_*(\lbrace K^n \rbrace) = \ker i_*^{1,2} \oplus \ker i_*^{2,3} \oplus \ldots \oplus \ker i_*^{s,s+1}.$$

Here, from each group we take only the elements that are about to die. Since each dies only once, there is no double-counting. Since the sequence ends with 0, we know that everyone will die eventually. Hence every homology class appears once and only once.

These are a few simple facts about this group.

Proposition 1 If $i_*^{n,n+1}$ is an isomorphism for each n = 1, 2, ..., s - 1, then $H_*(\{K^n\}) = H_*(K^1)$.

Proposition 2 If $i_*^{n,n+1}$ is a monomorphism for each n = 1, 2, ..., s - 1, then $H_*(\{K^n\}) = H_*(K^s)$.

Proposition 3 Suppose $\{K^n, i^{nm}, n, m = 1, 2, ..., s\}$ and $\{L^n, j^{nm}, n, m = 1, 2, ..., s\}$ are filtrations. Then $H_*(\{K^n \sqcup L^n\}) = H_*(\{K^n\}) \oplus H_*(\{L^n\}).$

Proposition 4 Suppose $\{K^n, i^{nm}, n, m = 1, 2, ..., s\}$ and $\{L^n, j^{nm}, n, m = 1, 2, ..., s\}$ are filtrations and $f : K^s \to L^s$ is a cell map. Then the homology map of the homology groups of these filtrations $f_* : H_*(\{K^n\}) \to H_*(\{L^n\})$ is well defined as

$$f_*(x_1, x_2, ..., x_s) = (f_*^1(x_1), f_*^2(x_2), ..., f_*^s(x_s)),$$

where f^n is the restriction of f to K^n .

The stability of the homology group of a filtration follows from the stability of its persistence diagram, i.e., the set of points $\{(birth, death)\} \subset \mathbf{R}^2$ for the generators of the homology groups of the filtration, plus the diagonal. It is proven in [5] that $d_B(D(f), D(g)) \leq$ $||f - g||_{\infty}$, where d_B is the "bottle-neck distance" between the persistence diagrams D(f), D(g) of two filtrations generated by tame functions f, g. Function F(x, y) = y - x creates an analogue bottle-neck distance for the set of points $\{persistence\} \subset \mathbf{R}$ and its stability follows from the continuity of F.

6 Motivation: the high contrast homology of a gray scale image

To justify our approach to persistence, we observe that some of the features in the gray scale image in Figure 1 are more prominent than others. Specifically, some of the features have lower contrast. These are the holes in the second and the third rings as well as the third ring itself. By *contrast* of a lower level set of the gray level function we understand the difference between the highest gray level adjacent to the set and the lowest gray level within the set. An easy computation shows that the homology generators with persistence of 3 or higher among the generators are: A_1, B_1, a_1 . However, the set of the classes of high persistence isn't a subgroup of the homology group of the respective complex. Instead, we look at the classes with *low* persistence, i.e., classes that represent the noise. In particular, the classes in $H_*(K^1)$ of persistence 2 or lower form the kernel of F_2F_1 . We now "remove" this noise from the homology groups of the filtration by considering their quotients over these kernels. In particular, the 3-persistent homology groups of the image are:

$$H_0^3(\{K^i\}) = \langle A_1, B_1 \rangle / 0 = \langle A_1, B_1 \rangle, H_1^3(\{K^i\}) = \langle a_1, b_1 \rangle / \langle b_1 \rangle = \langle a_1 \rangle.$$

It is important that the output is identical to the homology of a single complex, i.e., a binary image, with two components and one hole. The way persistence is defined ensures that we can never remove a component as noise but keep a hole in it.

This approach to image analysis was tested with reallife images in [8].

Observe now that the holes in the second and third rings have the same persistence (contrast) and, therefore, occupy the same position in the homology group regardless of their birth dates (gray level). Second, if we shrunk one of these rings, its persistence and, therefore, its place in the homology group wouldn't change. These observations confirm the fact that the homology group of the gray scale image, unlike the barcodes, captures only its topology.

In the case of a Vietoris-Rips complex, not only the barcode, the interval [birth, death], but also the persistence, the number death - birth, of a homology class contains information about the size of representatives of these classes. For example, a set of points arranged in a circle will produce a 1-cycle with twice as large birth, death, and persistence than the same set shrunk by a factor of 2. However, persistence defined as death/birth will have the desired property of scale independence. The same result can be achieved by an appropriate reparametrizing of the filtration.

7 Persistent homology groups of filtrations

In the general context of filtrations the measure of importance of a homology class is its persistence which is the length of its lifespan within the direct system of homology of the filtration.

Given filtration $\{K^n\}$, we say that the persistence P(x) of $x \in H_*(K^n)$ is equal to p if $i_*^{n,n+p}(x) = 0$ and $i_*^{n,n+p-1}(x) \neq 0$. Our interest is in the "robust" homology classes, i.e., the ones with high persistence. However, the collection of these classes is not a group as it doesn't even contain 0. So we deal with "noise"

first. Given a positive integer p, the *p*-noise (homology) group $N^p_*(K^n)$ of $\{K^n\}$ is the group of all elements of K^n with persistence less than p.

Alternatively, we can define these groups via kernels of the homomorphisms of the inclusions: $N_*^p(K^n) = \ker i_*^{n,n+p}$.

Proposition 5 $N^{p+1}_*(K^n) \subset N^p_*(K^n).$

Next, we "remove" the noise from the homology group. The *p*-persistent (homology) group of K^n with respect to the filtration $\{K^n\}$ is defined as

$$H^p_*(K^n) = H_*(K^n)/N^p_*(K^n).$$

The point of this definition is that, given a threshold for noise, if the difference between two homology classes is noise, they should be equivalent.

Next, just as in the case of noise-less analysis, we define a single structure to capture all (robust) homology classes. Let p be a positive integer. Suppose $x \in \ker i_*^{k,k+p}$ and let $y = i_*^{k,k+1}(x)$. Then

$$i_*^{k+1,k+1+p}(y) = i_*^{k+1,k+1+p}(i_*^{k,k+1}(x))$$

= $i_*^{k,k+1+p}(x) = i_*^{k+p,k+p+1}(i_*^{k,k+p}(x))$
= $i_*^{k+p,k+p+1}(0) = 0.$

Hence $y \in \ker i_*^{k+1,k+1+p}$. We have proved that

$$i_*^{k,k+1}(\ker i_*^{k,k+p}) \subset \ker i_*^{k+1,k+1+p}.$$

It follows that the homomorphism $i_*^{k,k+1}$: ker $i_*^{k,k+p} \rightarrow$ ker $i_*^{k+1,k+1+p}$ generated by the inclusion is well-defined.

Next, we use these homomorphisms to define the *p*noise (homology) group $N^p_*(\{K^n\})$ of filtration $\{K^n\}$ as

$$N_*^p(\{K^n\}) = \ker i_*^{1,2} \oplus \ldots \oplus \ker i_*^{s,s+1}$$

Observe that the formula is the same as the one in the definition of $H^p_*(\{K^n\})$. Since $i^{k,k+1}_* : \ker i^{k,k+p}_* \to \ker i^{k+1,k+1+p}_*$ is a restriction of $i^{k,k+1}_* : H^p_*(K^k) \to H^p_*(K^{k+1})$, each term in the above definition is a subgroup of the corresponding term in the definition of $H_*(\{K^n\})$. The proposition below follows.

Proposition 6 $N_*^p(\{K^n\}) \subset H_*(\{K^n\}).$

Finally, the *p*-persistent (homology) group of filtration $\{K^n\}$ is

$$H^p_*(\{K^n\}) = H_*(\{K^n\}) / N^p_*(\{K^n\}).$$

The results about $H^p_*(\{K^n\})$ analogous to the ones about $H_*(\{K^n\})$ in Section 5 hold.

8 Computational aspects

For 2-dimensional gray scale images, this approach to homology and persistence has been used in an image analysis program. The algorithm described in [8] has complexity of $O(n^2)$, where *n* is the number of pixels in the image, in the worst case. As a result, the processing time for images of common sizes is several seconds on a typical PC.

For the general case, the analysis algorithm may be outlined as follows:

- 1. The input is a filtration.
- 2. The homology groups of its members and the homomorphisms induced by inclusions are computed.
- 3. The homology group of the filtration is computed.
- 4. The persistence of all elements of the homology groups is computed.
- 5. The user sets a threshold p for persistence and the p-noise group of the filtration is computed.
- 6. The *p*-persistent homology group of the filtration is computed and given as output.

If the user changes the threshold, the last two steps are repeated as necessary without repeating the rest.

The algorithm above computes the homology group of filtration, as defined, incrementally. This may be both a disadvantage and an advantage. In comparison, the *persistence complex* [3] also contains information about all homology classes of the filtration but its computation does not require computing the homology of each complex of the filtration. Meanwhile, the above algorithm may have to compute the same homology over and over if consecutive complexes are identical. Hence, the algorithm has a disadvantage in terms of processing time. On the other hand, the incremental nature of the algorithm makes its use of memory independent from the length of the filtration. Another advantage is that multi-parameter filtrations are dealt with in the exact same manner (see next section).

The inefficiency of the above algorithm can be addressed with a proper algebraic tool. This tool is the mapping cone [9]. Suppose, for simplicity, that our filtration has only two elements: $i: K^1 \hookrightarrow K^2$. The mapping cone is, in a sense, a combination of the kernel and the cokernel of i_* . It captures the difference between K^1 and K^2 on the chain level: everything in $C_*(K^1)$ is killed unless it also appears in $C_*(K^2)$ under i_* . Then the algorithm is to construct the homology group from the chain complexes $C_*(K^1), C_*(K^2)$ of the elements of the filtration and the chain map $i_*: C_*(K^1) \to C_*(K^2)$.

9 Multiparameter filtrations

Multiparameter filtrations come from the same main sources as one-parameter filtrations. First, color images are thresholded according to their three color channels. Second, point clouds are thresholded by the closeness of their points and, for example, the density of the points.

Let's limit our attention to the two-parameter case. A (finite) two-parameter filtration $\{K^{nm}\}$ is a table of complexes connected by inclusions

$$i(n,m,n+p,m+q):K^{nm}\rightarrow K^{n+p,m+q}, p,q\geq 0,$$

These inclusions generate homomorphisms

$$i_*(n, m, n+q, m+p) : H_*(K^{nm}) \to H_*(K^{n+q, m+p}),$$

with 0s added in the end of each row and each column. Define the homology group of the filtration $\{K^{nm}\}$ as

$$H_*(\lbrace K^{nm}\rbrace)$$

= $\bigoplus_{nm} \ker i_*(n, m, n+1, m) \cap \ker i_*(n, m, n, m+1).$

The analogues of the results in Section 5 hold.

There are many ways to define persistence in the multiparameter setting. For example, we can evaluate the robustness of a homology class $x \in H_*(K^{nm})$ in terms of the pairs (p,q) of positive integers satisfying

$$i_*(n, m, n+p, m)(x) = 0$$
 and $i_*(n, m, n, m+q)(x) = 0$.

Next, just as in Section 7, we restrict the homomorphisms generated by the inclusions to the homology classes of low persistence:

$$\begin{split} & i_*(n,m,n+1,m): \\ & \ker i_*(n,m,n+p,m) \to \ker i_*(n+1,m,n+1+p,m), \\ & i_*(n,m,n,m+1): \\ & \ker i_*(n,m,n,m+q) \to \ker i_*(n+1,m,n,m+1+q). \end{split}$$

Then the (p, q)-noise group of K^{nm} is defined via these homomorphisms:

$$N^{pq}_*(\lbrace K^{nm} \rbrace) = \bigoplus_{nm} \ker i_*(n, m, n+1, m) \cap \ker i_*(n, m, n, m+1).$$

Finally, the (p,q)-persistent (homology) group of filtration $\{K^{nm}\}$ is defined as

$$H^{pq}_*(\{K^n\}) = H_*(\{K^{nm}\})/N^{pq}_*(\{K^{nm}\}).$$

The results about $H^{pq}_*(\{K^{nm}\})$ analogous to the ones about $H^p_*(\{K^n\})$ in Section 7 hold.

10 Summary and further research

The main contributions of the present paper are the following.

- Homology group of filtration is defined to serve as a substitute for the homology group of the dataset that produced the filtration.
- This group is an algebraic treatment of persistence alternative to the persistence module. It is arguably easier to compute as it is simply the sum of kernels.
- The algorithm has been tested with real-life images and proven practical in terms of both output and processing time.
- For analysis of point clouds, the approach provides the output that is scale independent.
- For image analysis, the approach provides the output that is both scale independent and gray-level independent.
- Unlike the persistence module, our approach yields a natural generalization to multiparameter filtrations.

The following issues will be addressed in a forthcoming paper:

- the stability of the homology group of filtration;
- the functoriality properties of the homology group of filtration;
- the relation between the homology group of filtration and the persistence complex;
- the mapping cone construction for the homology group of filtration;
- the homology group of multiparameter and poset filtrations.

I thank the reviewers for their comments that helped improve this paper.

References

- G. Bredon, Topology and Geometry, Springer Verlag, 1993.
- [2] G. Carlsson, Topology and data, Bulletin of the Amer. Math. Soc., Vol. 46, No. 2, pp. 255-308, 2010.
- [3] G. Carlsson and A. Zomorodian, Computing persistent homology. Discrete and Computational Geometry, 2005, 20th ACM Symposium on Computational Geometry, Brooklyn, NY, 2004.

- [4] G. Carlsson and A. Zomorodian, The theory of multidimensional persistence. 23rd ACM Symposium on Computational Geometry, Gyeongju, South Korea, 2007. Discrete and Computational Geometry, 2009.
- [5] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, Stability of persistence diagrams, Discrete and Computational Geometry, vol. 37, no. 1, pp. 103-120 (2007).
- [6] H. Edelsbrunner, D. Letscher, and A. Zomorodian, Topological persistence and simplification. Discrete Comput. Geom. 28 (2002), pp. 511-533.
- [7] T. Kaczynski, K. Mischaikow, and M. Mrozek, Computational Homology, Appl. Math. Sci. Vol. 157, Springer Verlag, NY, 2004.
- [8] P. Saveliev, A graph, non-tree representation of the topology of a gray scale image, Image Processing, Algorithms and Systems IX, 2011, Volume 7870, O1-O19.
- [9] C. A. Weibel, An Introduction to Homological Algebra, Cambridge University Press, 1994.

Planar Pixelations and Shape Reconstruction

Brandon Rowekamp*

Abstract

Given a PL (piecewise linear) set S in the plane \mathbb{R}^2 we consider the set $P_{\varepsilon}(S)$ consisting of all pixels of size ε that touch S. This pixelation $P_{\varepsilon}(S)$ resembles the original set, but may not well approximate various important invariants of the original set such as Betti numbers, perimeter, curvature measures. We describe an algorithm that associates to the pixelation $P_{\varepsilon}(S)$ a PL-set $\mathcal{P}_{\varepsilon}(S)$ which approximates S in a very strong sense.

1 Introduction

The ε -pixelation of the Euclidean plane is the decomposition determined by the lines

$$x \in \varepsilon \mathbb{Z}$$
 and $y \in \varepsilon \mathbb{Z}$.

An ε -pixel is a square of the form

$$[\varepsilon(i-1),\varepsilon i] \times [\varepsilon(j-1),\varepsilon j], \ i,j \in \mathbb{Z}$$

with center located at

$$C[i,j] := \left(\frac{2i-1}{2}\varepsilon, \frac{2j-1}{2}\varepsilon\right)$$
(1.1)

For any compact subset of the plane we define its ε pixelation to be the union of all the ε -pixels that touch S. We denote it by $P_{\varepsilon}(S)$.



Figure 1: A pixelation of an angle

Roughly speaking, the main goal of this paper is to algorithmically associate to $P_{\varepsilon}(S)$ a planar *PL*-region $\mathcal{P}_{\varepsilon}(S)$ that approximates *S* very well as $\varepsilon \searrow 0$. More specifically, we would like to recover in the limit basic geometric and topological invariants of *S* such as, area, perimeter, curvature (measures) and Betti numbers.

While $P_{\varepsilon}(S)$ converges to S in the Hausdorff distance, this notion of convergence fails to recover even the most stable of invariants. For example lengths from the pixelation may not converge to the corresponding lengths in the original set. Additionally, topological information such as the Betti numbers may be lost at all resolutions ε .

For example, in Figure 1 we have depicted a pixelation of the angle A formed by two segments of slopes $\frac{2}{3}$ and 1 that have a common endpoint at the origin. The homotopy type of this pixelation is independent of the size of the pixel, and as seen in Figure 1, $b_1(P_{\varepsilon}(A)) = 2$, $\forall \varepsilon > 0$. The situation with geometric invariants such as perimeter or curvature is much worse (even in the case of a line the total curvature will explode while the perimeter will not converge to the correct length). Therefore the pixelation itself is not a reasonable approximation of the original shape. The goal of this paper is to generate algorithmically a better approximation using only information from the pixelation.

2 Basic Results

Proofs for all claims in this section are omitted due to lack of space. They are publicly available on the author's website¹.

Given the ε -pixelations of a *PL* subset of the plane, we would like to construct a sequence of *PL* approximations of the original set which converge in a strong sense to the original set. We delay a precise definition of the notion "strong convergence" until the main result. For now it suffices to say that we seek an approximation which recovers the homotopy type of the original set, as well as geometric invariants such as the perimeter, area and the curvature measures of the boundary.

As we have seen in the introduction, the pixelation itself will not recover these invariants. A better approximation is described explicitly in the Algorithm section given that the original set is PL^2 . The remainder of

^{*}Department of Mathematics, University of Notre Dame, browekam@nd.edu

¹http://www.nd.edu/~browekam

²We are currently working to extend this technique to semi-

this section describes results about pixelations which motivated the approximation which is described later.

Throughout this discussion we will use the concepts of *column* and *stack*. The column of a pixelation at the x-value x_0 is simply the union of pixels from $P_{\varepsilon}(S)$ which intersect the line $\{x = x_0\}$. A stack is a connected component of a column (that is to say, a series of pixels "stacked" on top of each other with no gaps).

An *elementary set* is a region of the following form:

$$\{(x,y) \in \mathbb{R}^2 : x \in [a,b], \beta(x) \le y \le \tau(x)\}.$$

where β and τ are continuous piecewise C^2 functions defined on [a, b] with the property that $\beta(x) \leq \tau(x)$ for all $x \in [a, b]$.

Proposition 1 If S is an elementary set, then it is contractible and $P_{\varepsilon}(S)$ is also contractible for every resolution ε .

Therefore, the pixelation of an elementary set has the same homotopy type as the elementary set itself. We will approximate elementary sets by connecting points along the tops and bottoms of the stacks that make up their pixelations. To ensure convergence of perimeter and total curvature we will connect points from about every σ -th column, where σ is a number determined by ε (this number is explicitly shown in the algorithm). If σ satisfies certain constraints (see (3.1)), then this method of approximation will recover the desired invariants.



Figure 2: This approximation of an elementary set is the region outlined by the black line.

As suggested by Figure 1, for non elementary sets there is no guarantee of convergence in homotopy type. Therefore for the approximation of a general piecewise linear set we must determine which cycles in the pixelation come from the original shape, and which cycles are artifacts of the pixelation process. Intuitively these

algebraic subsets of the plane.

fake cycles must be located very close to columns that undergo a change in the number of their stacks (since fake cycles will have small area). We define the function $n_{\varepsilon} : \mathbb{R} \setminus \varepsilon \mathbb{Z} \to \mathbb{Z}_{\geq 0}$, where

 $\boldsymbol{n}_{\varepsilon}(x_0) = \#$ of stacks of $P_{\varepsilon}(S)$ in the column at x_0 .

A point $x_0 \in \mathbb{R} \setminus \varepsilon \mathbb{Z}$ is called a *jumping point* of $\boldsymbol{n}_{\varepsilon}$ if

$$\boldsymbol{n}_{\varepsilon}(x_0+\varepsilon)\neq\boldsymbol{n}_{\varepsilon}(x_0)$$

A column over a jumping point is called a *jumping* column. From investigating examples of pixelations, we expect topological noise to occur "near" jumping points. The following theorem gives a precise sense of how "near" topological noise must be to jumping points.

Proposition 2 If S is a generic piecewise linear set, *i.e.*, no two of its vertices lie on the same vertical line. Then the following hold.

- 1. There is an integer $k = k_S$ depending only on S such that any fake cycles of $P_{\varepsilon}(S)$ is within at most k-columns from a jumping column.
- 2. The function $S \ni (x, y) \stackrel{h}{\mapsto} x \in \mathbb{R}$ is a stratified Morse function in the sense of [7] and for any jumping point x_0 of \mathbf{n}_{ε} there exists a critical value x_0^h of h such that $|x_0 - x_0^h| < k\varepsilon$.

In practical terms this proposition states that the fake cycles only occur in narrow vertical strips of the plane containing the jumping points of n_{ε} . Moreover the jumping points of n_{ε} cannot be too far from the critical values of the function h. Thus we can assume that any cycle which occurs close to a jumping column is fake and so our approximation should fill it in (the meaning of "close" here will be explicitly shown in the algorithm). We can fill these fake cycles somewhat carelessly, since they take up a very small area of the plane for small resolutions. We call the columns which can contain fake cycles "noise" and cover each connected component of $P_{\varepsilon}(S)$ within in these columns by rectangles.

So far we have a way to approximate two situations. The first is for elementary regions, and the second is for noise columns that accumulate near the critical values of h. Away from the critical values of h the set S is a disjoint union of elementary sets. Therefore these two approximation techniques suffice to approximate the entire set.

The next section gives an explicit algorithm for generating an approximation of a PL set from its pixelations.

3 The algorithm

The input for this algorithm is a pixelation. We encode a pixelation by an $m \times m$ matrix A with 0, 1 entries,



Figure 3: An example of approximating in noise columns (near jumping points of n_{ε}).

where $a_{ij} = 1$ if and only if the pixel with center C[i, j] belongs to our pixelation. We think of the parameter m as defining a $m \times m$ subdivision of a computer screen, consisting of squares of size $\varepsilon := \frac{1}{m}$. Define

$$\sigma(\varepsilon) = \lfloor m^{\rho} \rfloor,$$

where ρ is a fixed rational number $\rho \in (\frac{1}{2}, 1)$. Note that

$$\lim_{\varepsilon \searrow 0} \varepsilon(\sigma(\varepsilon))^2 = \infty, \quad \lim_{\varepsilon \searrow 0} \varepsilon\sigma(\varepsilon) = 0. \tag{3.1}$$

We denote by P(A) the pixelation determined by the matrix A. The output of the algorithm will be a PL set $\mathcal{P}_{\varepsilon}(A)$ that decomposes in a canonical fashion as a finite union of trapezoids with vertical bases. We will refer to such regions as *polytrapezoids*. We allow for degenerate trapezoids, such as points, segments, or triangles.

The algorithm uses several basic subroutines. The first one is the the subroutine **stack**. Its input is a list

$$C = C_1, \ldots, C_m, \quad C_i = 0, 1,$$

which encodes a column of the pixelation P(A). The output of stack is a list of nonnegative integers

$$n(C); \ b_1 \le t_1 < b_2 \le t_2 < \dots < b_{n(C)} \le t_{n(C)},$$

where n(C) is the number of stacks in the column encoded by C, and the location of the bottom and top pixel in the *j*-th stack is determined by the integers b_j, t_j . More formally

$$C_k = 1 \iff \exists 1 \le j \le \boldsymbol{n}(C) : b_j \le k \le t_j.$$

If $C = C_i$, the *i*-th column of P(A), i.e.,

$$C_i = a_{i,1}, \ldots, a_{i,m}$$

then we will denote the output $stack(C_i)$ by

$$n_i, b_{i,1} \leq t_{i,1} < \cdots < b_{i,n_i} \leq t_{i,n_i}.$$

A number $1 \leq i \leq m-1$ is called a *jump point* if

$$\boldsymbol{n}_i \neq \boldsymbol{n}_{i+1}.$$

The next subroutine that we need is called jump. Its input is an integer $k \in [1, m)$ and the output is an integer $j_k = \mathsf{jump}(k)$ defined as follows. If

$$\{i \in [k,m) \cap \mathbb{Z}; i \text{ is a jump point }\} = \emptyset,$$

then we set

$$\mathsf{jump}(k) := m+1$$

Otherwise

$$\mathsf{jump}(k) = \min\{i \in [k, m) \cap \mathbb{Z}; i \text{ is a jump point }\}.$$

The noise region is determined by a finite collection of intervals

$$[\ell_1, r_1], \dots, [\ell_\alpha, r_\alpha] \subset [1, m]$$

where the integers ℓ_k, r_k are determined inductively as follows.

$$\ell_1 = \max(\mathsf{jump}(1) - 2\sigma(\varepsilon), 1),$$

$$r_1 = \min(m, \mathsf{jump}(1) + 2\sigma(\varepsilon)).$$

Suppose that $\ell_1, r_1, \ldots, \ell_j, r_j$ are determined. If $\mathsf{jump}(r_j) > m$ we stop. Otherwise we set

$$\ell_{j+1} = \max(\mathsf{jump}(r_j) - 2\sigma(\varepsilon), 1),$$

$$r_{j+1} = \min(m, \mathsf{jump}(r_j) + 2\sigma(\varepsilon)).$$

The intervals $[\ell_1, r_1], \ldots, [\ell_\alpha, r_\alpha]$ may not be disjoint, but their union is a *disjoint* union of intervals

$$[a_1, b_1], \ldots, [a_J, b_J], b_i < a_{i+1}.$$

The intervals $[a_j, b_j], 1 \leq j \leq J$ are the noise intervals. The intervals

$$[1, a_1], [b_1, a_2], \dots, [b_{J-1}, a_J], [b_J, m]$$

are the *regular intervals*.

The heart of the algorithm consists of two procedures, one for dealing with the noise intervals and the other for dealing with the regular intervals. These procedures will return a number of polytrapezoids,

First some notation. Given a collection of points

$$B_0, T_0, \ldots, B_N, T_N \in \mathbb{R}^2$$

such that

$$\begin{aligned} x(B_i) &= x(T_i), \ y(B_i) \le y(T_i), \ \forall i = 0, \dots, N, \\ x(B_{j-1}) < x(B_j), \ \forall 1 \le j \le N, \end{aligned}$$

we denote by $polygon(B_0, T_0, \ldots, B_N, T_N)$ the region surrounded by the simple closed *PL*-curve obtained as the union of line segments

$$[B_0, B_1], \ldots, [B_{N-1}, B_N],$$

 $[B_N, T_N], \ldots, [T_1, T_0], [T_0, B_0].$

Note that each of the quadrilaterals $B_{i-1}, B_i, T_i, T_{i-1}$ is a (possibly degenerate) trapezoid with vertical bases.

Consider first the regular intervals. Given a regular interval I := [p, q] we observe that the number of stacks n_i is independent of $i \in [p, q]$. We denote this shared number by n = n(I).

We construct inductively a sequence of numbers $i_0 < \cdots < i_N$ as follows:

- We set $i_0 = p$.
- If $q p < 2\sigma(\varepsilon)$ we set N = 1 and $i_1 = q$.
- If i_0, \ldots, i_k are already constructed, then, if $q i_k < 2\sigma(\varepsilon)$ we set N = k + 1 and $i_{k+1} = q$, else $i_{k+1} = i_k + \sigma(\varepsilon)$.

Note that if $q - p > \sigma(\varepsilon)$, then $N \ge 1$, $i_0 = p$, $i_N = q$ and

$$N = 1$$
 if $q - p < \sigma(\varepsilon)$.

We have

$$stack(C_{i_k}) = n, \ b_{i_k,1}, t_{i_k,1}, \dots, b_{i_k,n}, t_{i_k,n}.$$

For $j = 1, ..., \mathbf{n}$, and k = 0, ..., N we denote by $B_{k,j}$ the center of the ε -pixel corresponding to the element entry $b_{i_k,j}$ in the column C_{i_k} . Similarly we denote by $T_{k,j}$ the center of the pixel corresponding to the entry $t_{i_k,j}$ of the column C_{i_k} . For $1 \le j \le \mathbf{n}(I)$, we set

$$\mathcal{P}_j(I) := \mathsf{polygon}(B_{0,j}, T_{0,j}, \dots, B_{N,j}, T_{N,j}).$$

Define

$$\mathcal{P}(I) = \bigcup_{j=1}^{n(I)} P_j(I), \quad \mathcal{P}_{\text{reg}} := \bigcup_{I \text{ regular interval}} P(I).$$

Suppose now that I = [p, q] is a noise interval. We modify the column

$$C_p = a_{p,1}, \ldots, a_{p,m}$$

to a column

$$C'_p = a'_{p,1}, \dots, a'_{p,m}$$

by setting

$$a'_{p,k} := \begin{cases} 1, & \text{if } \sum_{i=p}^{q} a_{i,k} > 0 \\ \\ 0, & \text{if } \sum_{i=p}^{q} a_{i,k} > 0. \end{cases}$$

We apply the subroutine stack to the new column C'_p and the output is

$$\mathsf{stack}(C'_p) = n(I), \ b_1 \leq t_1 < \dots < b_n \leq t_n.$$

For $j = 1, \ldots, \boldsymbol{n}(I)$ we set

$$B_{0,j} := C[p, b_j], \ T_{0,j} := C[p, t_j],$$

$$B_{1,j} := C[q, b_j], \ T_{0,j} := C[q, t_j],$$

where C[i, j] is defined by (1.1). Next, for $j = 1, \ldots, n(I)$ we define the rectangle

$$\mathcal{R}_j(I) := \mathsf{polygon}(B_{0,j}, T_{0,j}, B_{1,j}, T_{1,j}),$$

and we set

$$\mathfrak{R}(I) = \bigcup_{j=1}^{n(I)} \mathfrak{R}_j(I), \quad \mathfrak{P}_{\text{noise}} := \bigcup_{I \text{ noise interval}} R(I).$$

The output of the algorithm is the polytrapezoid

$$\mathcal{P}_{\varepsilon}(A) := \mathcal{P}_{\text{regular}} \cup \mathcal{P}_{\text{noise}}.$$

4 The main result

We wish to prove that the above algorithm produces an approximation of the original set which preserves the Euler characteristic, perimeter of the boundary, total curvature of the boundary and other important invariants. One way to precisely state this is to use the concept of *normal cycle* of a (subanalytic) set. The definition of a normal cycle uses basic concepts of geometric measure theory (for an introduction to the subject see [4, 8]).

Recall that a 1-current on a smooth manifold is an element of the dual space of compactly supported differential 1-forms. Thus, a current is an object T which associates a number $T(\omega)$ to a compactly supported 1-form ω . The number $T(\omega)$ can be thought of as the integral of ω over the current ω . For example an oriented smooth arc is a 1-current which acts through integration. The mass of a current T is the quantity

$$\sup\{T(\omega): \omega \text{ is a 1-form}, \sup_{x} ||\omega_{x}|| \leq 1\},\$$

where $\|-\|$ denotes the Euclidean norm on the dual of \mathbb{R}^2 . The mass of a 1-current defined by an oriented compact arc is equal to its length.

The normal cycle of of a planar PL set S is 1-current N^S living on the unit tangent sphere bundle of \mathbb{R}^2 . It consists of a finite collection of compact, oriented real analytic arcs with multiplicities such that the resulting singular chain is a cycle. Each of these arcs is a Legendrian curve with respect to the canonical contact structure on the unit tangent sphere bundle. This cycle is uniquely characterized by the Morse theoretic properties of the restrictions to S of the linear functions on \mathbb{R}^2 . Roughly speaking, the normal cycle is obtained as follows.

Denote by $T_{\varepsilon}(S)$ the tube of radius ε around S,

$$T_{\varepsilon}(S) := \left\{ p \in \mathbb{R}^2; \text{ dist}(p, S) \le \varepsilon \right\}.$$

This is a domain in the plane and its normal cycle is the current $N^{T_{\varepsilon}(S)}$ defined the graph of the Gauss map of

the boundary. If we canonically identify the unit sphere bundle with the Cartesian product $S^1 \times \mathbb{R}^2$, then the graph of the Gauss map can be identified with the collection of points $(\nu(p), p) \in S^1 \times \mathbb{R}^2$, where $p \in \partial T_{\varepsilon}(S)$ and $\nu(p)$ is the outer unit normal vector to $\partial T_{\varepsilon}(S)$ at p. Then

$$N^S = \lim_{\varepsilon \searrow 0} N^{T_\varepsilon(S)}$$

For a precise definition of the normal cycle we refer to [1, 2, 6, 9, 10]. In particular, [9] gives a beautiful description of the normal cycle, together with specific examples of normal cycles and how to retrieve geometric information from them.

In this setting, we can show the following theorem:

Theorem 3 Suppose S is a generic PL subset of \mathbb{R}^2 , *i.e.*, no two vertices lie on the same vertical line. Fix a function $\sigma : \mathbb{R}^+ \to \mathbb{Z}^+$ satisfying (3.1).

Let $\mathcal{P}_{\varepsilon}(S)$ be the PL approximation of S constructed via the Algorithm described above. Then the normal cycle $N^{\mathcal{P}_{\varepsilon}(S)}$ of $\mathcal{P}_{\varepsilon}(S)$ converges weakly to the normal cycle N^{S} of S as $\varepsilon \to 0$.

Proof: Due to lack of space we will omit most of the details. The complete proof is publicly available on the author's website³. We confine ourselves to outlining the most salient features of the proof.

The theorem is largely a consequence of the approximation theorem for normal cycles proved by Joseph Fu in [5]. This theorem implies the following result.

Proposition 4 Suppose S is a PL subset of the plane and for each $\varepsilon \in (0,1)$ $L_{\varepsilon}(S)$ is a PL set such that the following hold

- 1. There is a compact subset K of the plane such that $L_{\varepsilon}(S) \subset K$ for all $\varepsilon \in (0, 1)$.
- 2. There is a M > 0 such that

$$\max\left(N^{L_{\varepsilon}(S)}\right) < M, \quad \forall \varepsilon \in (0,1).$$

3. For almost every half plane H,

$$\lim_{\varepsilon \searrow 0} \chi(H \cap L_{\varepsilon}(S)) = \chi(H \cap S),$$

where χ denotes the Euler characteristic.

Then the normal cycles $N^{L_{\varepsilon}(S)}$ converge weakly to the normal cycle N^{S} .

The bulk of the proof consists of verifying all the conditions in the above theorem for $\mathcal{P}_{\varepsilon}(S)$. Note that the first condition follows immediately from the construction of $\mathcal{P}_{\varepsilon}(S)$.

The second condition is a bit more difficult. If X is a PL set, then the mass of its normal cycle can be

expressed in terms of its perimeter and its total curvature; see [9]. The conditions (3.1) allow us to produce upper bounds on the perimeter and the total curvature of $\mathcal{P}_{\varepsilon}(S)$ that are independent of ε . Condition 2 follows immediately from these upper bounds.

The third condition is the most challenging. To verify it we associate to each approximation $\mathcal{P}_{\varepsilon}(S)$ a graph Γ_{ε} , and a key observation is the fact that for ε sufficiently small the graph Γ_{ε} is isomorphic to the Reeb graph, [3, VI.4], of the map $h: S \to \mathbb{R}$ defined by the projection onto the *x*-axis. As a matter of fact, our entire algorithm can be viewed as a discretization of the Morse theory of the above map. \Box

5 Conclusion

Weak convergence in normal cycles implies the convergence of Euler characteristic, perimeter, curvature. This implies that the approximation $\mathcal{P}_{\varepsilon}(S)$ created by our algorithm converges in a very strong way to the original set, recovering information that the pixelation destroys.

A simple example of the approximation algorithm applied to the case of two intersecting line segments is shown in the Figure 4. Note the rectangle around the point of intersection indicating a noise region.



Figure 4: An example of $L_{\varepsilon}(S)$ where S is the union of two intersecting lines.

We note that the current approximation technique only applies to piecewise linear sets. This is because the construction relied on Proposition 2, which is only true for PL sets. In more general cases, such as semialgebraic sets, fake cycles can stray further away from the singular points of S. (Think of the pixelation of a cusp.) Therefore more of the approximation will need

³http://www.nd.edu/~browekam

to be devoted to noise intervals in this case. We can show⁴ that with a cleverer choice of σ , the general approximation method still works for semi-algebraic sets.

References

- A. Bernig: The normal cycle of compact definable sets, Israel J. Math., 159(2007), 373-411.
- [2] J. Cheeger, W. Müller, R. Schrader: Kinematic and tube formulas for piecewise linear spaces, Indian Univ. Math. J., 35(1986), 737-754.
- [3] H. Edelsbrunner, J. Harer: Computational Topology. An Introduction, Amer. Math. Soc., 2010.
- [4] H. Federer: Geometric Measure Theory, Springer Verlag, 1969.
- [5] J. Fu: Convergence of curvatures in secant approximations, J. Diff. Geom. 37(1993), 177-190.
- [6] J. Fu: Curvature measures of subanalytic sets, Am. J. Math. 116(1994), 819-890.
- [7] M. Gorseky, R. MacPherson: Stratified Morse Theory, Springer Verlag, 1988.
- [8] F. Morgan: Geometric Measure Theory: A Beginner's Guide, Elsevier, 2009.
- [9] J.M. Morvan: Generalized Curvatures, Springer Verlag, 2008.
- [10] L.I. Nicolaescu: On the normal cycles of subanalytic sets, Ann. Glob. Anal. Geom., 39(2011), 427-454.

⁴This is work in progress.

Counting Simple Polygonizations of Planar Point Sets

Emo Welzl *

Given a finite planar point set, we consider all possible spanning cycles whose straight line realizations are crossingfree – such cycles are also called *simple polygonizations* – and we are interested in the number of such simple polygonizations a set of N points can have. While the *minumum* number over all point configurations is easy to obtain – this is 1 for points in convex position –, the maximum seems to be more involved. M. Newborn and W.O.J. Moser were the first to ask the question around 1980 and they gave first evidence that this number has to be significantly less than the overall number (N - 1)!/2 of all spanning cycles. In 2000 A. Garcia, M. Noy and J. Tejel describe points sets that have as many as $\Omega(4.65^N)$ simple polygonizations, no improvement on this end has been reported since then. Despite of several improvements on the upper bound over the years, the currently best upper bound of $O(54.6^N)$ (recent joint work with A. Sheffer and M. Sharir) leaves obviously a big gap to be closed.

We report on the history of the problem and show how it connects to counting triangulations and crossingfree perfect matchings, and how Kasteleyn's algebraic method for counting perfect matchings in planar graphs enters the picture. Basicially nothing is known for related algorithmic questions (determining the number of simple polygonizations for a given point set, enumerating all simple polygonizations).

^{*}ETH Zürich, Switzerland

Algorithms for Bivariate Majority Depth

Dan Chen *

Pat Morin^{*}

Abstract

The majority depth of a point with respect to a point set is the number of major sides it is in. An algorithm for majority depth in \mathbb{R}^2 is given in this paper, and it is the first algorithm to compute the majority depth. This algorithm runs in $O((n+m)\log n)$ time with Brodal and Jacob's data structure, and in $O\left((n+m)\frac{\log n}{\log\log n}\right)$ time in the word RAM model.

1 Introduction

A data depth is a measure of the centrality of a point with respect to a given data cloud in \mathbb{R}^d . Many depth notions have been introduced, such as *Tukey depth* [21], Oja depth [17], Simplicial depth [15], and majority depth [18, 16]. For the introduction of these notions, one can refer to the surveys by Small [19] and Aloupis [2]. In this paper we give an algorithm for the majority depth. Let S be a set of points in \mathbb{R}^d . If the points in S are in general position (no d+1 points of S lie on a common hyperplane), any d points in S define a unique hyperplane \hbar . With \hbar as the common boundary, two closed half-spaces are obtained. The one containing more than or equal to $\frac{n+d}{2}$ points is called the major side of \hbar . Note that halving hyperplanes have two major sides. Given a finite set S of n points and a point p in \mathbb{R}^d , the majority depth of p is the number of major sides it is in.

In this paper we consider the problem of computing the majority depth of a point p with respect to a set Sof n points in \mathbb{R}^2 . We assume that the points in S are in general position. The tools for the algorithm are given in Section 2 and Section 3, and the algorithm is given in Section 4.

2 Dual Arrangement

Let H be a set of n hyperplanes in \mathbb{R}^d . We say that H is in general position, if every subset of d hyperplanes intersect in one point, and no d + 1 hyperplanes intersect in one point. We say a hyperplane is *vertical* if it contains a line parallel to the x_d -axis. Without loss of generality, we assume that no hyperplane in H is vertical. The arrangement $\mathcal{A}(H)$ of H is the partitioning

of \mathbb{R}^d induced by H into vertices (intersections of any d hyperplanes in H), faces (each flat in $\mathcal{A}(H)$ is divided into pieces by the hyperplanes in H that do not contain the flat, a *j*-face is a piece in a *j*-flat), and regions (connected components in \mathbb{R}^d separated by hyperplanes in H). We call $\mathcal{A}(H)$ a simple arrangement if H is in general position.

In an arrangement, we say a point p is at the klevel [1, 11, 14], if there are k hyperplanes in H lying vertically below p. (Above and below are with respect to the x_d coordinate.) The k-level of $\mathcal{A}(H)$ is the clo-



Figure 1: The 1-level of an arrangement in \mathbb{R}^2

sure of all the points of H at level k. Let m be the number of vertices of the k-level. Tight bounds for m are still open problems. In \mathbb{R}^2 the best known upper bound of m is $O(nk^{1/3})$ [9], and the best known lower bound for m is $n2^{\Omega(\sqrt{\log k})}$ [20]. In \mathbb{R}^2 , constructing the k-level takes $O((n+m)\log n)$ time using Edelsbrunner and Welzl's algorithm [13] with the data structure in [3], and it takes $O(n\log n + nk^{1/3})$ expected time with Chan's randomized algorithm [4] which is output insensitive. In the word RAM model, the construction takes $O\left((n+m)\frac{\log n}{\log \log n}\right)$ time [8].

Let $\mathcal{A}(T)$ be the dual arrangement [1, 11, 14] of S, where T is a set of dual hyperplanes of the points in S. For a hyperplane \hbar determined by d points in S, the major side of \hbar contains at least $\lceil \frac{n-d}{2} \rceil$ points in its interior, and they are either above or below \hbar . Let \hbar^* be the dual image of \hbar in $\mathcal{A}(T)$. Then, below or above \hbar^* there are the same number of hyperplanes. We define the major side of \hbar^* as a direction of the x_d -axis along which the ray from \hbar^* intersects at least $\lceil \frac{n-d}{2} \rceil$ hyperplanes. The directions of the major sides of \hbar and \hbar^* are opposite since the relative position between a point and a hyperplane is reversed in the dual space. However, if a point is in the major side of \hbar , the dual

^{*}School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada, dchen4@connect.carleton.ca, morin@scs.carleton.ca.

image of the point (a hyperplane) is on the major side of $\hbar^*.$

In the dual arrangement, we call vertices red if they have major side facing down, blue if the have major side facing both up and down. Then the majority depth of p is equal to the number of purple vertices plus the number of red vertices above p^* plus the number of blue vertices below p^* .

When n is odd, the vertices with level less than $\lceil \frac{n-d}{2} \rceil$ in $\mathcal{A}(T)$ are blue, and the ones with level more than that



Figure 2: The vertices and major sides when n is odd

are red (see Figure 2). For each vertex on the $\lceil \frac{n-d}{2} \rceil$ -level, if the convex angle of its two adjacent segments faces up it is blue, and if it faces down it is red.

When *n* is even, the situation is a little different. As shown in Figure 3, the vertices with level less than $\lceil \frac{n-d}{2} + 1 \rceil$ are blue, and the ones with level more than $\lceil \frac{n-d}{2} \rceil$ are red. The ones on both of these two levels are purple.

Computing the majority depth of p with respect to S is to count the number of major sides p is in. Since the total number of vertices in a simple arrangement is $\binom{n}{d}$, to compute the majority depth it is sufficient to count the number of vertices in $\mathcal{A}(T)$ whose major side does not contain p^* . This problem involves counting the number of vertices of $\mathcal{A}(T)$ that are contained in a set of polygons whose boundary is determined by p^* and the median level of $\mathcal{A}(T)$. We study this problem in the next section.

3 Counting Vertices

In this section we discuss how to count the vertices of a 2-dimensional arrangement of n line segments confined by a simple polygon (see Figure 4). Since there can be $\Omega(n^2)$ intersections in this arrangement, a sweep line algorithm would take too much time. In the following we discuss a couple of more efficient ways of counting the vertices.

We first transform the arrangement into a structure as shown in Figure 5, which makes the pattern of in-



Figure 4: An arrangement in a simple polygon

tersections clearer to us. In this structure, the polygon is cut at some point and laid flat, and all the line segments are bent into arcs, so that no two arcs intersect twice. The number of intersections in the new structure is the same as that in the original one, because, for any two line segments intersecting in the polygon, the corresponding arcs intersect once.

Notice that for an arc a, any other arc that intersects a has an endpoint laying between the two ends of a. To count the intersections in the new structure, we can use a queue. Starting from one end of the new structure, we add the endpoints of the arcs to the queue. Once the other end of an arc is in the queue, we count the number of endpoints between the two endpoints of the arc, which is the number of intersections the arc contributes. We then remove the two ends from the queue. Upon reaching the other end of the structure, the queue will be empty and all the intersections will be counted. If we implement the queue with an augmented binary tree [7, Chapter 14.1], finding the distance between the two ends of an arc and deleting the other end of the arc takes $O(\log n)$ time, so the number of intersections can be counted in $O(n \log n)$ time.

Another way to count the intersections is to use an array A of size 2n. Starting from one end of the structure, we walk to the other end. Once we come across a starting end of an arc, we append a 1 to A. Once we come across a finishing end of an arc a, we append a 0 to A. Let the index of the starting end of a in A be i, and that of the finishing end be j. We then set A[i] to 0. Let sum(k) denote $\sum_{l \leq k} A[l]$. The number of the intersections that a contributes is the number of endpoints we came across between A[i] and A[j], which is sum(j) - sum(i). We can compute sum(k) in $O\left(\frac{\log n}{\log \log n}\right)$ time with Dietz's algorithm [10] in the word RAM model. Then counting all intersections takes $O\left(n \frac{\log n}{\log \log n}\right)$ time.

4 The Algorithm

In this section we show how to use the intersection counting structure of the previous section to obtain an efficient algorithm for the majority depth problem in





Figure 5: The transformed arrangement

 \mathbb{R}^2 . In the following we will first describe the algorithm when n is odd, then we describe the modifications needed when n is even.

If n is odd, we first compute the median level of the dual arrangement of S and the intersections between it and p^* . If they intersect, p^* splits the levels into sections (as the schematic example shown in Figure 6). Each section along with p^* form a simple polygon except the leftmost and rightmost sections, which form unbounded regions. In order to count the vertices in the unbounded region with the methods in Section 3, we need to find the leftmost and rightmost vertices. Since the extreme points of the set of vertices of $\mathcal{A}(T)$ can be found in $O(n \log n)$ time by sorting the lines by slope [6], we can find those two vertices in $O(n \log n)$ time. Then we can add a vertical line to the left of the leftmost vertex, and one to the right of the rightmost vertex to bound the unbounded region (An example is shown in Figure 7). Now we can count the vertices in each polygon, and the ones on the median level whose major side does not contain p^* .

If n is even we need to compute both the $\frac{n}{2}$ -level and $(\frac{n}{2}-1)$ -level. The polygons should be formed by part of p^* and the one of the two median levels which is further away from p^* (see the schematic example in Figure 8). In Figure 9 is an example where all regions are bounded. Then we need to count all the vertices in the polygons, and count the vertices that on both those levels since they should be counted twice for the depth of p.



Figure 7: The polygons when n is odd

The number of lines that intersect with p^* is n, and the number of lines that intersect with the two vertical lines is no more than 2n. Since, in the polygons, each line segment that intersects with the median level has a unique extension on the median level, the total number of line segments that intersect with the median level is no more than m. Each line segment in the polygons has two ends on the boundaries, therefore, the total number of line segments in all the polygons is no more than 3n + m.

We obtain two different algorithms for computing majority depth depending on which algorithm we use for computing the median level and counting the vertices in a polygon.



Figure 6: The regions when n is odd



Figure 8: The regions when n is even

Theorem 1 The majority depth in \mathbb{R}^2 can be computed in

- 1. $O((n+m)\log n)$ time with Brodal and Jacob's data structure.
- 2. $O\left((n+m)\frac{\log n}{\log\log n}\right)$ time in the word RAM model.

The complexity of these algorithms is determined by the value of m, which is the number of vertices of the median level.

5 Conclusion

We have given an algorithm for computing the majority depth of a point p with respect to a set S of n points in \mathbb{R}^2 . The algorithm's running time is dependent on the size of the median level of the dual arrangement of S. Even without leaving 2 dimensions, this work leaves several open questions:

- 1. (Depth of a point) Is there an $O(n \log^{O(1)} n)$ time algorithm for computing the majority depth of a point p with respect to a set S of n points in \mathbb{R}^2 ?
- 2. (Deepest point) Given a set S of n points in \mathbb{R}^2 , how quickly can we compute a point p whose majority depth (with respect to S) is maximum?
- 3. (Centerpoint) Determine the maximum value k = f(n) for which the following statement is true: For

any set S of n points in \mathbb{R}^2 , there exists a point $p \in \mathbb{R}^2$ whose majority depth, with respect to S, is at least $\binom{n}{2}/2 + k$.

4. (Faster algorithm in the word RAM model) The related problem of counting inversions has recently been solved in $O(n\sqrt{\log n})$ running time [5]. This unfortunately does not improve our algorithm. Can the factor $\frac{\log n}{\log \log n}$ in the running time of our algorithm be replaced by $\sqrt{\log n}$?

An algorithm for the first problem would have to avoid computing the median level. The second problem is easily solved in $O(n^4)$ time and $O(n^2)$ space by traversing the arrangement of lines through all $\binom{n}{2}$ pairs of points in S using the topological sweep algorithm [12].

References

- P. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geome*try, pages 49–119. Elsevier Science Publishers North-Holland, 1998.
- [2] G. Aloupis. Geometric measures of data depth. In DI-MACS Series in Discrete Mathematics and Theoretical Computer Science, 2006.
- [3] G. Brodal and R. Jacob. Dynamic planar convex hull. In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, pages 617–626, 2002.
- [4] T. Chan. Remarks on k-level algorithms in the plane. Manuscript, 1999.



Figure 9: The polygons when n is even

- [5] T. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In Proceedings of the 21st ACM/SIAM Symposium on Discrete Algorithms (SODA), pages 161–173, 2010.
- [6] Y. Ching and D. Lee. Finding the diameter of a set of lines. *Pattern Recognition*, 18(3-4):249–255, 1985.
- [7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, Cambridge, MA USA, 2nd edition, 2001.
- [8] E. Demaine and M. Pătraşcu. Tight bounds for dynamic convex hull queries (again). In Proceedings of the 23rd annual ACM symposium on Computational geometry, SoCG '07, pages 354–363, New York, NY, USA, 2007. ACM.
- [9] T. Dey. Improved bounds for planar k-sets and related problems. Discrete & Computational Geometry, 19(3):373-382, 1998.
- [10] P. Dietz. Optimal algorithms for list indexing and subset rank. In F. Dehne, J. Sack, and N. Santoro, editors, *Algorithms and Data Structures*, volume 382 of *Lecture Notes in Computer Science*, pages 39–46. Springer Berlin, 1989.
- [11] H. Edelsbrunner. Algorithms in Combinatorial Geometry. Springer-Verlag, Heidelberg, Germany, 1987.
- [12] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38(1):165–194, 1989.
- [13] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. SIAM Journal on Computing, 15(1):271–284, 1986.
- [14] D. Halperin. Handbook of discrete and computational geometry. chapter 24, pages 529–562. Chapman and Hall / CRC, Boca Raton, FL, USA, 2nd edition, 2004.
- [15] R. Liu. On a notion of data depth based on random simplices. Annals of Statistics, 18(1):405–414, 1990.
- [16] R. Liu and K. Singh. A quality index based on data depth and multivariate rank tests. *Journal of the American Statistical Association*, 88(421):252–260, 1993.

- [17] H. Oja. Descriptive statistics for multivariate distributions. Statistics and Probability Letters, 1(6):327–332, 1983.
- [18] K. Singh. A notion of majority depth. Technical report, Department of Statistics, Rutgers University, 1991.
- [19] C. Small. A survey of multidimensional medians. International Statistical Review, 58(3):263–277, 1990.
- [20] G. Tóth. Point sets with many k-sets. Discrete & Computational Geometry, 26(2):187–194, 2001.
- [21] J. W. Tukey. Mathematics and the picturing of data. In Proceedings of the International Congress of Mathematicians: Vancouver, volume 2, pages 523–531, Montreal, 1975. Canadian Mathematical Congress.

Exact Algorithms and APX-Hardness Results for Geometric Set Cover

Timothy M. Chan^{*}

Elvot Grant[†]

Abstract

We study several geometric set cover problems in which the goal is to compute a minimum cover of a given set of points in Euclidean space by a family of geometric objects. We give a short proof that this problem is APX-hard when the objects are axis-aligned fat rectangles, even when each rectangle is an ϵ -perturbed copy of a single unit square. We extend this result to several other classes of objects including almost-circular ellipses, axis-aligned slabs, downward shadows of line segments, downward shadows of graphs of cubic functions, 3-dimensional unit balls, and axis-aligned cubes, as well as some related hitting set problems. Our hardness results are all proven by encoding a highly structured minimum vertex cover problem which we believe may be of independent interest.

In contrast, we give a polynomial-time dynamic programming algorithm for 2-dimensional set cover where the objects are pseudodisks containing the origin or are downward shadows of pairwise 2-intersecting *x*-monotone curves. Our algorithm extends to the weighted case where a minimum-cost cover is required.

1 Introduction

In a geometric set cover problem, we are given a range space (X, S)—a universe X of points in Euclidean space and a pre-specified configuration S of regions or geometric objects. The goal is to select a minimum-cardinality subfamily $C \subseteq S$ such that each point in X lies inside at least one region in C. In the related hitting set problem, the goal is instead to select a minimum cardinality subset $Y \subseteq X$ such that each set in S contains at least one point in Y. In the weighted generalizations of these problems, we are also given a vector of positive costs $\mathbf{w} \in \mathbb{R}^S$ or $\mathbf{w} \in \mathbb{R}^X$ and we wish to minimize the total cost of all objects in C or Y respectively. Instances without costs (or with unit costs) are termed unweighted.

Geometric covering problems have found many applications to real-world engineering and optimization problems in areas such as wireless network design, image compression, and circuit-printing [11] [15]. Unfortunately, even for very simple classes of objects such as unit disks or unit squares in the plane, computing the exact minimum set cover is strongly NP-hard [18]. Consequently, much of the research surrounding geometric set cover has focused on approximation algorithms. A large number of constant and almost-constant approximation algorithms have been obtained for various hitting set and set cover problems of low VC-dimension via ϵ -net based methods [8] [13]. These methods have spawned a rich literature concerning techniques for obtaining small ϵ -nets for various weighted and unweighted geometric range spaces [12] [1] [22]. Results include constant-factor linear programming based approximation algorithms for set cover with objects like fat rectangles in the plane and unit cubes in \mathbb{R}^3 .

However, these approaches have limitations. So far, ϵ -net based methods have been unable to produce anything better than constant-factor approximations, and typically the constants involved are quite large. Their application is also limited to problems involving objects with combinatorial restrictions such as low union complexity (see [12] for details). A recent construction due to Pach and Tardos has proven that small ϵ -nets need not always exist for instances of the rectangle cover problem—geometric set cover where the objects are axisaligned rectangles in the plane [20]. In fact, their result implies that the integrality gap of the standard set cover LP for the rectangle cover problem can be as big as $\Theta(\log n)$. Despite this, a constant approximation using other techniques has not been ruled out.

The approximability of problems like rectangle cover also has connections to related capacitated covering problems [10]. Recently, Bansal and Pruhs used these connections, along with a weighted ϵ -net based algorithm of Varadarajan [22], to obtain a breakthrough in approximating a very general class of machine scheduling problems by reducing them to a weighted covering problem involving points 4-sided boxes in \mathbb{R}^3 —axisaligned cuboids abutting the xy and yz planes [9]. The 4-sided box cover problem generalizes the rectangle cover problem in \mathbb{R}^2 and thus inherits its difficulty.

In light of the drawbacks of ϵ -net based methods, Mustafa and Ray recently proposed a different approach. They gave a PTAS for a wide class of unweighted geometric hitting set problems (and consequently, related set cover problems) via a *local search* technique [19]. Their method yields PTASs for:

• Geometric hitting set problems involving half-

^{*}David R. Cheriton School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca

 $^{^\}dagger Department$ of Combinatorics and Optimization, University of Waterloo, <code>egrant@uwaterloo.ca</code>

spaces in \mathbb{R}^3 and pseudodisks (including disks, axisaligned squares, and more generally homothetic copies of identical convex regions) in the plane.

By implication, geometric set cover problems with lower half-spaces in R³ (by geometric duality, see [5]), disks in R² (by a standard lifting transformation that maps disks to lower halfspaces in R³, see [5]), and translated copies of identical convex regions in the plane (again, by duality).

Their results currently do not seem applicable to set cover with general pseudodisks in the plane. On a related note, Erlebach and van Leeuwen have obtained a PTAS for the weighted version of geometric set cover for the special case of unit squares [14].

1.1 Our Results

We present two main results—a series of APX-hardness proofs for several geometric set cover and related hitting set problems, and a polynomial-time exact algorithm for a different class of geometric set cover problems.

For a set Y of points in the plane, we define the *downward shadow* of Y to be the set of all points (a, b) such that there is a point $(a, y) \in Y$ with $y \ge b$.

Theorem 1 Unweighted geometric set cover is APXhard with each of the following classes of objects:

- (C1) Axis-aligned rectangles in \mathbb{R}^2 , even when all rectangles have lower-left corner in $[-1, -1+\epsilon] \times [-1, -1+\epsilon]$ and upper-right corner in $[1, 1+\epsilon] \times [1, 1+\epsilon]$ for an arbitrarily small $\epsilon > 0$.
- (C2) Axis-aligned ellipses in \mathbb{R}^2 , even when all ellipses have centers in $[0, \epsilon] \times [0, \epsilon]$ and major and minor axes of length in $[1, 1 + \epsilon]$.
- (C3) Axis-aligned slabs in \mathbb{R}^2 , each of the form $[a_i, b_i] \times [-\infty, \infty]$ or $[-\infty, \infty] \times [a_i, b_i]$.
- (C4) Axis-aligned rectangles in \mathbb{R}^2 , even when the boundaries of each pair of rectangles intersect exactly zero times or four times.
- (C5) Downward shadows of line segments in \mathbb{R}^2 .
- (C6) Downward shadows of (graphs of) univariate cubic functions in ℝ².
- (C7) Unit balls in \mathbb{R}^3 , even when all the balls contain a common point.
- (C8) Axis-aligned cubes in \mathbb{R}^3 , even when all the cubes contain a common point and are of similar size.
- (C9) Half-spaces in \mathbb{R}^4 .

Additionally, unweighted geometric hitting set is APX-hard with each of the following classes of objects:

- (H1) Axis-aligned slabs in \mathbb{R}^2 .
- (H2) Axis-aligned rectangles in R², even when the boundaries of each pair of rectangles intersect exactly zero times or four times.
- (H3) Unit balls in \mathbb{R}^3 .
- (H4) Half-spaces in \mathbb{R}^4 .

Mustafa and Ray ask if their local improvement approach might yield a PTAS for a wider class of instances; Theorem 1 immediately rules this out for all of the covering and hitting set problems listed above by proving that no PTAS exists for them unless P = NP. Item (C1) demonstrates that even tiny perturbations can destroy the behaviour of the local search method. (C2) rules out the possibility of a PTAS for arbitrarily fat ellipses (that is, ellipses that are within ϵ of being perfect circles). (C5) and (C6) stand in contrast to our algorithm below, which proves that geometric set cover is polynomial-time solvable when the objects are downward shadows of horizontal line segments or quadratic functions. In the case of (C4) and (H2), the intersection graph of the rectangles is a comparability graph (and hence a perfect graph); even then, neither set cover nor hitting set admits a PTAS. (C7), (C8), (C9), (H3), and (H4) complement the result of Mustafa and Ray by showing that their algorithm fails in higher dimensions.

All of our hardness results are proven by directly encoding a restricted version of unweighted set cover, which we call *SPECIAL-3SC*:

Definition 2 In an instance of SPECIAL-3SC, we are given a universe $U = A \cup W \cup X \cup Y \cup Z$ comprising disjoint sets $A = \{a_1, \ldots, a_n\}$, $W = \{w_1, \ldots, w_m\}$, $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_m\}$, and $Z = \{z_1, \ldots, z_m\}$ where 2n = 3m. We are also given a family S of 5msubsets of U satisfying the following two conditions:

- For each $1 \leq t \leq m$, there are integers $1 \leq i < j < k \leq n$ such that S contains the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{a_j, x_t, y_t\}$, $\{y_t, z_t\}$, and $\{a_k, z_t\}$ (summing over all t gives the 5m sets contained in S.)
- For all 1 ≤ t ≤ n, the element a_t is in exactly two sets in S.

In section 2, we show:

Lemma 3 SPECIAL-3SC is APX-hard.

Our second result is a dynamic programming algorithm that exactly solves weighted geometric set cover with various simple classes of objects:

Theorem 4 There exists a polynomial-time exact algorithm for the weighted geometric set cover problem involving downward shadows of pairwise 2-intersecting x-monotone curves in \mathbb{R}^2 . Moreover, it runs in
$O(mn^2(m+n))$ time on a set system consisting of n points and m regions.

Our algorithm is a generalization and simplification of a similar algorithm appearing in [10] for a combinatorial problem equivalent to geometric set cover with downward shadows of horizontal line segments in \mathbb{R}^2 . We believe that our current presentation is much shorter and cleaner; in particular, we do not require shortest path as a subroutine. We can also extend our algorithm to some related geometric set systems:

Corollary 5 There exists a polynomial-time exact algorithm for the weighted geometric set cover problem involving a configuration of pseudodisks in \mathbb{R}^2 where the origin lies within the interior of each pseudodisk. Furthermore, it runs in $O(mn^2(m+n))$ time on a set system consisting of n points and m pseudodisks.

Proof. Via the topological sweep given in Lemma 2.11 of [4], we transform the arrangement of pseudodisks into a topologically equivalent one in which the pseudodisks are star-shaped about the origin. We note that the transformation can be completed in $O(m^2 + mn)$ time assuming a representation allowing appropriate primitive operations. We then map the star-shaped pseudodisks to the downward shadows of 2-intersecting x-monotone functions on $[0, 2\pi)$ via a polar-to-cartesian transformation, enabling us to apply Theorem 4.

1.2 Related Work

The problem of assembling a given rectilinear polygon from a minimum number of (possibly overlapping) axisaligned rectangles was first proven to be MAX-SNPcomplete by Berman and Dasgupta [6], which rules out the possibility of a PTAS unless P = NP. Since set cover with axis-aligned rectangles can encode these instances, it too is MAX-SNP-complete. However, the proof in [6] cannot be applied to produce an instance of geometric set cover using only fat rectangles.

In his recent Ph.D. thesis, van Leeuwen proves APXhardness for geometric set cover and dominating set with axis-aligned rectangles and ellipses in the plane [23]. Har-Peled provides a simple proof that set cover with triangles is APX-hard, even when all triangles are fat and of similar size [16]. Har-Peled also notes that set cover with circles (that is, with boundaries of disks) is APX-hard for a similar reason. However, neither the results of van Leeuwen nor Har-Peled can be directly extended to fat axis-aligned rectangles or fat ellipses.

There are few non-trivial examples of geometric set cover problems that are known to be poly-time solvable. Har-Peled and Lee give a dynamic programming algorithm for weighted cover of points in the plane by half-planes [17]; their method runs in $O(n^5)$ time on an instance with *n* points and half-planes. Our algorithm both generalizes theirs and reduces the run time by a factor of n. Ambühl et al. give a poly-time dynamic programming algorithm for weighted covering of points in a narrow strip using unit disks [3]; their method appears to be unrelated to ours.

An interesting PTAS result is that of Har-Peled and Lee, who give a PTAS for minimum weight cover with any class of fat objects, provided that each object is allowed to expand by a small amount [17]. Our results show that without allowing this, a PTAS cannot be obtained.

2 APX-Hardness of SPECIAL-3SC

In this section, we prove Lemma 3. We recall that a pair of functions (f, g) is an L-reduction from a minimization problem A to a minimization problem B if there are positive constants α and β such that for each instance x of A, the following hold:

- (L1) The function f maps instances of A to instances of B such that $OPT(f(x)) \leq \alpha \cdot OPT(x)$.
- (L2) The function g maps feasible solutions of f(x) to feasible solutions of x such that $c_x(g(y)) OPT(x) \leq \beta \cdot (c_{f(x)}(y) OPT(f(x)))$, where c_x and $c_{f(x)}$ are the cost functions of the instances x and f(x) respectively.

We exhibit an L-reduction from minimum vertex cover on 3-regular graphs (hereafter known as 3VC) to SPECIAL-3SC. Since 3VC is APX-hard [2], this proves that SPECIAL-3SC is APX-hard (see [21] for details).

Given an instance x of 3VC on edges $\{e_1, \ldots, e_n\}$ with vertices $\{v_1, \ldots, v_m\}$ where 3m = 2n, we define f(x) be the SPECIAL-3SC instance containing the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{a_j, x_t, y_t\}$, $\{y_t, z_t\}$, and $\{a_k, z_t\}$ for each 4tuple (t, i, j, k) such that v_t is a vertex incident to edges e_i , e_j , and e_k with i < j < k. To define g, we suppose we are given a solution y to the SPECIAL-3SC instance f(x). We take vertex v_t in our solution g(y) of the 3VC instance x if and only if at least one of $\{a_i, w_t\}$, $\{a_j, x_t, y_t\}$, or $\{a_k, z_t\}$ is taken in y. We observe that g maps feasible solutions of f(x) to feasible solutions of x since e_i is covered in g(y) whenever a_i is covered in y.

Our key observation is the following:

Proposition 6 OPT(f(x)) = OPT(x) + 2m.

Proof. For $1 \leq t \leq m$, let $\mathcal{P}_t = \{\{w_t, x_t\}, \{y_t, z_t\}\}$ and $\mathcal{Q}_t = \{\{a_i, w_t\}, \{a_j, x_t, y_t\}, \{a_k, z_t\}\}$. Call a solution \mathcal{C} of f(x) segregated if for all $1 \leq t \leq m$, \mathcal{C} either contains all sets in \mathcal{P}_t and no sets in \mathcal{Q}_t , or contains all sets in \mathcal{Q}_t and no sets in \mathcal{P}_t .

Via local interchanging, we observe that there exists an optimal solution to f(x) that is segregated. Additionally, our function g, when restricted to segregated solutions of f(x), forms a bijection between them and feasible solutions of x. We check that g maps segregated solutions of size 2m + k to solutions of x having cost precisely k, and the result follows.

Proposition 6 implies that f satisfies property (L1) with $\alpha = 5$, since $\operatorname{OPT}(x) \geq \frac{m}{2}$. Moreover, $c_x(g(y)) + 2m \leq c_{f(x)}(y)$ since both $\{w_t, x_t\}$ and $\{y_t, z_t\}$ must be taken in y whenever v_t is not taken in g(y), and at least three of $\{\{a_i, w_t\}, \{w_t, x_t\}, \{a_j, x_t, y_t\}, \{y_t, z_t\}, \{a_k, z_t\}\}$ must be taken in y whenever v_t is taken in g(y). Together with Proposition 6, this proves that g satisfies property (L2) with $\beta = 1$. Thus (f, g) is an L-reduction.

3 Encodings of SPECIAL-3SC via Geometric Set Cover

In this section, we prove Theorem 1 using Lemma 3, by encoding instances of various classes of geometric set cover and hitting set problems as instances of SPECIAL-3SC. The beauty of SPECIAL-3SC is that it allows many of our geometric APX-hardness results to follow immediately from simple "proofs by pictures" (see Figure 3). The key property of SPECIAL-3SC is that we can divide the elements into two sets A and $B = W \cup X \cup Y \cup Z$, and linearly order B in such a way that all sets in S are either two adjacent elements from B, one from B and one from A, or two adjacent elements from B and one from A. We need only make $[w_t, x_t, y_t, z_t]$ appear consecutively in the ordering of B.

For (C1), we simply place the elements of A on the line segment $\{(x, x - 2) : x \in [1, 1 + \epsilon]\}$ and place the elements of B, in order, on the line segment $\{(x, x + 2) : x \in [-1, -1 + \epsilon]\}$, for a sufficiently small $\epsilon > 0$. As we can see immediately from Figure 3, each set in S can be encoded as a fat rectangle in the class (C1).

(C2) is similar. It is not difficult to check that each set can be encoded as a fat ellipse in this class.

For (C3), we place the elements of A on a horizontal line (the top row). For each $1 \leq t \leq m$, we create a new row containing $\{w_t, x_t\}$ and another containing $\{y_t, z_t\}$ as shown in Figure 3. This time, we will need the second property in Definition 2—that each a_i appears in two sets. If $\{a_i, w_t\}$ is the first set that a_i appears in, we place w_t slightly to the left of a_i ; if it is the second set instead, we place w_t slightly to the right of a_i . Similarly, the placement of x_t, y_t (resp. w_t) depends on whether a set of the form $\{a_j, x_t, y_t\}$ (resp. $\{a_k, w_t\}$) is the first or second set that a_j (resp. a_k) appears in. As we can see from Figure 3, each set in S can be encoded as a thin vertical or horizontal slab.

(C4) is similar to (C3), with the slabs replaced by thin rectangles. For example, if $\{a_i, w_t\}$ and $\{a_i, w_{t'}\}$ are the two sets that a_i appears in, with w_t located higher than $w_{t'}$, we can make the rectangle for $\{a_i, w_t\}$ slightly wider than the rectangle for $\{a_i, w_{t'}\}$ to ensure that these two rectangles intersect 4 times.

For (C5), we can place the elements of A on the ray $\{(x, -x) : x > 0\}$ and the elements of B, in order, on the ray $\{(x, x) : x < 0\}$. The sets in S can be encoded as downward shadows of line segments as in Figure 3.

(C6) is similar to (C5). One way is to place the elements of A on the line segment $\ell_A = \{(x,x) : x \in [-1, -1 + \epsilon]\}$ and the elements of B (in order) on the line segment $\ell_B = \{(x,0) : x \in [1.5, 1.5 + \epsilon]\}$. For any $a \in [-1, -1 + \epsilon]$ and $b \in [1.5, 1.5 + \epsilon]$, the cubic function $f(x) = (x - b)^2[(a + b)x - 2a^2]/(b - a)^3$ is tangent to ℓ_A and ℓ_B at x = a and x = b. (The function intersects y = 0 also at $x = 2a^2/(a + b) \gg 1.5 + \epsilon$, far to the right of ℓ_B .) Thus, the size-2 sets in S can be encoded as cubics. A size-3 set $\{a_j, x_t, y_t\}$ can also be encoded if we take a cubic with tangents at a_j and x_t , shift it upward slightly, and make x_t and y_t sufficiently close.

For (C7), we place the elements in A on a circular arc $\gamma_A = \{(x, y, 0) : x^2 + y^2 \leq 1, x, y \geq 0\}$ and the elements in B (in order) on the vertical line segment $\ell_B = \{(0, 0, z) : z \in [1-2\epsilon, 1-\epsilon]\}$. (This idea is inspired by a known construction [7], after much simplification.) We can ensure that every two points in A have distance $\Omega(\sqrt{\epsilon})$ if $\epsilon \ll 1/n^2$. The technical lemma below allows us to encode all size-2 sets (by setting b = b') and size-3 sets by unit balls containing a common point.

Lemma 7 Given any $a \in \gamma_A$ and $b, b' \in \ell_B$, there exists a unit ball that (i) intersects γ_A at an arc containing a of angle $O(\sqrt{\epsilon})$, (ii) intersects ℓ_B at precisely the segment from b to b', and (iii) contains $(1/\sqrt{2}, 1/\sqrt{2}, 1)$.

Proof. Say a = (x, y, 0), b = (0, 0, z - h), b' = (0, 0, z + h). Consider the unit ball K centered at $c = ((1 - h^2)x, (1 - h^2)y, z)$. Then (ii) is self-evident and (iii) is straightforward to check. For (i), note that a lies in K since $||a - c||^2 = h^2 + z^2 \le \epsilon^2 + (1 - \epsilon)^2 < 1$. On the other hand, if a point $p \in \gamma_A$ lies in the unit ball, then letting $a' = ((1 - h^2)x, (1 - h^2)y, 0)$, we have $||p - c||^2 = ||p - a'||^2 + z^2 \le 1$, implying $||p - a|| \le ||p - a'|| + ||a' - a|| \le \sqrt{1 - z^2} + h = O(\sqrt{\epsilon})$.

(C8) is similar to (C1); we place the elements in Aon the line segment $\ell_A = \{(t, t, 0) : t \in (0, 1)\}$ and the elements in B on the line segment $\ell_B = \{(0, 3-t, t) : t \in (0, 1)\}$. For any $(a, a, 0) \in \ell_A$ and $(0, 3-b, b) \in \ell_B$, the cube $[-3+b+2a, a] \times [a, 3-b] \times [-3+a+2b, b]$ is tangent to ℓ_A at (a, a, 0), is tangent to ℓ_B at (0, 3-b, b), and contains (0, 1, 0). Size-3 sets $\{a_j, x_t, y_t\}$ can be encoded by taking a cube with tangents at a_j and x_t , expanding it slightly, and making x_t and y_t sufficiently close.

(C9) follows from (C7) by the standard lifting transformation [5].

For (H1), we map each element a_i to a thin vertical slab. For each $1 \le t \le m$, we map $\{w_t, x_t, y_t, z_t\}$ to a



Figure 1: APX-hardness proofs of geometric set cover problems.

cluster of four thin horizontal slabs as in Figure 3. Each set in S can be encoded as a point in the arrangement.

- (H2) is similar; see Figure 3.
- (H3) follows from (C7) by duality.
- (H4) follows from (C9) by duality.

4 Algorithm for Weighted Covering by Downward Shadows of 2-Intersecting *x*-Monotone Curves

Here, we prove Theorem 4 by giving a polynomialtime dynamic programming algorithm for the weighted cover of a finite set of points $X \subseteq \mathbb{R}^2$ by a set S of downward shadows of 2-intersecting x-monotone curves C_1, \ldots, C_m . For $1 \leq i \leq m$, define the region $S_i \in S$ to be the downward shadow of the curve C_i and let it have positive cost w_i . Define n = |X|.

We shall assume that each C_i is the graph of a smooth univariate function with domain $[-\infty, \infty]$, that all intersections are transverse (no pair of curves intersect tangentially), and that no points in X lie on any curve C_i . It is not difficult to get around these assumptions, but we retain them to simplify our explanation.

We shall abuse notation by writing $C_i(x)$ for the unique $y \in \mathbb{R}$ such that (x, y) lies on the curve C_i . We say curve C_i is wider than curve C_j (written $C_i \succ C_j$) whenever $C_i(x) > C_j(x)$ for all sufficiently large x. We may also write $S_i \succ S_j$ whenever $C_i \succ C_j$. We note that \succ is a total ordering and thus we can order all curves by width, so we assume without loss of generality that $C_i \succ C_j$ whenever i > j. The width-based ordering of curves is useful because of the following key observation:

Proposition 8 If $C_i \succ C_j$, then $S_j \setminus S_i$ is connected.

Proof. This is clearly true if C_i and C_j intersect once or less. If C_i and C_j intersect transversely twice—say, at (x_1, y_1) and (x_2, y_2) with $x_2 > x_1$ —then the area above C_i but below C_j can only be disconnected if $C_j(x) >$ $C_i(x)$ for $x < x_1$ and $x > x_2$, implying $C_j \succ C_i$. \Box

For all $1 \leq i \leq m$ and all intervals [a, b], define X[a, b] to be all points in X with x-coordinate in [a, b], and define X[a, b, i] to be $X[a, b] \setminus S_i$. Define $S_{<i}$ to be the set $\{S_1, \ldots, S_{i-1}\}$ of all regions of width less than S_i . Let M[a, b, i] denote the minimum cost of a solution to the weighted set cover problem on the set system $(X[a, b, i], S_{<i})$ (with weights inherited from the original problem). If such a covering does not exist, $M[a, b, i] = \infty$. For simplicity, we assume that C_m , the widest curve, contains no points in its downward shadow (that is, $X \cap S_m$ is empty). Our goal is then to determine $M[-\infty, \infty, m]$ via dynamic programming; the key structural result we need is the following:

Proposition 9 If X[a, b, i] is non-empty, then

$$M[a, b, i] = \min \left\{ \min_{c \in (a, b)} \{ M[a, c, i] + M[c, b, i] \}, \\ \min_{j \le i} \{ M[a, b, j] + w_j \} \right\}.$$

Proof. Clearly $M[a, b, i] \leq M[a, c, i] + M[c, b, i]$ for all $c \in (a, b)$. Also, for j < i, $M[a, b, j] + w_i$ is the cost of

purchasing S_j and then covering the remaining points in X[a, b] using regions less wide than S_j (and hence less wide than S_i). Thus $M[a, b, j] + w_j$ is a cost of a feasible solution to $(X[a, b, i], S_{< i})$ and hence is at least M[a, b, i]. It follows that M[a, b, i] is bounded above by the right hand side.

To show that M[a, b, i] is bounded below by the right hand side, we let $\mathcal{Z} \subseteq \mathcal{S}_{\langle i \rangle}$ be a feasible set cover for $(X[a, b, i], \mathcal{S}_{\langle i \rangle})$. We consider two cases:

Case 1: There is some $c \in (a, b)$ such that $(c, C_i(c))$ is not covered by \mathcal{Z} . Let $\mathcal{Z}_{<c}$ be the set of all regions in \mathcal{Z} containing a point in X[a, c, i], and let $\mathcal{Z}_{>c}$ be the set of all regions in \mathcal{Z} containing a point in X[c, b, i]. Let $Z \in \mathcal{Z}$. Since $Z \prec S_i$, by Proposition 8, $Z \setminus S_i$ is connected and thus cannot contain points both in X[a, c, i] and X[c, b, i]. Hence $\mathcal{Z}_{<c} \cap \mathcal{Z}_{>c} = \emptyset$ and thus the cost of \mathcal{Z} is at least M[a, c, i] + M[c, b, i].

Case 2: For all $c \in (a, b)$, the point $(c, C_i(c))$ is covered by \mathcal{Z} . Then \mathcal{Z} covers $X[a, b, i] \cup S_i$ and hence covers all points in X[a, b]. Let C_j be the widest curve in \mathcal{Z} , noting that j < i. Then the cost of \mathcal{Z} is at least $w_j + M[a, b, j]$ since $\mathcal{Z} \setminus S_j$ must cover all points in X[a, b, j].

It follows that \mathcal{Z} must cost as much as either $\min_{c \in (a,b)} \{ M[a,c,i] + M[c,b,i] \}$ or $\min_{j < i} \{ M[a,b,j] + w_j \}$, and the result follows.

Proposition 9 immediately implies the existence of a dynamic programming algorithm to compute $M[-\infty, \infty, m]$ and return a cover having that cost. There are at most n + 1 combinatorially relevant values of a and b when computing optimal costs M[a, b, i]for subproblems, so there are $O(mn^2)$ distinct values of M[a, b, i] to compute. Recursively computing M[a, b, i]requires O(m + n) table lookups, so the total runtime of our algorithm is $O(mn^2(m + n))$, assuming a representation allowing primitive operations in O(1) time.

References

- B. Aronov, E. Ezra, and M. Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. In ACM Symposium on Theory of Computing (2009), 639-648.
- [2] P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoretical Comp. Sci.* 237 (2000) 123-134.
- [3] C. Ambühl, T. Erlebach, M. Mihalák, and M. Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In APPROX and RANDOM (2006) 3-14.
- [4] P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky. Lenses in arrangements of pseudocircles and their applications. J. ACM 51(2) (2004), 139-186.
- [5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Ap-

plications (Third edition). Springer-Verlag, Heidelberg, (2008).

- [6] P. Berman and B. DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica* 17(4) (1997), 331-356.
- [7] H. Brönnimann and O. Devillers The union of unit balls has quadratic complexity, even if they all contain the origin. arXiv:cs/9907025v1 [cs.CG] (1999).
- [8] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.* 14 (1995), 263-279.
- [9] N. Bansal and K. Pruhs. The geometry of scheduling. IEEE 51st Annual Symposium on Foundations of Computer Science (2010), 407-414.
- [10] D. Chakrabarty, E. Grant, and J. Koenemann. On column-restricted and priority covering integer programs. In *Integer Programming and Combinatorial Optimization* (2010) 355-368.
- [11] Y. Cheng, S.S. Iyengar, and R.L. Kashyap. A new method for image compression using irreducible covers of maximal rectangles. *IEEE Trans. Software Engrg.* 14 (1988), 651-658.
- [12] K. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete Comput. Geom.* 37 (2007), 43-58.
- [13] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Information Processing Let*ters 95(2) (2005), 358-362.
- [14] T. Erlebach and E. J. van Leeuwen PTAS for weighted set cover on unit squares. In APPROX and RANDOM (2010), 166-177.
- [15] A. Hegedüs. Algorithms for covering polygons with rectangles. Comput. Aided Geom. Design 14 (1982), 257-260.
- [16] S. Har-Peled. Being Fat and Friendly is Not Enough. arXiv:0908.2369v1 [cs.CG] (2009).
- [17] S. Har-Peled and M. Lee. Weighted geometric set cover problems revisited. Unpublished manuscript, (2008).
- [18] D.S. Hochbaum and W. Maass. Fast approximation algorithms for a nonconvex covering problem. J. Algorithms 8(3) (1987), 305-323.
- [19] N.H. Mustafa and S Ray. Improved results on geometric hitting set problems. *Discrete Comput. Geom.* 44(4) (2010), 883-895.
- [20] J. Pach, G. Tardos. Tight lower bounds for the size of epsilon-nets. In Proc. 27th ACM Sympos. Comput. Geom. (2011) 458-463.
- [21] C.H. Papadimitriou, M. Yannakakis. Optimization, approximation, and complexity classes. J. Comput. Systems Sci. 43 (1991), 425-440.
- [22] K. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In ACM Symposium on Theory of Computing (2010) 641-648.
- [23] E. J. van Leeuwen. Optimization and Approximation on Systems of Geometric Objects. PhD thesis, Universiteit van Amsterdam, (2009).

Enumerating Minimal Transversals of Geometric Hypergraphs

Khaled Elbassioni*

Imran Rauf[†]

Saurabh Ray[‡]

Abstract

We consider the problem of enumerating all minimal hitting sets of a given hypergraph (V, \mathcal{R}) , where V is a finite set, called the *vertex set* and \mathcal{R} is a set of subsets of V called the hyperedges. We show that, when the hypergraph admits a *balanced subdivision*, then a recursive decomposition can be used to obtain efficiently all minimal hitting sets of the hypergraph. We apply this decomposition framework to get incremental polynomial-time algorithms for finding minimal hitting sets and minimal set covers for a number of hypergraphs induced by a set of points and a set of geometric objects. The set of geometric objects includes half-spaces, hyper-rectangles and balls, in fixed dimension. A distinguishing feature of the algorithms we obtain is that they admit an efficient global parallel implementation, in the sense that all minimal hitting sets can be generated in *polylogarith*mic time in |V|, $|\mathcal{R}|$ and the total number of minimal transversals T, using a polynomial number of processors.

1 Introduction

Let (V, \mathcal{R}) be a hypergraph, where V is a finite set called the *vertices* and \mathcal{R} is a family of subsets of V called the *hyperedges*. We say a vertex $v \in V$ hits hyperedge $R \in \mathcal{R}$ when $v \in R$. A subset of vertices $X \subseteq V$ is said to be a hitting set (or transversal) for \mathcal{R} if every hyperedge R in \mathcal{R} is hit by an element x in X. A hitting set is minimal if none of its proper subsets is a hitting set.

In this paper, we will consider hypergraphs induced by a set of points and certain families of geometric objects in \mathbb{R}^d . When the hypergraph is arbitrary, we obtain the well-known hypergraph transversal or dualization problem [1], which calls for finding all minimal hitting sets for a given hypergraph. This problem has received considerable attention in the literature (see, e.g., [2, 8, 9, 17, 22, 24]), since it is known to be polynomially or quasi-polynomially equivalent with many problems in various areas.

Clearly, the number T of minimal transversals of a hypergraph (V, \mathcal{R}) can be exponential in |V| and $|\mathcal{R}|$, and hence one can only hope for an algorithm whose efficiency is measured in terms of the parameters |V|, $|\mathcal{R}|$ and T. The currently fastest known algorithm [15] for solving the hypergraph transversal problem runs in quasi-polynomial time $|V|N^{o(\log N)}$, where N is the combined input and output size $N = |\mathcal{R}| + T$. Several quasipolynomial time algorithms with some other desirable properties also exist [4, 12, 13, 16, 25]. While it is still open whether the problem can be solved in polynomial time for arbitrary hypergraphs, polynomial time algorithms exist for several classes of hypergraphs, e.g. hypergraphs of bounded edge-size [3, 8], of bounded degree [7, 9], of bounded-edge intersections [3], of bounded conformality [3], of bounded treewidth [9], and read-once (exact) hypergraphs [11].

Recently [14], this polynomiality frontier has been extended to a number of geometric hypergraphs. In particular, polynomial-time algorithms were given for enumerating minimal hitting sets and minimal set covers, when the hyperedges are induced by hyper-rectangles (in \mathbb{R}^d for fixed d), or half-planes (in \mathbb{R}^2). We observe that the algorithms in [14] depend on the types of ranges considered. For instance, in the case of rectangles, two different types of algorithms are used for the hitting set and set covering versions, and a substantially modified algorithm is needed for half-planes. Furthermore, it is not clear, how these algorithms can be generalized to other geometric objects, such as balls, or half-spaces in fixed dimension $d \geq 3$.

In this paper, we present a simple and unified algorithm that works for both the hitting set and set covering versions, and extends to more general objects, such as balls, half-spaces, and polytopes with fixed number of facets. In fact, as we will see, all that is needed is that the hypergraph admits a certain *balanced subdivi*sion which can be shown to exist in several geometric instances. One more important property of the algorithms we obtain, is that they admit a global parallel implementation, in the sense that all minimal hitting sets can be generated in polylogarithmic time (in |V|, $|\mathcal{R}|$ and the total number of minimal transversals T) using a polynomial number of processors (in the PRAM model). Among all polynomially solvable classes of hypergraphs, only very few are known to exhibit this nice property, see [20, 21]. We remark that the general algorithms given in [4, 13] do not satisfy this global paral-

^{*}Max-Planck-Institut für Informatik, Saarbrücken, Germany, elbassio@mpi-inf.mpg.de

[†]Friedrich-Schiller-Universität, Jena, Germany, imran.rauf@uni-jena.de

[‡]Max-Planck-Institut für Informatik, Saarbrücken, Germany, saurabh@mpi-inf.mpg.de

lelism property, in the sense that they only produce *each* output in parallel, that is, the time needed to produce T minimal transversals is *polylogarithmic* in $|V|, |\mathcal{R}|$, but can be *linear* (or super-linear) in T. Finding a global parallel algorithm (even with a quasi-polynomial number of processors) for the general case is an outstanding open problem, and is very useful in practice since the number of minimal transversals is typically very large.

Theorem 1 Let (V, \mathcal{R}) be a hypergraph defined by a set of points $P \in \mathbb{R}^d$ and set of ranges $\mathcal{R} \subseteq \mathbb{R}^d$. If \mathcal{R} is a set of half-planes, balls, or a polytopes with a fixed number of facets, in fixed dimension d = O(1), then there is an algorithm that enumerates all T minimal hitting sets (resp., set covers) of \mathcal{R} in parallel time polylog($|V|, |\mathcal{R}|, T$) on poly($|V|, |\mathcal{R}|, T$) number of processors.

Even though the bound in Theorem 1 is stated in terms of the total number of minimal hitting sets T, we will see that our algorithm can be modified to work in an *incremental setting*, i.e., for a given integer $T' \leq T$, it finds T' minimal hitting sets (resp., set covers) of \mathcal{R} in time polylog($|\mathcal{R}|, T'$) $\cdot \tau(|V|)$ on poly($|V|, |\mathcal{R}|, T'$) number of processors, where $\tau(|V|)$ is the time needed to find a single minimal hitting set. The currently best known parallel implementation for the latter problem [19] has $\tau(|V|) = O(\sqrt{|V|})$.

Our algorithm builds on and extends the *covering decomposition* approach initially suggested in [20, 21], and used in a number of subsequent works [12, 4]. So far, the use of such decompositions has been successful for developing polynomial time dualization algorithms for limited cases, such as hypergraphs of bounded size or bounded degree. In this paper, we show that the limitations of the previous approaches can be overcome using a modified version. This allows us to obtain a large class of hypergraphs for which covering decompositions are effective.

The enumeration of minimal geometric hitting sets, as the ones described above, arises in various areas such as computational geometry, machine learning, and data mining [10]. Moreover, our efficient enumeration algorithms might be useful in developing exact algorithms, fixed-parameter tractable algorithms, and polynomialtime approximation schemes for the corresponding optimization problems (see, e.g., [18]).

2 Notation

In this paper, we will often write \mathcal{R} for a hypergraph (V, \mathcal{R}) for notational convenience. Also, we will often refer to hypergraphs as range spaces in accordance with the terminology in the Computational Geometry literature. Accordingly, we will refer to the vertex set as

the ground set and the hyperedges as ranges. Given a range spaces \mathcal{R} , we will denote by $\operatorname{Tr}(\mathcal{R})$ the set of all minimal hitting sets of \mathcal{R} . Given a range (V, \mathcal{R}) , and a subset $V' \subseteq V$, we will denote by $\mathcal{R}|_{V'}$ the set $\{R \cap V' : R \in \mathcal{R}\}$. The hypergraph $(V', \mathcal{R}_{V'})$ is called the projection of the hypergraph (V, \mathcal{R}) on V'.

3 Balanced Subdivisions

Given any range space (V, \mathcal{R}) , we say that a subset $V' \subseteq V$ is *stabbed* by a range $R \in \mathcal{R}$ if there exist $x, y \in V'$ s.t. $x \in R$ and $y \notin R$. A *balanced subdivision* for a range space (V, \mathcal{R}) is a collection of a constant number of subsets V_1, V_2, \ldots, V_k of V such that

- 1. For each $i \in \{1, ..., k\}, |V_i| \ge \epsilon |V|$ for some constant $0 < \epsilon \le 1$.
- 2. For each range $R \in \mathcal{R}$, there are two disjoint subsets V_i and V_j in the collection which are not stabled by R.

Remark 1 In the above definition, the fact that a subset $V' \subset V$ is not stabled by $R \in \mathcal{R}$ implies that V' is also not stabled by $V \setminus R$. Consequently, we conclude that a balanced subdivision for any range space (V, \mathcal{R}) is also a balanced subdivision for the range space (V, \mathcal{R}^c) , where \mathcal{R}^c denotes the hypergraph defined by compliments of ranges in \mathcal{R} , i.e., $\mathcal{R}^c = \{V \setminus R \mid R \in \mathcal{R}\}$.

In the next section, we show that if we can compute a balanced subdivision for a range space then we can enumerate its minimal hitting sets in global parallel time.

In this section, we show that several geometrically induced range spaces admit balanced subdivisions which can be computed efficiently (in parallel). We consider range spaces induced by a set of points P and a set of geometric objects \mathcal{H} in \mathbb{R}^d . There are two natural range spaces defined by them depending on whether we let the points or the objects form the ground set. We denote by (P, \mathcal{H}) the range space in which P is the ground set and each $H \in \mathcal{H}$ defines the range $H \cap P$. Similarly, we denote by (\mathcal{H}, P) the range space in which the ground set is \mathcal{H} and each $p \in P$ defines the range $\{H \in \mathcal{H} \mid p \in$ $H\}$.

We now show that for any point set $P \subset \mathbb{R}^d$, a balanced subdivision exists and can be computed efficiently for both (P, \mathcal{H}) and (\mathcal{H}, P) if \mathcal{H} is a family of objects of the following kind: (i) Half-Spaces in \mathbb{R}^d (ii) Polytopes with at most a constant number of facets in \mathbb{R}^d and (iii) Balls in \mathbb{R}^d .

We will use the following results:

Theorem 2 (Fine Simplicial Partitions [23])

Given any set P of n points in \mathbb{R}^d and any parameter $1 \leq r \leq n$, there exists a partition

 $\Pi = \{P_1, P_2, \dots, P_t\} \text{ of } t \leq r \text{ disjoint subsets of } P \text{ and a set } \Delta = \{\Delta_1, \dots, \Delta_t\} \text{ of simplices with the } following properties: (i) <math>P_i \subseteq \Delta_i$ (ii) $\bigcup_i P_i = P$, (iii) $n/r \leq |P_i| \leq 2n/r \text{ for all } i \in \{1, \dots, t\} \text{ and } (iv) \text{ no } half-space in <math>\mathbb{R}^d$ intersects more than $C_d r^{1-1/d}$ of the simplices in Δ , where C_d is a constant for any fixed d. The last property also implies that no half-space in \mathbb{R}^d stabs more than $C_d r^{1-1/d}$ of the sets in Π . Further, for any $\delta > 0$, such a Π can be computed in time $O(n^{1+\delta})$. When r is bounded by a constant, Π can be computed in O(n) time.

Theorem 3 (Cuttings [5]) Given any set of n halfspaces in \mathbb{R}^d and any parameter $1 \leq r \leq n$, there exists a partition of \mathbb{R}^d into r simplices such that none of the simplices is stabled by more than $C'_d n/r^{1/d}$ of the given half-spaces. Further, for any $\delta > 0$, such a partition can be computed deterministically in time $O(nr^{1-1/d})$.

Parallel Implementation: Even though we only mention sequential running times above, such fine simplicial partitions and cuttings can be computed in polylog(n) time using poly(n) processors. For example, in the case of cuttings in any fixed dimension d, while the simplices are allowed to be arbitrary, it can be argued that they can always be chosen so that they are among a polynomial number of *canonical* simplices. In fact, it is not hard to argue that we can restrict to simplices whose corners are defined by the intersection of d of the hyperplanes defining the given set of halfspaces and indeed the construction in [5] does restrict to such simplices. The number of such simplices is at most $O(n^{d(d+1)})$. Once we have a polynomial bound on the number of canonical simplices. we can check all possible sets of t canonical simplices in parallel using a polynomial number of processors and find a simplicial partition in polylogarithmic time. A similar argument holds for simplicial partitions.

Half-Spaces. Let us first consider the case when \mathcal{H} is a set of half-spaces in \mathbb{R}^d . Given any set P with n points in \mathbb{R}^d , we can set r to be a large enough constant so that $C_d r^{1-1/d} \leq r-2$. Then, clearly, the collection Π given by Theorem 2 also gives us a balanced subdivision of (P, \mathcal{H}) with $\epsilon = 1/r$ and k < r. To get a balanced subdivision of (\mathcal{H}, P) , we apply Theorem 3 to the halfspaces in \mathcal{H} . Assuming that $|\mathcal{H}| = n$, we set r to be a large enough constant so that $C'_d n/r^{1/d} \leq n/2$. Theorem 3 gives a partition of \mathbb{R}^d into r regions R_1, \ldots, R_r each of which is stabled by at most n/2 half-spaces in \mathcal{H} . Consequently, for each region R_i , we either have at least n/4 half-spaces of \mathcal{H} each of which contains R_i or we have at least n/4 half-spaces of \mathcal{H} none of which intersects R_i . Let \mathcal{H}_i be a set of those half-spaces. Then $|\mathcal{H}_i| \geq n/4$. We arbitrarily partition each \mathcal{H}_i into two disjoint sets \mathcal{H}_i^1 and \mathcal{H}_i^2 each of size at least n/8. These sets \mathcal{H}_i^1 and \mathcal{H}_i^2 for $i \in \{1, \ldots, r\}$ give us a balanced subdivision of (\mathcal{H}, P) with k = 2r and $\epsilon = 1/8$ since each point $p \in P$ lies in some region R_j and hence does not stab the disjoint sets \mathcal{H}_j^1 and \mathcal{H}_j^2 .

Remark 2 In the case of half-spaces, one may also reduce the problem of finding minimal set covers to that of finding minimal hitting sets by using geometric duality, which maps points in \mathbb{R}^d to hyperplanes and vice versa (see e.g. [6], Chapter 8). However this method does not work for polytopes.

Polytopes. Suppose now that \mathcal{H} is a set of polytopes in \mathbb{R}^d , each with at most f facets. In this case the collection Π given by Theorem 2 with r being a large enough constant so that $f \cdot C_d r^{1-1/d} \leq r-2$ gives us the required balanced subdivision for (P, \mathcal{H}) . This is because each facet of a polytope can stab at most $C_d r^{1-1/d}$ members of Π and therefore a polytope with at most ffacets can stab at most $f \cdot C_d r^{1-1/d} \leq r-2$ members of Π . To get a balanced subdivision for (\mathcal{H}, P) , we consider for each $H \in \mathcal{H}$, the set of at most f half-spaces whose intersection forms H. Let \mathcal{H}' be the set of all these half-spaces. Assuming that $|\mathcal{H}| = n$, we have that $|\mathcal{H}'| \leq fn$. We then invoke Theorem 3 for the halfspaces in \mathcal{H}' with r being a large enough constant so that $C'_d(nf)/r^{1/d} \leq n/2$. The regions in the resulting partition are stabled by at most n/2 half-spaces in \mathcal{H}' and hence at most n/2 polytopes in \mathcal{H} . We can then construct the balanced subdivision for (\mathcal{H}, P) consisting of sets \mathcal{H}_i^1 and \mathcal{H}_i^2 , $i \in \{1, \ldots, r\}$, as in the last paragraph.

Balls. Finally assume that \mathcal{H} is a set of balls in \mathbb{R}^d . There is a standard *lifting* (veronese map) which maps each point in \mathbb{R}^d to a point in \mathbb{R}^{d+1} and each ball in \mathbb{R}^d to a half-space in \mathbb{R}^{d+1} so that the incidence relations among them are preserved. Since balanced subdivisions exist for half-spaces in \mathbb{R}^{d+1} , we can conclude that balanced subdivisions exist for balls in \mathbb{R}^d as well (for both (P, \mathcal{H}) and (\mathcal{H}, P)). The same trick can be applied to other algebraically defined sets like ellipses, parabolas etc.

Since we invoke Theorems 2 and 3 with r being a constant, the collection in Theorem 2 and the partition in Theorem 3 can be computed in time O(n). It follows that balanced subdivisions for the above range spaces can be computed in O(n + m) time where n is the size of the ground set and m is the number of ranges.

4 The Enumeration Algorithm

Given a range space (V, \mathcal{R}) , a divide-and-conquer algorithm to enumerate all minimal hitting sets of \mathcal{R} is presented in Figure 1. If |V| is at most some fixed constant μ , all minimal hitting sets of \mathcal{R} can be enumerated by just enumerating all subsets of V and outputting those which form a minimal hitting set of \mathcal{R} . We assume the existence of a procedure Enumerate-Small for the enumeration of minimal hitting sets in these trivial cases.

Algorithm 1 Procedure Enumerate (V, \mathcal{R}) :

Input: A finite range space (V, \mathcal{R}) **Output:** The set of all minimal hitting sets of \mathcal{R} 1: if $|V| \leq \mu$ then 2: return Enumerate-Small (V, \mathcal{R}) 3: end if 4: Type1-Set:= \emptyset 5: Compute a balanced subdivision V_1, \ldots, V_{λ} of (V, \mathcal{R}) 6: **for** $i = 1, ..., \lambda$ **do** Type1-Set := Type1-Set \cup Enumerate($V \setminus$ 7: $V_i, \mathcal{R}|_{V \setminus V_i})$ 8: end for 9: Type1-Set := **Remove-Duplicates**(Type1-Set) 10: Type2-Set := \emptyset 11: **for** $i = 1, ..., \lambda$ **do** $\mathcal{X}_i := \mathbf{Enumerate}(V \setminus V_i, \mathcal{R}_i)$ 12:13: end for for each $(M_1, \ldots, M_\lambda) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_\lambda$ do 14:15: $M := \bigcup_i M_i$ if M is a type 2 minimal hitting set of \mathcal{R} then 16:Type2-Set := Type2-Set $\cup \{M\}$ 17:end if 18:19: end for 20: Type2-Set := **Remove-Duplicates**(Type2-Set) 21: return Type1-Set \cup Type2-Set

When $|V| > \mu$, we assume the existence of a balanced subdivision $\Pi = (V_1, \ldots, V_{\lambda})$, where λ is a constant and for each $i \in \{1, \ldots, \lambda\}$, $|V_i| \leq \epsilon |V|$ where $0 < \epsilon < 1$ is another constant. We classify the minimal hitting sets of \mathcal{R} into two types. Type 1 minimal hitting sets are those that have an empty intersection with one of the V_i s. The remaining minimal hitting sets which contain elements from each V_i are of type 2. Since each V_i contains a constant fraction of the elements in V, type 1 hitting sets are easily enumerated recursively. This is done in line 7 of Figure 1. Enumerating Type 2 minimal hitting sets require more work.

Let us first observe that any minimal hitting set Mof \mathcal{R} and for any $v \in M$, there is always some range $R \in \mathcal{R}$ which requires v, i.e., $R \cap M = \{v\}$. We call such a range a *certificate* range for v in M. Clearly, M is also a minimal hitting set for the set of certificate ranges of its elements.

Let M be any type 2 minimal hitting set and let $R \in \mathcal{R}$ be any range that has a nonempty intersection with each of the V_i s. Since Π is a balanced subdivision, there are at least two sets V_j and V_k which are not stabled by R. Since R has a nonempty intersection with both

of them, it must contain both the sets as subsets. Now, since M contains an element from each V_i , R contains at least two elements of M implying that R cannot be a certificate range for any element of M. This means that for the purpose of enumerating type 2 minimal hitting sets, we can discard all ranges which have a non-empty intersection with each of the V_i s. Let $\mathcal{R}_i = \{R \in \mathcal{R} :$ $R \cap V_i = \emptyset\}$ and let $\tilde{\mathcal{R}} = \bigcup_i \mathcal{R}_i$.

Let M be any type 2 minimal hitting set of \mathcal{R} . Since $\tilde{\mathcal{R}}$ contains all certificate ranges of M, M is also a minimal hitting set for $\tilde{\mathcal{R}}$. Also, since the ranges in \mathcal{R}_i do not contain any element of V_i , $M \setminus V_i$ is a hitting set (not necessarily minimal) for \mathcal{R}_i and therefore contains some $M_i \subseteq M \setminus V_i$ which is a minimal hitting set for \mathcal{R}_i .

Notice that each element of M appears in at least one of the M_i s. This is because each $v \in M$ has a certificate range R which belongs to some \mathcal{R}_i implying that $v \in M_i$. In other words, $M = \bigcup_i M_i$. This suggests the following algorithm for enumerating the minimal hitting sets of type 2. For each $i \in \{1, \ldots, \lambda\}$, recursively compute the set \mathcal{X}_i of all minimal hitting sets of $(V \setminus V_i, \mathcal{R}_i)$. Then, try all possible ways of picking one minimal hitting set $M_i \in \mathcal{X}_i$ from each \mathcal{X}_i and output $M = \bigcup_i M_i$ if it is a type 2 minimal hitting set for \mathcal{R} . This way we surely enumerate all type 2 minimal hitting sets. Now, we need to bound the number of combinations we try. We do it by proving an upper bound on each $|\mathcal{X}_i|$.

Lemma 4 Let T be the number of minimal hitting sets of \mathcal{R} . Then, $|\mathcal{X}_i| \leq T$.

Proof. We show that each $N \in \mathcal{X}_i$ can be extended to $N' = N \cup S$ for some $S \subseteq V_i$ so that N' is a minimal hitting set of \mathcal{R} . The lemma then follows since each distinct $N \in \mathcal{X}_i$ is extended to a distinct minimal hitting set N' of \mathcal{R} . Given any $N \in \mathcal{X}_i$, let $\overline{\mathcal{R}}$ be the set of ranges in \mathcal{R} that are not hit by N. Clearly, each range in $\overline{\mathcal{R}}$ has a non-empty intersection with V_i (otherwise it would be in \mathcal{R}_i and thus would be hit by N). Therefore, there exists a set $S \subseteq V_i$ which is a minimal hitting set for $\overline{\mathcal{R}}$. Now, $N' = N \cup S$ is certainly a hitting set of \mathcal{R} . Furthermore, it is also minimal since each element of N' has a certificate range. The certificate ranges for each element v in N are also certificate ranges for v in N' since these ranges belong to \mathcal{R}_i and hence do not contain any elements of S. Also, the certificate ranges for each v in S are certificate ranges for v in N', since these ranges belong to $\bar{\mathcal{R}}$ and hence do not intersect N. \square

It follows from the above lemma that the number of combinations of M_i 's we need to try is at most T^{λ} . After we find all type 1 minimal hitting sets we run a procedure called Remove-Duplicates to remove any duplicates we may have generated. Similarly, after we find all type 2 minimal hitting sets, we run Remove-Duplicates to

remove any duplicates. This ensures that in the end we do not output any duplicates.

We now do an analysis of the running time of the algorithm. In the analysis, we treat the number of ranges $m = |\mathcal{R}|$ and the number T of the number of minimal hitting sets of \mathcal{R} as constants. We denote by t(n), the running time of the procedure Enumerate on a hypergraph (V, \mathcal{R}) where |V| = n. The recursive calls in Line 7 of Algorithm 1 for enumerating the type 1 minimal hitting sets take time $\lambda t((1-\epsilon)n)$. Similarly, the total time spent in Line 12 is $\lambda t((1-\epsilon)n)$. The loop starting on Line 14 is executed at most T^{λ} times. In each iteration, checking whether M is a type 2 minimal hitting set of \mathcal{R} takes O(mn) time. Hence the total time spent in the loop is $O(mnT^{\lambda})$. Since there are at most T distinct minimal hitting sets of \mathcal{R} , when we reach Line 9, Type1-Set has at most λT minimal hitting sets. Each of these have to be tested against a set of at most T distinct minimal hitting sets to see if it has already been reported. Therefore, this takes $O(\lambda T^2 n)$ time assuming that it takes O(n) to check if two minimal hitting sets are the same. Similarly, when we reach Line 20, the size of Type2-Set is at most T^{λ} and each of the hitting sets in it is compared against a set of at most T minimal hitting sets to see if it has been reported before. This takes $O(T^{\lambda+1}n)$ time. We therefore have the following recursion:

$$t(n) \le 2\lambda t((1-\epsilon)n) + \lambda nT^2 + mnT^{\lambda} + nT^{\lambda+1} + \tau,$$

where τ is the time required to find a balanced subdivision. Using the fact that t(n) is a constant when n is smaller than some constant μ , we see that $t(n) = O((\tau + nT^{\lambda+1} + nmT^{\lambda}) \cdot n^{\frac{\log \lambda}{\log 1/(1-\epsilon)}})$. We therefore have the following theorem.

Theorem 5 Procedure Enumerate(V, \mathcal{R}) finds all minimal hitting sets of a range space (V, \mathcal{R}) which admits a balanced subdivision V_1, \ldots, V_{λ} with each $|V_i| \ge \epsilon |V|$, whenever |V| is larger than a fixed constant μ , in time $O((\tau + nT^{\lambda+1} + nmT^{\lambda}) \cdot n^{\frac{\log \lambda}{\log 1/(1-\epsilon)}})$, where n = |V|, $m = |\mathcal{R}|$, T is the number of minimal hitting sets of \mathcal{R} and τ is the time required to compute a balanced subdivision.

Remark 3 The way the above algorithm is described gives an output polynomial algorithm for generating $\operatorname{Tr}(\mathcal{R})$. Using techniques from [20], we can modify the algorithm to become incremental polynomial, that is, for every $M' \leq M$ the algorithm outputs M' transversals in time polynomial in n, m and M'.

Parallel Implementation of the Algorithm: Algorithm 1 can be parallelized in an obvious way. Each of the For loops can be executed in parallel, i.e., all the iterations are done in parallel. Using poly(n, m, T)

processors, each of the other steps can be executed in $\operatorname{polylog}(n, m, T)$ time. If we denote by $t^{\parallel}(n)$ the running time of such a parallel algorithm, again treating m and T as constants, we get the following recurrence: $t^{\parallel}(n) = t^{\parallel}((1-\epsilon)n) + \operatorname{polylog}(n, m, T)$. We therefore have that $t^{\parallel}(n)$ is in $\operatorname{polylog}(n, m, T)$. It can be checked that the total number of processors required is only $\operatorname{poly}(n, m, T)$.

Using the techniques in [20], we can also get an incremental version of this, i.e., for any $M' \leq M$, the running time depends polylogarithmically on M', provided that there is an efficient parallel algorithm for finding a single minimal transversal of the input hypergraph \mathcal{R} . The existence of the latter algorithm for general hypergraphs, and in particular for range spaces, is an outstanding open question (see e.g. [19]). The currently best known parallel implementation for the later problem is due to Karp, Upfal, and Wigderson [19] who gave a randomized algorithm which makes only $O(\sqrt{n})$ parallel oracle calls on $O(n^{3/2})$ processors to compute a maximal independent set (complement of a minimal transversal, in the case of explicitly given hypegraphs) of an independence system given by an oracle on n vertices.

The running time of the algorithm described above is super-linear in the output size and hence not ideal for some applications. Similarly, previous algorithms for generating minimal hitting sets of half-planes [14] suffer from the same short-coming. In some simple cases, however, there exist algorithms which produce output with polynomial delay i.e. the time spent between enumerating two minimal hitting sets is polynomial in the size of the input and does not depend on the size of the output. Such algorithms clearly have a running time linear in the size of the output. We state the following result without proof.

Theorem 6 Let P be a set of n points and \mathcal{R} be a set of m half-planes in \mathbb{R}^2 . Then all minimal hitting sets of the range spaces (P, \mathcal{R}) and (\mathcal{R}, P) can be generated in $poly(n, m) \cdot k$ time where k is the size of the output.

References

- C. Berge. Hypergraphs. Elsevier-North Holand, Amsterdam, 1989.
- [2] J. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive boolean functions. *Information and Computation*, 123(1):50–63, 1995.
- [3] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. Generating maximal independent sets for hypergraphs with bounded edgeintersections. In *LATIN* '04, pages 488–498, 2004.

- [4] E. Boros and K. Makino. A fast and simple parallel algorithm for the monotone duality problem. In *ICALP '09, to appear*, 2009.
- [5] Bernard Chazelle. Cutting hyperplanes for divideand-conquer. Discrete & Computational Geometry, 9:145–158, 1993.
- [6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry, Algorithms and Applications. SpringerVerlag, Amsterdam, 1997.
- [7] C. Domingo, N. Mishra, and L. Pitt. Efficient readrestricted monotone cnf/dnf dualization by learning with membership queries. *Machine Learning*, 37(1):89–110, 1999.
- [8] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Computing*, 24(6):1278–1304, 1995.
- [9] T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Computing*, 32(2):514–537, 2003.
- [10] T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035– 2049, 2008.
- [11] Thomas Eiter. Exact transversal hypergraphs and application to Boolean μ-functions. Journal of Symbolic Computation, 17(3):215–225, 1994.
- [12] K. Elbassioni. On the complexity of the multiplication method for monotone CNF/DNF dualization. In ESA '06, pages 340–351, 2006.
- [13] K. Elbassioni. On the complexity of monotone dualization and generating minimal hypergraph transversals. *Discrete Applied Mathematics*, 156(11):2109–2123, 2008.
- [14] K. Elbassioni, K. Makino, and I. Rauf. Outputsensitive algorithms for enumerating minimal transversals for some geometric hypergraphs. In *ESA*, 2009.
- [15] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.
- [16] D. R. Gaur and R. Krishnamurti. Average case selfduality of monotone boolean functions. In *Canadian AI '04*, pages 322–338, 2004.

- [17] G. Gottlob. Hypergraph transversals. In FoIKS '04: Proc. of the 3rd International Symposium on Foundations of Information and Knowledge Systems, pages 1–5, 2004.
- [18] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *Comput. J.*, 51(1):7–25, 2008.
- [19] R. M. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *Journal of Computer* and System Sciences, 36(2):225–253, 1988.
- [20] L. Khachiyan, E. Boros, K. Elbassioni, and V. Gurvich. A global parallel algorithm for the hypergraph transversal problem. *Information Processing Letters*, 101(4):148–155, 2007.
- [21] L. Khachiyan, E. Boros, V. Gurvich, and K. Elbassioni. Computing many maximal independent sets for hypergraphs in parallel. *Parallel Process*ing Letters, 17(2):141–152, 2007.
- [22] L. Lovász. Combinatorial optimization: some problems and trends. DIMACS Technical Report 92-53, Rutgers University, 1992.
- [23] Jirí Matousek. Efficient partition trees. Discrete & Computational Geometry, 8:315–334, 1992.
- [24] C. Papadimitriou. NP-completeness: A retrospective. In *ICALP* '97, 1997.
- [25] H. Tamaki. Space-efficient enumeration of minimal transversals of a hypergraph. Technical Report IPSJ-AL 75, 2000.

Helly Numbers of Polyominoes

Jean Cardinal *

Hiro Ito^{\dagger}

Matias Korman*

Stefan Langerman*

Abstract

We define the Helly number of a polyomino P as the smallest number h such that the h-Helly property holds for the family of symmetric and translated copies of P on the integer grid. We prove the following: (i) the only polyominoes with Helly number 2 are the rectangles, (ii) there does not exist any polyomino with Helly number 3, (iii) there exist polyminoes of Helly number k for any $k \neq 1, 3$.

1 Introduction

Helly's theorem on convex sets is a cornerstone of discrete geometry, with countless corollaries and extensions in both geometry and combinatorics. For instance, Helly-type properties of convex lattice subsets and hypergraphs have been studied since the 70's [3]. On the other hand, the theory of polyominoes, connected subsets of the square lattice \mathbb{Z}^2 , has been developed since the 50's with the seminal works of Solomon Golomb [5] and the famous recreational mathematician Martin Gardner.

In this paper, we propose a natural definition of the Helly number of a polyomino P by considering families of symmetric and translated copies of P. We show that the only polyominoes with Helly number 2 are rectangles. We prove the surprising fact that there does not exist any polyomino with Helly number 3. Finally, we exhibit polyominoes of Helly number k for any $k \ge 4$. Since there cannot be polyominoes of Helly number 1, this completely characterizes the values of k for which there exist polyominoes with Helly number k.

Definitions

We define a planar graph $G = (\mathbb{Z}^2, E)$ that represents the adjacency relation between grid points. Each vertex (i, j) is connected to its four neighbors (i, j-1), (i-1, j),(i+1, j), and (i, j+1). A subset of \mathbb{Z}^2 is *connected* if its induced subgraph in G is connected.

Definition 1 A polyomino is a connected finite subset of \mathbb{Z}^2 .



Figure 1: Eight possible symmetries of a polyomino.

We often identify the point $(i, j) \in \mathbb{Z}^2$ with the unit square $[i, i + 1] \times [j, j + 1] \subset \mathbb{R}^2$. With this transformation a polyomino becomes an orthogonal polygon whose edges are on the unit grid. A *copy* of a polyomino P is the image of P by the composition of an integer translation with one of the eight symmetries of the square (that is, a mirror image and/or a 90, 180, or 270-degree rotation of P). Figure 1 shows an example of a polyomino and its eight symmetries. The cardinality of a polyomino will be denoted by |P| (and will be referred as the *size* of P).

Definition 2 For any $k \in \mathbb{N}$ a polyomino P is called k-Helly [7] if, for any finite family \mathcal{A} of copies of P in which $A_1 \cap \ldots \cap A_k \neq \emptyset$ for any $A_1, \ldots, A_k \in \mathcal{A}$, we have $\bigcap_{A \in \mathcal{A}} A \neq \emptyset$. The Helly number $\mathcal{H}(P)$ of a polyomino P is the smallest $k \in \mathbb{N}$ such that P is k-Helly.

By definition, any polyomino P that is k-Helly will also be k'-Helly for any $k' \ge k$.

Previous work

A convex lattice set in \mathbb{Z}^d is the intersection of a convex set in \mathbb{R}^d with the integer grid \mathbb{Z}^d . In 1973, Doignon proved that any family of convex lattice sets in \mathbb{Z}^d is 2^d -Helly [3]. A matching lower bound is obtained by considering all subsets of size $2^d - 1$ of $\{0, 1\}^d$. In our context, this implies that any convex polyomino (i.e. a polyomino that is the intersection a convex set in \mathbb{R}^2 with \mathbb{Z}^2) is 4-Helly. Note that this is different from the term *convex polyomino*, which usually refers to polyominoes that are simultaneously row and column convex.

^{*}Computer Science Department, Université Libre de Bruxelles (ULB), Belgium, {jcardin, mkormanc, stefan.langerman}@ulb.ac.be

[†]School of Informatics, Kyoto University, Japan, itohiro@lab2.kuis.kyoto-u.ac.jp

Fractional Helly numbers of convex lattice subsets are studied by Bárány and Matousek [1]. Recently, Golumbic, Lipshteyn, and Stern showed that 1-bend paths on a grid have Helly number 3 [6]. We note the environment considered is slightly different, since they considered that two paths have nonempty intersection whenever they share an edge.

2 Helly Number up to 4

In this Section we study polyominoes of small Helly number. Since we are considering finite polyominoes, it is easy to see that no polyomino can have Helly number 1. Thus, we first look for polyominoes with Helly number two.

Definition 3 A rectangle in \mathbb{Z}^2 is the cartesian product of two intervals in \mathbb{Z} . The bounding box of a polyomino P is the smallest rectangle in \mathbb{Z}^2 that contains P.

It is easy to see that rectangles have Helly number 2. We show that the converse also holds.

Theorem 1 A polyomino has Helly number 2 if and only if it is a rectangle.

In the following we give a slightly stronger result; we will show that the only polyominoes that satisfy the 3-Helly property are rectangles.

Definition 4 A polyomino P has the small empty quadrant structure if for some copy P' of P, there exist values $x_1, y_1 \in \mathbb{Z}$ such that the intersection of P' with the 2×2 rectangle $[x_1, x_1+1] \times [y_1-1, y_1]$ has cardinality ≥ 3 , and P' contains no point in $\{(x, y) : x \geq x_1, y > y_1\}$ (see Figure 2 (a)).

Definition 5 A polyomino P has the big empty quadrant structure if for some copy P' of P, there exist values $x_1, y_1, x_2, y_2 \in \mathbb{Z}$, $y_1 < y_2$, $x_1 < x_2$ such that $\{(x_1, y_2), (x_1, y_1), (x_2, y_1)\} \subset P'$ and P' contains no point in the upper right quadrant $\{(x, y) : x > x_1, y > y_1\}$ (see Figure 2 (b)).

Given a rectangle $[x_0, x_1] \times [y_0, y_1]$, its *height* is $y_1 - y_0 + 1$. Analogously, its *width* is $x_1 - x_0 + 1$. The height and width of a polyomino P are equal to the height and width of the smallest enclosing rectangle of P, respectively.

Lemma 2 Every polyomino P whose height and width are 2 or more either has the small empty quadrant or the big empty quadrant structure.

Proof. Observe that if P has either height or width exactly 1 it must be a rectangle. Hence, in particular, this Lemma shows that any polyomino (other than some



Figure 2: Illustration of Lemma 2. In order for a polyomino P (of height at least 2) to not have the small empty quadrant structure (case (a)), P cannot have two consecutive points on its upper boundary. If this occurs, we can find a large empty quadrant (case (b)). The coordinates x_1, x_2, y_1, y_2 that generate the big or small empty quadrant are shown in black.

rectangles), has one of the two structures. Let (x_0, y_0) be the point of P highest x-coordinate along the upper boundary of its bounding box.

We first show that if $(x_0, y_0 - 1) \notin P$, then there exists $i \in \mathbb{N}$ such that $(x_0 - i + 1, y_0), (x_0 - i, y_0), (x_0 - i, y_0 - 1) \in P$. Proof is as follows: by definition of (x_0, y_0) , we have that $(x_0 + 1, y_0) \notin P$, and $(x_0, y_0 + 1) \notin P$. If we suppose that $(x_0, y_0 - 1) \notin P$, then, in order for P to be connected, we must have $(x_0 - 1, y_0) \in P$. By applying the same argument iteratively on this new point, we must have that eventually there exists an i such that both $(x_0 - i - 1, y_0) \in P$ and $(x_0 - i - 1, y_0 - 1) \in P$, otherwise P is a rectangle of height 1.

Therefore, if $(x_0, y_0 - 1) \notin P$, P has the small empty quadrant structure. Now assume otherwise and let jbe the smallest integer such that $(x_0, y_0 - j) \in P$ and $(x_0, y_0 - j - 1) \notin P$. If the quadrant $\{(x, y) : x > x_0, y \ge y_0 - j\}$ contains no point of P, then, by the same argument as in the above claim, there must be a point of P immediately left of the column x_0 between y_0 and $y_0 - j$. In other words, there must be an integer $j' \in [0, j - 1]$ such that $|P \cap ([x_0 - 1, x_0] \times [y_0 - j' - 1, y_0 - j'])| \ge 3$, and again P has the small empty quadrant structure.

Finally, if the quadrant $\{(x, y) : x > x_0, y \ge y_0 - j\}$ is not empty, let (x', y') be the highest point in that quadrant (pick one arbitrarily if many exist). In that case, the three points $(x_0, y_0), (x_0, y'), (x', y')$ form a big empty quadrant structure.

Lemma 3 If a polyomino P has the big empty quadrant structure, then $\mathcal{H}(P) \geq 4$.

Proof. We construct an arrangement of four copies of P such that every subset of three copies have a common

point, but there is no point common to all four copies. We denote these copies by P_i , with $i = 1, \ldots, 4$.

Consider the three points (x_1, y_2) , (x_1, y_1) , and (x_2, y_1) given by the big empty quadrant structure in P. We construct the copies P_i by flipping Paround the x and/or y axis so that those three points map to all possible triples of points in the set $\{(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)\}$. Since $(x_2, y_2) \notin P$, each of the four points is missing from exactly one copy P_i , but belongs to the other three.

Now we observe that the empty quadrants of the four copies P_i cover \mathbb{Z}^2 . Hence for any $(x, y) \in \mathbb{Z}^2$, there exists at least one $i \in \{1, 2, 3, 4\}$ such that $(x, y) \notin P_i$. Therefore, the four copies have no common intersection point.

We now consider polyominoes that have the small empty quadrant structure. We will use the following observation.

Observation 1 For any polyomino P that is not a rectangle, there exists a 2×2 rectangle R such that $|P \cap R| = 3$.

Lemma 4 If a polyomino P has the small empty quadrant structure and is not a rectangle, then $\mathcal{H}(P) \geq 4$.

Proof. We construct an arrangement of at most 8 copies of P such that every subset of three copies have a common point, but there is no point common to all copies. Let (x_1, y_1) be the point given by the small empty quadrant structure, and P' the corresponding copy of P.

We first consider the case in which the intersection Lof P' with the 2 × 2 rectangle $[x_1, x_1 + 1] \times [y_1 - 1, y_1]$ has cardinality exactly 3. In that case, we can use a similar construction as in Lemma 3, with four copies of P; we define the copies P_i for i = 1, 2, 3, 4 as the four rotations of P that map the bounding box of Lto the same 2 × 2 rectangle. Those four points are the respective intersection points of all four possible triples. Similar to the previous case, the four empty quadrants cover all the other points of \mathbb{Z}^2 , hence there cannot be a common intersection point.

It remains to consider the case in which the intersection L has size 4. In this situation we use the same construction, but complete it with four more copies. From Observation 1 and the fact that P is not a rectangle, we know that there exists a 2×2 rectangle R such that $|P' \cap R| = 3$. We add four additional copies P_i , with i = 5, 6, 7, 8, that are the four rotations of a translated copy of P' mapping R to the bounding box of L. Each of the four points of this rectangle belongs to copies P_1, P_2, P_3, P_4 (since |L| = 4), and to exactly three of the four copies P_5, P_6, P_7, P_8 (since $|P' \cap R| = 3$). Hence every triple of copies intersects. However, from the previous construction, there still exists no point common to all 8 copies. This construction does not work for rectangles, since Observation 1 does not hold in that case. $\hfill \Box$

Corollary 5 There is no polyomino of Helly number 3.

Combining this result with the upper bound of [3], we can compute the Helly number of any convex polyomino:

Corollary 6 Let P be a polyomino that is the intersection a convex set in \mathbb{R}^2 with \mathbb{Z}^2 . If P is a rectangle then $\mathcal{H}(P) = 2$. Otherwise $\mathcal{H}(P) = 4$.

3 Hypergraph Generalization

In this section we study some interesting properties of polyominoes of Helly number k. Since these results hold for subsets of a discrete set of points, we state these results in a more general fashion. Instead of copies of a given polyomino we can consider the same definitions for families of subsets of \mathbb{Z}^2 . Using this idea, one can extend the Helly property to hypergraphs.

Definition 6 A hypergraph $G = (V, \mathcal{E})$ is k-Helly if for any $W \subseteq \mathcal{E}$ such that $e_1 \cap \ldots \cap e_k \neq \emptyset$ for all $e_1, \ldots, e_k \in \mathcal{W}$, we have $\bigcap_{e \in \mathcal{W}} e \neq \emptyset$. The Helly number $\mathcal{H}(G)$ of a hypergraph G is the smallest value k such that G is k-Helly.

Helly numbers of hypergraphs have been deeply studied (see for example the survey of Dourado, Protti, and Szwarcfiter [4]). Observe that the above definition is a generalization of the previous definition for the polyomino case. Indeed, the polyomino formulation is the particular case in which $V = \mathbb{Z}^2$ and \mathcal{E} contains all subsets of points contained in copies of a fixed polyomino P.

Let G be a hypergraph that is not k-Helly. By definition, there exists a subset $W \subseteq \mathcal{E}$ such that $\bigcap_{e \in W} e = \emptyset$ and $e_1 \cap \ldots \cap e_k \neq \emptyset$ for any $e_1, \ldots, e_k \in W$. Any such family is called a a k-witness set of G. For every $V' \subset V$, define the restriction of G to V' as $G|_{V'} = (V', \mathcal{E}|_{V'})$, where $\mathcal{E}|_{V'} = \{e \cap V' | e \in \mathcal{E}\}$. With these definitions we can prove an upper bound on the Helly number of any hypergraph:

Theorem 7 Let $G = (V, \mathcal{E})$ be a hypergraph. If $|e| \le k$ $\forall e \in \mathcal{E}$, then G is (k + 1)-Helly.

Proof. We will show the result by induction on k. Observe that the claim for k = 0 is trivial, hence we focus on the induction step. Assume otherwise: let $\mathcal{W} \subseteq \mathcal{E}$ be a (k + 1)-witness set, and e be an edge of maximum size among those of \mathcal{W} (by hypothesis we know that $|e| \leq k$).

Consider the hypergraph $G' = (e, \mathcal{W}|_e \setminus \{e\})$ (that is, we disregard all other vertices except those contained in e). Since $|e| \leq k$, its intersection with any other edge of \mathcal{W} must be of size at most k-1. Furthermore, every k-tuple of edges in G' have a common intersection (since every k+1 tuple in \mathcal{W} including e had a common intersection). Therefore, by induction G' is k-Helly. In particular all edges in G' have a common intersection, which by construction intersects e and contradicts the witness property. \Box

Corollary 8 Any polynomino P satisfies $\mathcal{H}(P) \leq |P| + 1$.

The proof is direct from the fact that the associated hypergraph is |P|-uniform. We also note that the bound of Corollary 8 is tight: the polyomino $\{(0,0), (1,0), (0,1)\}$ (commonly referred as El[2]) has cardinality 3 and contains the small empty quadrant structure. In particular, by Lemma 4 its Helly number must be at least 4.

In the following we give a few more tools to use when proving that a given hypergraph is k-Helly (or equivalently, that there cannot exist a k-witness).

Lemma 9 Any k-witness W of a hypergraph G satisfies $|W| \ge k + 1$ and $|e_1 \cap \ldots \cap e_\ell| \ge k - \ell + 1$ for all $e_1, \ldots, e_\ell \in W$.

Proof. Observe that the first claim is trivial, since if \mathcal{W} has size k or less it cannot have an empty intersection. The proof of the second claim is by contradiction: assume otherwise and let $e_1, \ldots, e_\ell \in \mathcal{W}$ such that such that $e_1 \cap \ldots \cap e_\ell = \{v_1, \ldots, v_m\}$ for some $m \leq k - \ell$. Since $\bigcap_{e \in \mathcal{W}} e = \emptyset$, for any $i \leq k - \ell$ there exists $f_i \in \mathcal{W}$ such that $v_i \notin f_i$.

Consider now the intersection of $e_1 \cap \ldots \cap e_{\ell} \cap f_1 \cap \ldots \cap f_m$: by construction, this set is empty. Moreover, the size of the set $\{e_1, \ldots, e_{\ell}, f_1, \ldots, f_m\}$ is at most $\ell + m \leq \ell + k - \ell = k$, which contradicts the witness property of \mathcal{W} .

For any hypergraph G and vertex $v \in V$, we define $c_v = \{e \in \mathcal{W}, v \in e\}$ as the edges that contain v. In the following we show that we can ignore vertices that are not heavily covered.

Lemma 10 Let \mathcal{W} be a k-witness set of G and let $V' = \{v \in V, |c_v| \ge k\}$. The set $\mathcal{W}|_{V'}$ is a k-witness for $G|_{V'}$.

Proof. Observe that $\bigcap_{e \in \mathcal{W}} e = \emptyset \Rightarrow \bigcap_{e \in \mathcal{W}|_{V'}} e = \emptyset$. Hence, it suffices to show that $e_1 \cap \ldots \cap e_k \cap V' \neq \emptyset$, for any $e_1, \ldots, e_k \in \mathcal{W}$,

Let $S = e_1 \cap \ldots \cap e_k$. Observe that, since \mathcal{W} is a witness set, we have $S \neq \emptyset$. Moreover all points of S are covered by at least k hyperedges (since they are contained in e_1, \ldots, e_k). Hence we have $S \subseteq V'$. In particular, we obtain $e_1 \cap \ldots \cap e_k = e_1 \cap \ldots \cap e_k \cap V' \neq \emptyset$ which proves the Lemma. \Box



Figure 4: Polyominoes A_0 (solid blue) and B_2 (dashed in red). In the example q = 8.

Lemma 9 gives a lower bound on the size of a witness set. We use a similar reasoning to find an upper bound as well:

Lemma 11 Let G be any hypergraph such that $\mathcal{H}(G) = k$. There exists a (k-1)-witness set $\mathcal{W} \subseteq \mathcal{E}$ of P such that $|\mathcal{W}| = k$.

Proof. Let \mathcal{W}_{\min} be the (k-1)-witness set of smallest size (pick any arbitrarily if many exist) and let $m = |\mathcal{W}_{\min}|$. By Lemma 9 we have $m \ge k$. If m = k we are done, thus we focus in the m > k case.

By minimality of \mathcal{W}_{\min} , there cannot exist a proper subset $\mathcal{W}' \subset \mathcal{W}_{\min}$ such that $\cap_{A \in \mathcal{W}'} A = \emptyset$ (otherwise we would have a witness set of smaller size). In particular, any subset $\{e_1, \ldots, e_k\} \subset \mathcal{W}_{\min}$ must have non-empty intersection. Since G is k-Helly, we have $\cap_{e \in \mathcal{W}_{\min}} e \neq \emptyset$ which contradicts the witness property. \Box

4 Higher Helly Numbers

In the following we use the above tools to show the existence of polyominoes of Helly number k (for any $k \geq 5$). For any $q \in \mathbb{N}$, we define polyomino F_q is defined as the union of rectangles $[-\lfloor q/2 \rfloor, -1] \times [0, 0], [1, q] \times [0, 0]$ and $[-1, 1] \times [1, 1]$. Observe that $|F_q| = \lfloor 3q/2 \rfloor + 3$, see Figure 3.

Lemma 12 For any $q \ge 4$, we have $\mathcal{H}(F_q) = q + 1$.

Proof. We show the lower bound by constructing a q-witness set \mathcal{W} of F_q . For any $i \leq q$, we define A_i as the copy of F_q translated such that the leftmost point is at position (i, 0). Analogously, we define polyomino B_i as the 180-degree rotation of F_q translated so as the leftmost point is at position (i, 0)



Figure 5: q-Witness set for polyomino F_q (for clarity, each of the copies has been shifted vertically). Observe that, although the intersection of the witness set is empty, any q elements of the set have nonempty intersection. In the figure, we depicted with a vertical strip the point that is contained in all polyominoes except $A_{\lceil q/2 \rceil} - 1$.

(see Figure 4). We define the witness set as $\mathcal{W} = \{A_0, \ldots, A_{\lceil q/2 \rceil - 1}, B_0, B_0, \ldots, B_{\lfloor q/2 \rfloor}\}$. Observe that $|\mathcal{W}| = \lceil q/2 \rceil + \lfloor q/2 \rfloor + 1 = q + 1$ and that the intersection between polyminoes A_i and B_j is in the rectangle $[0, |3q/2|] \times [0, 0]$ (for any i and j).

More interestingly, for any $0 \leq i \leq \lceil q/2 \rceil - 1$, polyomino A_i does not contain point $(\lfloor q/2 \rfloor + i, 0)$ (and this point is contained in all other polyominoes). The same result holds for polyomino B_i : for any $0 \leq i \leq \lfloor q/2 \rfloor$, point (q+i, 0) is contained in all polyominoes except B_i . In particular, we have $\cap_{C \in \mathcal{W}} C = \emptyset$ and any subset of size q has nonempty intersection (see Figure 5). Hence, \mathcal{W} is a q-witness set of F_q .

In order to finish the proof of the Lemma, we must show that polyomino F_q indeed is (q+1)-Helly. Assume that F_q is not (q+1)-Helly. Let \mathcal{W} be a (q+1)-witness set and let A be the leftmost copy of F_q in \mathcal{W} (pick any arbitrarily if more than one exist). Without loss of generality, we can assume that $A = A_0$. By Lemma 9, there must exist at least q+1 other copies A of F_q such that $|A \cap A_0| \geq q$.

First notice that if any two copies of the polyomino do not align their longest segment horizontally, they only have an intersection of size at most 4 with A_0 . Moreover, the only case when this intersection has size 4 is if they are two copies flipped across the horizontal axis. In the latter case, any further copy can have an intersection of size at most 3 with at least one of those two copies. Since in either case we obtain a contradiction with Lemma 9 and the fact that $q \ge 4$, we can assume that for any q+1-witness set, all copies of \mathcal{W} are aligned horizontally.

Consider now the 3 lower points $(\lfloor q/2 \rfloor - 1, -1), (\lfloor q/2 \rfloor, -1)$ and $(\lfloor q/2 \rfloor + 1, -1)$ of A_0 . Since A_0 is the leftmost copy of P and $q \ge 4$ and copies are aligned horizontally, the three points can only be covered by at most two other copies $(A_1 \text{ and } A_2)$. Therefore we apply Lemma 10 to show that any (q + 1)-witness set of \mathbb{Z}^2 would be a witness set of $\mathbb{Z}^2 \setminus \{(\lfloor q/2 \rfloor - 1, -1), (\lfloor q/2 \rfloor, -1), (\lfloor q/2 \rfloor + 1, -1)\}$. Thus, we focus our attention in the rectangle $[0, \lfloor 3q/2 \rfloor] \times [0, 0]$.

Observe that, since we are considering only this rectangle, the extra copies caused by reflections across the horizontal axis are eliminated because they become the same hyperedge in the restricted hypergraph. Hence, all elements of \mathcal{W} must be of the form A_i or B_j for some $i, j \geq 0$. Also notice that we have $|A_0 \cap A_i| \geq q$ if and only if $i \in \{1, \ldots, \lfloor q/2 \rfloor - 1\}$ (provided that $q \geq 4$). Analogously, if $q \geq 2$ we have $|A_0 \cap B_j| \geq q \Leftrightarrow j \in$ $\{0, \ldots, \lfloor q/2 \rfloor - 1\}$. In particular, the set \mathcal{W} can have at most $2\lfloor q/2 \rfloor$ elements, hence there cannot exist a (q+1)witness set.

Theorem 13 For any $k \in \mathbb{N}$ such that $k \neq 1, 3$, there exists a polyomino P such that $\mathcal{H}(P) = k$.

5 Conclusion

In this paper we have completely characterized for which values of k there exist polyominoes of Helly number k. An interesting problem is to find a method to compute the Helly number of a given polyomino. Using the results of Section 3, it is not hard to devise an algorithm that runs in exponential time, testing all possible witness sets. Although finding an algorithm that works for general hypergraphs is difficult [4], we wonder whether one can devise an algorithm that runs in polynomial time for any given polyomino P.

Finally note that we defined a copy of P as any image of P with respect to translations and the 8 symmetries of the square. Our results do not hold if we only consider translations (or rotations and translations). Hence, it would be interesting to see how much can the Helly number of a given polyomino change when allowing or forbidding these operations.

References

- I. Bárány and J. Matousek. A fractional Helly theorem for convex lattice sets. Advances in Mathematics, 174(2):227 – 235, 2003.
- [2] J. Beck. Combinatorial games: Tic-Tac-Toe theory. Encyclopedia of mathematics and its applications. Cambridge Univ. Press, Cambridge [u.a.], 2008.
- [3] J.-P. Doignon. Convexity in crystallographic lattices. Journal of Geometry, 3:71–85, 1973.
- [4] M. C. Dourado, F. Protti, and J. L. Szwarcfiter. Computational aspects of the helly property: a survey. *Journal* of the Brazilian Computer Society, 12(1):7–33, 2006.
- [5] S. W. Golomb. Polyominoes: Puzzles, Patterns, Problems, and Packings. Princeton University Press, 2nd edition, 1996.
- [6] M. C. Golumbic, M. Lipshteyn, and M. Stern. Edge intersection graphs of single bend paths on a grid. *Net*works, 54(3):130–138, 2009.
- [7] J. E. Goodman and J. O'Rourke, editors. Handbook of discrete and computational geometry. CRC Press, Inc., Boca Raton, FL, USA, 2004.

Open Guard Edges and Edge Guards in Simple Polygons

Csaba D. Tóth*

Godfried Toussaint[†]

Andrew Winslow[‡]

Abstract

An open edge of a simple polygon is the set of points in the relative interior of an edge. We revisit several art gallery problems, previously considered for closed edge guards, using open edge guards. A guard edge of a polygon is an edge that sees every point inside the polygon. We show that every simple non-starshaped polygon admits at most one open guard edge, and give a simple new proof that it admits at most three closed guard edges. We characterize open guard edges, and derive an algorithm that finds all open guard edges of a simple n-gon in O(n) time in the RAM model of computation. Finally, we present lower bound constructions for simple polygons with n vertices that require $\lfloor n/3 \rfloor$ open edge guards, and conjecture that this bound is tight.

1 Introduction

Let P be a simply connected closed polygonal domain with n vertices. Two points $p, q \in P$ are mutually visible to each other if the closed line segment pq lies in P. In a starshaped polygon P, all points in P are visible from a single point $x \in P$, which is called a *guard point* for P. The set of all guard points is the *kernel* of P.

For a set $S \subseteq P$ of multiple guards, or the trajectories of mobile guards, we adopt the notion of weak visibility [2]. A point $p \in P$ is (weakly) visible to a set $S \subseteq P$ if it is visible from some point in S. If every point $p \in P$ is (weakly) visible from S, then S is a guard set.

Park et al. [8] considered guard sets restricted to (closed) edges of a polygon. They proved that a nonstarshaped simple polygon has at most three closed guard edges, and this bound is tight. They also designed an O(n) time algorithm for finding all closed guard edges in a simple *n*-gon. Later, it was shown that a shortest guard segment along the boundary of P, or anywhere in P can also be found in optimal O(n)time [3, 4]. A watchman tour is a closed curve $\gamma \subset P$ which is a guard set for P. Tan [11] gave an $O(n^5)$ time algorithm for finding a shortest watchman tour. If several guards are available, we are interested in the minimum number of guards that can jointly cover any simple polygon with n vertices. By a classical result of Klee, a set of $\lfloor n/3 \rfloor$ vertex guards are always sufficient and sometimes necessary to cover a simple n-gon. It is known that $\lfloor n/4 \rfloor$ closed edge guards are sometimes necessary, and $\lfloor 3n/10 \rfloor + 1$ are always sufficient [9]. It is a longstanding conjecture that $\lfloor n/4 \rfloor + O(1)$ closed edge guards are always sufficient. However, $\lfloor n/4 \rfloor$ (open or closed) segment guards are always sufficient and sometimes necessary [7].



Figure 1: The region visible by an open edge uv (left) and a closed edge uv (right) in a simple polygon.

Viglietta [12] recently suggested the use of open edge guards for various scenarios. A *closed edge* includes the endpoints, and an *open edge* does not. See Fig. 1. Intuitively, a closed edge can "see around the corner" if its endpoint is a reflex vertex, while an open edge cannot. In this note, we examine two art gallery problems involving edges of polygons. First, *guard edges* of a polygon; single edges that guard the entire polygon. Then we consider *edge guards*; sets of edges that together guard the entire polygon.

2 Preliminaries

It is easy to express visibility in terms of shortest paths in a simple polygon (c.f., [1]). Given two points, p and q, inside a simple polygon P, the geodesic path(p,q) is the shortest directed path from p to q that lies entirely in P. Points p and q see each other iff path(p,q) is a straight line segment. Every interior vertex of path(p,q)is a reflex vertex of P. We characterize weak visibility between a point and an edge in terms of geodesics.

Lemma 1 Let p be a point inside a simple polygon P.

- (a) Point p is visible from an open edge uv iff p is the only common vertex of path(p, u) and path(p, v);
- (b) p is visible from a closed edge uv iff all common vertices of path(p, u) and path(p, v) are in {p, u, v}.

^{*}Department of Mathematics and Statistics, University of Calgary, Calgary, AB, Canada, cdtoth@math.ucalgary.ca

[†]Department of Music, Harvard University, Cambridge, MA, USA, Department of Computer Science, Tufts University, Medford, MA, USA, School of Computer Science, McGill University, Montreal, QC, Canada, godfried@cs.mcgill.ca

[‡]Department of Computer Science, Tufts University, Medford, MA, USA awinslow@cs.tufts.edu

Proof. (a) If p is the only common vertex of the two geodesics, then uv, path(p, u), and path(p, v) form a pseudo-triangle lying in P with corners p, u and v. Each corner of a pseudo-triangle is weakly visible from the opposite side, hence p is visible from a point in uv (Fig. 2, left). If $q \neq p$ is the last common vertex of the two geodesics, then q is an interior point of every geodesic from p to any $w \in uv$, hence p is not visible from any point of the open edge uv (Fig. 2, middle).



Figure 2: The geodesics path(p, u) and path(p, v). Left: p is the only common vertex of path(p, u) and path(p, v). Middle: the common vertices are p and q. Right: The common vertices are p and v.

(b) If p is the only common vertex of the two geodesics, then p is visible from an interior point of uv as in part (a). If u or v is the only common vertex (apart from p) of the two geodesics, then point p is directly visible from u or v (Fig. 2, right). Finally, if $q \notin \{p, u, v\}$ is a common vertex of the two geodesics, then q is an interior point of every geodesic from p to any $w \in uv$, and hence p is not visible from any point of the closed edge uv.

3 Open Guard Edges

In this section we consider open guard edges. Observe that every edge of a convex polygon is a guard edge, since it lies in the kernel of the polygon; but there may be up to n/4 open guard edges even if all edges are disjoint from the kernel (Fig. 3, left). In this section, we show that every non-starshaped simple polygon has at most one open guard edge. This bound is tight, as shown by the example in Fig. 3, right.



Figure 3: Left: a starshaped *n*-gon P with n/4 open guard edges where the kernel lies in the interior of P. Right: a non-starshaped polygon with one open guard edge.

We prove the upper bound by contradiction: we prove that a simple polygon with at least two open guard edges is starshaped. Let P be a simple polygon, and suppose that edges ab and cd are open guard edges. We may assume without loss of generality that a, b, c, d are in counterclockwise order along the boundary of P (possibly, b = c or d = a).

Lemma 2 path(b,c) and path(a,d) are disjoint. path(a,c) = ac and path(b,d) = bd are line segments.

Proof. Note that ab, path(b, c), cd, and path(a, d) form a geodesic quadrilateral Q. Every geodesic between a point in ab and a point in cd lies in Q. If path(b, c) and path(a, d) have a common interior vertex q, then a or bis not visible from the open edge cd by Lemma 1, and so cd cannot be a guard edge. We conclude that path(b, c)and path(a, d) are disjoint, and Q is a simple polygon.

The geodesics path(a, c) and path(b, d) lie in Q, so any interior vertex of path(a, c) and path(b, d) is a vertex of Q. If an interior vertex of path(a, c) is in path(b, c), then c is not visible from ab. Similarly, if an interior vertex of path(a, c) is in path(a, d), then a is not visible from cd. Hence, path(a, c) has no interior vertices. Analogous argument shows that path(b, d) has no interior vertices, either. \Box

Lemma 3 The intersection point $x = ac \cap bd$ is in the kernel of P.

Proof. Refer to Fig. 4. It is enough to show that an



Figure 4: A schematic of the proof that a simple polygon with two open guard edges must be starshaped. The guard edges are ab and cd. The point $x = ac \cap bd$ is in the kernel of the polygon, since every point $p \in P$ is visible from x.

arbitrary point p in polygon P is visible from x. By Lemma 2, ac and bd are diagonals of P. The triangles $\Delta(abx)$ and $\Delta(cdx)$ lie inside P. If $p \in \Delta(abx)$ or $p \in \Delta(cdx)$, then segment px lies in the same triangle. Assume now that p is outside of both triangles. Since aband cd are open guard edges, p sees some points in their relative interiors, say $o \in ab$ and $q \in cd$. The quadrilateral Q = (o, p, q, x) is simple, and so it lies inside P. Note that Q has convex vertices at o and q. No matter whether Q is a convex or a non-convex quadrilateral, its diagonal px lies inside Q, and hence inside P.

Theorem 4 Every non-starshaped simple polygon has at most one open guard edge. **Proof.** If a simple polygon has two open guard edges, then it has a nonempty kernel by Lemma 3, and thus is starshaped. So every non-starshaped simple polygon has at most one open guard edge. \Box

Remark. The upper bound of Theorem 4 does not apply to polygons with holes. Note that an open edge on the boundary of a hole cannot see the entire boundary of the hole. So all open edge guards are on the boundary of the outer polygon. By the result in [8] there are at most 3 *closed* guard edges on the outer boundary of a polygon with holes. Since every open guard edge is a closed guard edge, as well, a polygon with holes has at most 3 open guard edges. This upper bound is tight, as shown by the following simple construction. Let the outer polygon and a hole be two centrally dilated triangles. Then all three open edges of the outer polygon are guard edges.

4 Closed Guard Edges

In this section, we extend the argument of the previous section to give a short proof for the following result of Park *et al.* [8].

Theorem 5 ([8]) Every non-starshaped simple polygon has at most three closed guard edges.

We proceed by contradiction, and show that the presence of four closed guard edges implies that the polygon is starshaped. Let P be a simple polygon where $g_1, g_2,$ g_3 , and g_4 , in counterclockwise order, are guard edges. Let $g_1 = ab$ and $g_3 = cd$ such that a, b, c, and d are in counterclockwise order along P. Note that the vertices a, b, c, and d are distinct.

Lemma 6 The geodesics path(b, c) and path(a, d) are disjoint; and all vertices of the geodesics path(a, c) and path(b, d) are in $\{a, b, c, d\}$.

Proof. Consider the geodesic quadrilateral Q formed by ab, path(b, c), cd, and path(a, d). Every geodesic between a point in ab and a point in cd lies in Q. Suppose that an interior vertex q of path(b, c) is a vertex of path(a, d). If q = a or an interior vertex of path(a, d), then b is not visible from the closed edge cd by Lemma 1. Similarly, if q = d, then c is not visible from the closed edge ab. We conclude that path(b, c) and path(d, a) are disjoint, and Q is a simple polygon.

The geodesics path(a, c) and path(b, d) lie in Q, so any interior vertex of path(a, c) and path(b, d) is a vertex of Q. If path(a, c) and path(b, c) have a common interior vertex, then c is not visible from ab. Similarly, no two geodesics from $\{a, b\}$ to $\{c, d\}$ can have any common interior vertex. Hence all interior vertices of path(a, c) and path(b, d) are in $\{a, b, c, d\}$.

Corollary 7

- If {a, b, c, d} is in convex position, then path(a, c) = ac and path(b, d) = bd. Fig. 5, left.
- Otherwise suppose w.l.o.g. that $\operatorname{conv}(\{a, b, c, d\}) = \Delta(abc)$. Then $\operatorname{path}(a, c) = (a, d, c)$ and $\operatorname{path}(b, d) = bd$. Fig. 5, right.



Figure 5: The convex hull of two closed guard edges, ab and cd, is either a quadrilateral or a triangle.

Lemma 8 The intersection point $x = path(a, c) \cap path(b, d)$ is in the kernel of P.

Proof. It is enough to show that an arbitrary point p in polygon P is visible from x. By Corollary 7, the triangles $\Delta(abx)$ and $\Delta(cdx)$ lie inside P (one of the triangles may be degenerate). If p is in $\Delta(abx)$ or $\Delta(cdx)$, then segment px lies in the same triangle. Refer to Fig. 6.



Figure 6: A schematic of the proof that a simple polygon with four closed guard edges must be starshaped. Suppose that $g_1 = ab$, g_2 , $g_3 = cd$, and g_4 are guard edges. If a point $p \in P$ is not visible from $x = \text{path}(a, c) \cap \text{path}(b, d)$, then we show that p is also not visible from g_2 or g_4 .

Assume now that p is outside of both triangles and, w.l.o.g. it is on the right side of the directed geodesics path(a, c) and path(b, d). That is, p and the guard edge g_4 are on opposite sides of these geodesics.

If path(p, x) = px, then p is visible from x. Suppose, to the contrary, that path(p, x) is not a straight line segment. Assume w.l.o.g. that path(p, x) makes a *right* turn at its last interior vertex q. Then path(p, d) also makes a right turn at q. Since p is visible from the guard edge cd, we must have q = c by Lemma 1(b). Recall that any geodesic from p to a point in g_4 crosses both path(a, c) and path(b, d). Since path(p, x) makes a right turn at c, every geodesic from p to a point in g_4 also make a right turn at c. However, c is disjoint from g_4 , and by Lemma 1(b), p is not visible from g_4 , contradicting our initial assumption. We conclude that path(p, x) is a straight line segment, and so p is visible from x.

Proof of Theorem 5. If a simple polygon has four closed guard edges, then it has a nonempty kernel by Lemma 8, and thus is starshaped. So every non-starshaped simple polygon has at most three closed guard edges. $\hfill \Box$

5 Characterizing Open Guard Edges

In this section, we characterize the open guard edges of a simple polygon P in terms of the left and right kernels of P (defined below). This leads to a straightforward algorithm to find all open guard edges in P.

Left and right kernels. Recall that the set of points from which the entire polygon P is visible is the *kernel*, denoted K(P), which is the intersection of all halfplanes bounded by a supporting line of an edge of P and facing towards the interior of P. Lee and Preparata [5] designed an optimal O(n) time algorithm for computing the kernel of simple polygon with n vertices. We now define a weaker version of the kernel: the *left* and *right kernels* of P, denoted $K_{left}(P)$ and $K_{right}(P)$.

For every reflex vertex r, we define two polygons $C_{\text{left}}(r) \subset P$ and $C_{\text{right}}(r) \subset P$. Shoot a ray from r in a direction collinear with the edge incident to r preceding (resp., following) r in counterclockwise order; and let $C_{\text{left}}(r)$ (resp., $C_{\text{right}}(r)$) be the part of P on the left (resp., right) of the ray. These polygons have previously been defined in [3]. It is clear that if P is weakly visible from a set $S \subset P$, then S must intersect both $C_{\text{left}}(r)$ and $C_{\text{right}}(r)$ for every reflex vertex r.

Now we define $K_{\text{left}}(P)$ as the intersection of polygons $C_{\text{left}}(r)$ for all reflex vertices r; and $K_{\text{right}}(P)$ as the intersection of polygons $C_{\text{right}}(r)$ for all r. See Fig. 7 for an example. Clearly, we have

$$K(P) = K_{\text{left}}(P) \cap K_{\text{right}}(P).$$

By construction, both $K_{\text{left}}(P)$ and $K_{\text{right}}(P)$ are convex polygons, whose edges are collinear with some of the edges of P.

Left and right kernel decompositions. In the following lemma we use two decompositions of a polygon based on its left and right kernels. The *left kernel decomposition* is the decomposition of the polygonal domain P in the following way: One cell of the decomposition is the left kernel $K_{\text{left}}(P)$. The region inside Pbut in the exterior of $K_{\text{left}}(P)$ is decomposed by extending each edge of $K_{\text{left}}(P)$ in clockwise direction. Refer to Fig. 7. Since $K_{\text{left}}(P)$ lies on the left side of rays



Figure 7: The left and right kernels of a polygon. The dotted lines bound some polygons $C_{\text{left}}(r)$ and $C_{\text{right}}(r)$, but they are not part of the kernel decompositions.

emitted from reflex vertices of P, the clockwise extension of every edge of $K_{\text{left}}(P)$ reaches a collinear edge of P. The right kernel decomposition is defined analogously: one cell is $K_{\text{right}}(P)$, and the rest of P is decomposed by counter-clockwise extensions of the edges of $K_{\text{right}}(P)$. Note that if an open edge of P is disjoint from the left kernel, then it is adjacent to a unique region of the left kernel decomposition. Additionally, each region of the decomposition, except for $K_{\text{left}}(P)$, has exactly one common edge with the left kernel.

Lemma 9 An open edge e of a simple polygon P is a guard edge of P iff e intersects both the left and the right kernels of P.

Proof. Let e = uv be an open edge of P. First assume that e is disjoint from the left kernel $K_{\text{left}}(P)$. Then e is adjacent to a unique region in the left kernel decomposition of P. This region is adjacent to a unique edge k of $K_{\text{left}}(P)$, and k lies on a ray emitted by a reflex vertex r on P. Then e and the polygon $C_{\text{left}}(r)$ lies on opposite sides of the supporting line of k. Hence e does not intersect $C_{\text{left}}(r)$, and so it is not a guard edge.

Now assume that e = uv is not a guard edge, that is, there is a point $p \in P$ such that p is not visible from e. By Lemma 1(a), the geodesics path(p, u) and path(p, v) have common interior vertices. Let r be their last common vertex, which is necessarily a reflex vertex of P, and assume w.l.o.g. that both geodesics make a right turn at r. Then $p \in C_{left}(r)$, but e is disjoint from $C_{left}(r)$. That is, e is disjoint from the left kernel of P.

Finding all open guard edges in O(n) time. We use Lemma 9 to create a simple O(n) time algorithm for finding all open guard edges of a simple polygon P with n vertices (independent of whether P is starshaped or not). The left and right kernels of P can be computed in O(n) time using a modified version of the algorithm of Lee and Preparata [5], originally designed for computing the kernel K(P). For each edge of the left and right kernels, mark any intersection with the collinear edge of P. Now check the marks on all edges of P in O(n) time, and report those that intersect both the left and the right kernels.

6 Open Edge Guards

Recall that every simple polygon with n vertices can be covered by $\lfloor 3n/10 \rfloor + 1$ closed edge guards, and there are n-gons that require at least $\lfloor n/4 \rfloor$ closed edge guards. It turns out that the endpoints of the closed edge guards are crucial for these bounds. Significantly more edge guards may be necessary if the endpoints are removed.

We construct four different infinite families of polygons that require $\lfloor n/3 \rfloor$ open edge guards for n vertices. Refer to Fig. 8. The lower bounds for all our constructions can be verified by a standard "hidden point" argument. We hide $\lfloor n/3 \rfloor$ points (gray dots in Fig. 8) in the interior of a polygon such that each open edge guard sees exactly one such point. That is, each hidden point requires a unique open edge guard, and any set of fewer than $\lfloor n/3 \rfloor$ open edge guards would miss at least one hidden point.

It is not difficult to see that $\lfloor n/2 \rfloor$ open edge guards are always sufficient. Partition the set of edges of the polygon into two subsets for which the interior normals of the edges have either a positive or negative *y*component. Each subset of open edges jointly covers the entire polygon. We conjecture this upper bound is weak, and that $\lfloor n/3 \rfloor$ is the tight bound for the number of open edge guards necessary and sufficient to guard any simple polygon with *n* vertices.

References

- D. Avis, T. Gum, and G. T. Toussaint, Visibility between two edges of a simple polygon, *The Visual Computer*, 2:342–357, 1986.
- [2] D. Avis and G. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, *IEEE Tran. Comput.* C-30:910–914, 1981.
- [3] B. K. Bhattacharya, G. Das, A. Mukhopadhyay, and G. Narasimhan, Optimally computing a shortest weakly visible line segment inside a simple polygon, *Comput. Geom. Theory. Appl.*, 23:1–29, 2002.
- [4] D. Z. Chen, Optimally computing the shortest weakly visible subedge of a simple polygon, J. Algorithms 20:459–478, 1996.
- [5] D. T. Lee and F. Preparata, An optimal algorithm for finding the kernel of a polygon, J. ACM 26:415–421, 1979.
- [6] B.-K. Lu, F.-R. Hsu, and C. Y. Tang, Finding the shortest boundary guard of a simple polygon, *Theor. Comp. Sci.*, 263:113–121, 2001.
- [7] J. O'Rourke, Galleries need fewer mobile guards: A variation on Chvtal's theorem, *Geometriae Dedicata* 14:273– 283, 1983.



Figure 8: Examples of polygons requiring n/3 open edge guards. The gray dots in each polygon indicates a set of points that require a distinct edge to guard each point.

- [8] J. Park, S. Y. Shin, K. Chwa, and T. C. Woo, On the number of guard edges of a polygon, *Discrete Comput. Geom.*, 10:447–462, 1993.
- [9] T. C. Shermer, Recent results in art galleries, *Proc. IEEE*, 80:1384–1399, 1992.
- [10] S. Y. Shin and T. Woo, An optimal algorithm for finding all visible edges in a simple polygon, *IEEE Tran. Robotics and Automation* 5:202–207, 1989.
- [11] X. Tan, Fast computation of shortest watchman routes in simple polygons, *Inf. Proc. Lett.* 77:27-33, 2001.
- [12] G. Viglietta, Searching polyhedra by rotating planes, manuscript, arXiv:1104.4137, 2011.

Computing k-Link Visibility Polygons in Environments with a Reflective Edge

Salma S. Mahdavi *

Ali Mohades *

Bahram Kouhestani *

Abstract

In this paper we consider the k-link visibility polygon of an object inside a polygonal environment with a reflective edge called a *mirror*. The k-link visibility polygon of an object inside a polygon P is the set of all points in P, which are visible to some points of that object with at most k - 1 intermediate points, under the property that consecutive intermediate points are mutually visible. We propose an optimal linear time algorithm for computing the k-link visibility polygon of an object inside a polygon P with a reflective edge. The object can be a point, a segment or a simple polygon. We observed that in computing k-link visibility polygons the mirror can affect in only two levels. We explain how to handle these levels efficiently to achieve an optimal algorithm.

1 Introduction

The visibility problem is a fundamental topic in computational geometry and different versions of it, such as art gallery problems, have been studied. The visibility polygon of an object is defined as the set of all points visible to some points of that object. Several linear time algorithms have been proposed to compute the visibility polygon of a point [8], a segment [4] or a polygon [7]. The minimum link path between two points of a polygon P is a path inside P that connects these points and has the minimum number of straight edges, called *links*. The link distance between two points is the number of links in their minimum link path. Suri [10] gave an O(n) time algorithm for computing the minimum link path between two points in a simple polygon. The k-link visibility polygon of a point q can be defined by using the minimum link distance concept; it is the set of all points having the link distance of at most k from q. Using the window partitioning the k-link visibility polygon of a point can be computed in linear time [9]. In the visibility literature, reflective surfaces were first mentioned by Klee [5]. He asked if every polygon whose all edges are reflective can be illuminated from any interior point. Tokarsky [11] answered no to this question by constructing a polygon for which there exists a dark point by putting a light source at a particular position. In this polygon the dark point is collinear with both the light source and an edge of the polygon.

When all the edges of the polygon are reflective but each light beam is allowed to reflect once, Aronov et al. [2] showed that the resulting visibility polygon of a source light may not be simple. They present an $O(n^2 \log^2 n)$ algorithm for computing visibility polygons in such environments. Later they allowed at most r reflections for each light beam and presented an $O(n^{2r} \log n)$ time and $O(n^{2r})$ space algorithm to compute visibility polygons of a source light [1].

Recently Kouhestani et al. [6] showed how to compute visibility polygons in environments with a single reflective edge in an optimal O(n) time. In this paper, we extend this study and achieve a linear time algorithm for computing k-link visibility polygons in such environments. If the polygon is entirely contained within one of the 2 half-spaces determined by the line on which the mirror lies, the k-link visibility polygon of an arbitrary point can be easily computed by gluing P and the reflection of P in the mirror (called P') together along the reflective edge. Known algorithms for computing the k-link visibility polygon [9] can be slightly modified to operate in such a polygon. To return the visibility polygon, the union of two visibility polygons is computed in linear time using the algorithm of Kouhestani et al. [6]. In the case that the polygon intersects both these halfspaces and vertices of the mirror are reflex, the resulting polygon from gluing P and P' is not simple anymore. Apart from the difficulties to adopt known algorithms to operate in this polygon, this method needs improvements to run in an optimal time. Consider the smallest value of k for which the k-link visibility polygon enters P'. In the computation of (k+1)-link visibility polygon, points of P which are visible from points of k-link visibility polygon located in P must be added to those points of P visible from the part of k-link visibility polygon located in P. We have a similar situation for the points of (k + 1)-link visibility polygon located in P'. Therefore, in order to return the (k + 1)-link visibility polygon the union of four polygons must be computed. It is not clear how to accomplish this task in linear time to obtain an optimal algorithm.

Suppose the k-link visibility polygon is constructed incrementally and for example in i^{th} level the *i*-link visibility polygon is computed from the resulting polygon of the previous level. We observed that the reflective edge affects only two levels, therefore handling these levels ef-

^{*}Laboratory of Algorithm and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran, ss.mahdavi@aut.ac.ir, mohades@aut.ac.ir, b_kouhestani@alum.sharif.edu.

ficiently can produce a linear time algorithm. We clarify this observation in the first lemma of section 4.2.

This paper is organized as follows: Section 2 describes notation and preliminaries, section 3 shows how to compute the 2-visibility polygon of an object inside a polygon with a reflective edge, from which in section 4 an algorithm to compute k-link visibility polygons in such an environment is proposed. Section 5 contains conclusions and discussions.

2 Notation and Preliminaries

Let P be a simple polygon with n vertices. int(P) and bd(P) denote the interior and boundary of P, respectively. Two points $x, y \in P$ are mutually visible (or can see each other directly), if the open line segment, \overline{xy} , lies completely in int(P). An alternative definition used in many visibility papers, allows the line segment to touch the bd(P). Throughout this paper we use the former definition which is sometimes called the *clear visibility*. Two points x and y inside P are k-link visible (or for simplicity k-visible), if they can reach each other using k-1 intermediate points a_1, \ldots, a_{k-1} , under the property that a_i and a_{i+1} are mutually visible for $1 \le i \le (k-2)$ and x sees a_1 and y sees a_{k-1} . The visibility polygon of a point q in P, denoted by $V_P(q)$ is the set of all points in P visible to q. An edge of $V_P(q)$ that is not a part of an edge of P is called a window of $V_P(q)$. Suppose w_i is a window of $V_P(q)$. w_i partitions P into two subpolygons. The subpolygon which does not contain $V_P(q)$ is called the *pocket* of w_i (*pocket*(w_i)).

The k-link visibility polygon of a point q in P, denoted by $V_P^k(q)$ is the set of all points in P which are k-visible to q. The weak visibility polygon of a segment s denoted by $WV_P(s)$ is the set of all points of P visible to some points on s, except the endpoints. In the same manner, the k-(weak)visibility polygon of a segment s, $V_P^k(s)$ can be defined. Let q be a point inside P. P can be partitioned into regions such that all points in the same region have the same link distance from q. This partitioning is called the *window partitioning* with respect to the point q and can be done in O(n) time [9]. Two regions are neighbors if they share a common window. The dual graph of the window partitioning is achieved by considering a node for each region and connecting each two nodes if their corresponding regions are neighbors.

Suppose one of the edges of P is reflective, this edge is called a *mirror*. Two points x and y can see each other through the mirror e (or indirectly), if and only if there exist a point r lying on e, visible to both x and y such that \overline{xr} and \overline{yr} lie on the opposite sides of the inward normal at r and make the same angle with it. r is not considered as an intermediate point and hence x and y are 1-link visible. Figure 1 illustrates two points y and

z which are 2-link visible to the point x with the intermediate points i_1 and i_2 , respectively.



Figure 1: An illustration of 2-link visibility when one of the edges of the polygon is reflective.

Let $V_{P,e}(q)$, (resp. $V_{P,e}^{k}(q)$) denote the visibility polygon (resp. the k-link visibility polygon) of a point q inside P with the reflective edge e. Note that in $V_{P,e}^{k}(q)$, consecutive intermediate points can see each other directly or by using the mirror e.

3 The 2-visibility polygon of objects in a polygon with a single reflective edge

We first concentrate on computing the 2-visibility polygon of an object when an edge of the polygon is reflective. Let P be a simple polygon with n vertices and ebe the reflective edge of P. Let O be an object inside P. The general idea of computing $V_{P,e}^2(O)$ is to compute $V_{P,e}(O)$ and identify parts of P that are 1-visible to $V_{P,e}(O)$. It is easy to see that any point of these parts is visible to a window of $V_{P,e}(O)$. If O is a point or a segment $V_{P,e}(O)$ is computed in O(n) time [6]. When O is a simple polygon with m vertices, we use the following process to compute $V_{P,e}(O)$:

First we compute $V_P(O)$ in O(n+m) time by using the algorithm of Langetepe et al. [7], and then determine the parts of P that O sees through e. This can be done with a slight modification to the algorithm of Kouhestani et al. [6]. The union of these parts is $V_{P,e}(O)$ which can be computed in O(n+m) time [6].

Lemma 1 Let Q be a simple polygon with m vertices inside the simple polygon P. $V_{P,e}(Q)$ can be computed in O(n+m) time.

 $V_{P,e}^2(O)$ is the set of all points in P which see some points of O using at most one intermediate point. The points of $V_{P,e}^2(O)$ can be categorized into four groups: (a) The points which see the intermediate point directly for which the intermediate point sees the object directly. (b) The points which see the intermediate point using the mirror, but for which the intermediate point sees the object directly.

(c) The points which see the intermediate point directly, but for which the intermediate point sees the object using the mirror.

(d) The points which see the intermediate point using the mirror for which the intermediate point sees the object using the mirror.

We denote these groups by $\{-e, -e\}$, $\{+e, -e\}$, $\{-e, +e\}$, $\{+e, +e\}$, in which -e means seeing directly and +e means seeing through the mirror e.

Lemma 2 Let P be a simple polygon with a reflective edge e, and O be an object completely inside P. At most two windows of $V_{P,e}(O)$ can intersect e.

Proof. Any line segment in P, can intersect with at most two windows of a visibility polygon [3]. Therefore, the edge e intersects at most two windows of $V_{P,e}(O)$.



Figure 2: The windows $w_1 = (c, i_1, d)$ and $w_2 = (a, i_2, b)$ with interior points i_1, i_2 . The shaded regions show the pocket of w_1 and w_2 .

Note that some windows of $V_{P,e}(O)$ may have an interior point (see Figure 2).

Lemma 3 Suppose w_i is a window of $V_{P,e}(O)$ and its pocket dose not intersect the mirror edge e. Then, all points of $V_{P,e}^2(O)$ which are located in pocket (w_i) , are directly visible from w_i . Therefore, other windows of $V_{P,e}(O)$ can not see additional points in pocket (w_i) and w_i can not see new points in pocket (w_i) through the mirror.

Proof. Let w_j be a window of $V_{P,e}(O)$ and $w_j \neq w_i$. If a point x in $pocket(w_i)$ is visible to w_j (either directly or by using the mirror), the line segment (or the path) $\overline{w_j x}$ will intersect w_i in a point y. Therefore, x and y are visible and x can see w_i directly (see Figure 3). With a similar argument w_i can not see new points in its pocket through the mirror. \Box



Figure 3: The illustration of Lemma 3.

Lemma 4 Let w_i be a window of $V_{P,e}(O)$ whose pocket intersects the reflective edge e. The set of points of $V_{P,e}^2(O)$ located in pocket (w_i) is equal to $V_{pocket(w_i),e}(w_i)$.

Proof. The proof is similar to the proof of the previous lemma. $\hfill \Box$

Lemma 5 Let Q be a polygon constructed by adding all $V_{pocket(w_i),e_i}(w_i)$ to $V_{P,e}(O)$, where w_i is a pocket of $V_{P,e}(O)$ and e_i is the intersection of e and pocket (w_i) . Then, Q is $V_{P,e}^2(O)$.

Proof. Windows of $V_{P,e}(O)$ are 1-visible to O, so newly added points are 2-visible to O. Therefore, Q is a subset of $V_{P,e}^2(O)$. As mentioned before, points of $V_{P,e}^2(O)$ can be categorized into four groups. We show that a point of each group lies in Q:

Case 1: $\{-e, -e\}$; a point of type $\{-e\}$ is in $V_P(O)$, and $\{-e, -e\}$ is a point which is directly visible to $V_P(O)$, so it is visible to a window of $V_P(O)$ and is located in the pocket of this window. $V_P(O)$ is a subset of $V_{P,e}(O)$, therefore, the point is in Q.

Case 2: $\{+e, -e\}$; a point of type $\{+e\}$ is in $V_{P,e}(O)$, $\{+e, -e\}$ is a point which is directly visible to a window of $V_{P,e}(O)$, so it is located in the pocket of this window, therefore, it is in Q.

Case 3: $\{-e, +e\}$; a point of type $\{-e, +e\}$ is visible to $V_P(O)$ by using the mirror, $V_P(O)$ is a subset of $V_{P,e}(O)$, so this point is visible to a window of $V_{P,e}(O)$ by using the mirror, and therefore, it is in Q.

Case 4: $\{+e, +e\}$; $\{+e, +e\}$ is a point visible to a window of $V_{P,e}(O)$ by using the mirror, so it is in Q. \Box

Now we can present an algorithm to compute the 2-visibility polygon of an object O in linear time.

Algorithm 3

- 1. Compute $V_{P,e}(O)$.
- 2. Let $\{w_1, ..., w_d\}$ be the windows of $V_P(O)$ and e_i be the intersection of e and $pocket(w_i)$ for i = 1, 2, ..d.
- 3. Compute all $V_{pocket(w_i),e_i}(w_i)$ and add them to $V_{P,e}(O)$.
- 4. Return the resulting polygon.

The time complexity of the algorithm:

Step 1 is computed in O(n + m) time due to Lemma1. Suppose n_i is the number of vertices of $pocket(w_i)$ for i = 1, 2, ...d. $V_{pocket(w_i),e_i}(w_i)$ is computed in $O(n_i)$ time. Each two pockets of $V_{P,e}(O)$ can at most share one vertex, therefore, the sum of vertices of these pockets is O(n) and step 3 is computed in linear time. Therefore, we can conclude the following theorem.

Theorem 6 The 2-visibility polygon $V_{P,e}^2(O)$ of an object O inside a simple polygon P with a reflective edge e, can be computed in O(n + m) time, where n is the number of vertices of P and m is the complexity of O.

4 *k*-visibility polygons

4.1 The *k*-visibility polygon of an object

Let O and $s = \overline{xy}$ be an object and a segment inside a simple polygon P with n vertices. The minimum link path between O and s, is a path in P, connecting some points of O to s, which has the minimum number of links. The link distance between O and s, is the number of the links in their minimum link path. The set of all points with the link distance of 1 from O is the visibility polygon of O. Let w_i be the window of $V_P(O)$ which its pocket completely contains s. If there is no such a window then the link distance between O and sis 1. Otherwise, the link distance between O and s is 1 +the link distance between s and w_i . The link distance between two segments can be computed in O(n)time [10, 9]. Therefore, the link distance between an object and a segment can be computed in linear time. Note that the window partitioning is computed using the link distance concept [9], so the time complexity of the window partitioning of an object is linear.

Lemma 7 Let P be a simple polygon with n vertices and O be an object inside P with the complexity of m. $V_P^k(O)$ can be computed in O(n+m) time.

Proof. A point x inside P is in $V_P^k(O)$ if the link distance between O and x is less or equal to k. By using the window partitioning of O all points with the link distance of at most k from O can be computed in O(n+m) time.

4.2 The *k*-visibility polygon of an object inside an environment with a single reflective edge

Let e be a reflective edge in P. Two points in P are k-visible if they can see each other using at most k-1 intermediate points. Consecutive intermediate points see each other directly or by using e. $V_{P,e}^k(O)$ is the set of all points which are k-visible to some points of O. By using the computation of the 2-visibility polygon of O, we present an algorithm to compute $V_{P,e}^k(O)$.

For an example of a 3-link visibility polygon of a point in the presence of a mirror, see Figure 4. In this figure, windows of each level are shown with the same color.

Lemma 8 Let P be a simple polygon with a reflective edge e and O be an object inside P. Let m be the link distance between O and e. Suppose $Q = V_{P,e}^{m+1}(O)$ is constructed. $V_{P,e}(Q)$ is equal to $V_P(Q)$.

Proof. *O* and *e* are *m*-link visible, therefore, *e* is completely inside $V_P^{m+1}(O)$ and no pockets of $V_P^{m+1}(O)$ intersect *e*. Let *pock*_i be a pocket of $V_P^{m+1}(O)$. Similar to Lemma 3, all points of $V_P^{m+2}(O)$ in *pock*_i are directly visible to the window of *pock*_i. Therefore, $V_{P,e}(Q)$ is equal to $V_P(Q)$.

Now we present an algorithm to compute $V_{P,e}^k(O)$:

Algorithm 4.2

- 1. Compute the link distance between O and e and store it in m.
- 2. If k < m, then, compute $V_P^k(O)$ and return.
- 3. If $k \ge m$, then,
 - (a) Compute $Q = V_P^{m-1}(O)$.
 - (b) If k = m, compute $V_{P,e}(Q)$ and return.
 - (c) Compute $R = V_{p,e}^2(Q)$
 - (d) Let $\{w_1, w_2, ..., w_d\}$ be windows of R.
 - (e) Compute $V_{pocket(w_i)}^{k-m-1}(w_i)$ for all i = 1, 2, ..., dand add them to R
 - (f) Return the resulting polygon

Theorem 9 Let P be a simple polygon with n vertices and O be an object inside P, with the complexity of m. Let e be a reflective edge of P. $V_{P,e}^k(O)$ can be computed in O(n+m) time.

Proof. The algorithm 4.2 computes $V_{P,e}^k(O)$. Lemma 8 ensures the correctness of the algorithm. We show that the time complexity of this algorithm is O(n+m). Steps 3(a), 3(b), 3(c) and 3(d) are computed in O(n+m) time due to Lemma 7, Lemma 1 and Theorem 6. Suppose the number of vertices of $pocket(w_i)$ is n_i , for all i =



Figure 4: The 3-link visibility polygon of a point, when one of the edges of the polygon is reflective.

1,2,...,d. $V_{pocket(w_i)}^{k-m-1}(w_i)$ is computed in $O(n_i)$. The sum of all n_i for i = 1, 2, ..., d is less or equal to n. Therefore, step 3(e) runs in O(n) time and the algorithm has the time complexity of O(n+m).

Note that the time complexity of computing $V_{P,e}^k(O)$ is independent to the value of k and for any k = 1, ..., n, $V_{P,e}^k(O)$ is computed in O(n+m) time.

5 Conclusion

In this paper we presented a linear time algorithm to compute the k-link-visibility polygon of an object inside a polygonal environment with a reflective edge. The object is considered to be a point, a segment or a simple polygon. If the polygon has two or more reflective edges, a light beam can be reflected repeatedly between these mirrors. It is not clear how to compute the visibility polygon of a point when there is no restriction on the number of the reflections. An interesting question will be how to compute the k-link visibility polygon of a point in a polygon with m reflective edges when each light beam can reflect at most t time. This question is the subject of our future study.

Acknowledgments

We would like to thank Mansour Davoudi and Farnaz Sheikhi for their fruitful comments.

References

 B. Aronov, A. Davis, T. K. Dey, S. P. Pal and D. C. Prasad. Visibility with multiple reflections. *Discrete* and Computational Geometry, 20: 61-78, 1998.

- [2] B. Aronov, A. Davis, T. K. Dey, S. P. Pal and D. C. Prasad. Visibility with one reflection. *Discrete and Computational Geometry*, 19: 553-574, 1998.
- [3] P. Bose, A. Lubiw, and J. Munro. Efficient visibility queries in simple polygons. *Computational Geometry: Theory and Applications*, 23: 313-335, 2002.
- [4] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2: 209-233, 1987.
- [5] V. Klee. Is every polygonal region illuminable from some point? Computational Geometry: Amer.Math. Monthly, 76: 180, 1969.
- [6] B. Kouhestani, M. Asgaripour, S. S. Mahdavi, A. Nouri and A. Mohades. Visibility Polygons in the Presence of a Mirror Edge. In Proc. 26th European Workshop on Computational Geometry, 26: 209-212, 2010.
- [7] E. Langetepe, R. Penninger and J. Tulke. Computing the visibility area between two simple polygons in linear time. In Proc. 26th European Workshop on Computational Geometry, 26: 237-240, 2010.
- [8] D. T. Lee. Visibility of a simple polygon. Computer Vision, Graphics, and Image Processing, 22: 207-221, 1983.
- [9] A. Maheshwari, J. -R. Sack and H. N. Djidjev. Link Distance Problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, 519-558, 2000.
- [10] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. Computer Graphics Vision, and Image Processing, 35: 99-110, 1986.
- [11] G. T. Tokarsky. Polygonal rooms not illuminable from every point. American Mathematical Monthly, 102: 867-879, 1995.

Edge-guarding Orthogonal Polyhedra

Nadia M. Benbernou^{*}

[•] Erik D. Demaine[†] Godfried Toussaint[§] Martin L. Demaine[†] Jorge Urrutia[¶] Anastasia Kurdia Giovanni Viglietta[∥] Joseph O'Rourke[‡]

Abstract

We address the question: How many edge guards are needed to guard an orthogonal polyhedron of e edges, r of which are reflex? It was previously established [3] that e/12 are sometimes necessary and e/6 always suffice. In contrast to the closed edge guards used for these bounds, we introduce a new model, open edge guards (excluding the endpoints of the edge), which we argue are in some sense more natural in this context. After quantifying the relationship between closed and open edge guards, we improve the upper bound to show that, asymptotically, (11/72)e (open or closed) edge guards suffice, or, in terms of r, that (7/12)r suffice. Along the way, we establish tight bounds relating e and r for orthogonal polyhedra of any genus.

1 Introduction

We consider a natural variation of the famous Art Gallery Problem: given an orthogonal polyhedron \mathcal{P} (possibly with holes) in \mathbb{R}^3 , select a minimum number of edges of \mathcal{P} (called *edge guards*) so that the interior of \mathcal{P} is fully guarded (i.e., each point of \mathcal{P} is visible to at least one guard).

Although traditionally edge guards are *closed* in that they occupy the entire edge, we suggest that *open edge guards*, which exclude their endpoints, are a more natural model. We establish that at most three times as many open edge guards are needed to cover the same polyhedron as closed edge guards, a tight bound. De-

[¶]Instituto de Matemáticas, Universidad Nacional Autónoma de México. Partially supported by SEP-CONACYT of Mexico, Proyecto 80268 and by Spanish Government under Project MEC MTM2009-08652. spite this apparent weakness, we improve the previous upper bound for closed edge guards to a better bound for open edge guards.

Open guards, open polyhedra. In line with our focus on open edge guards, we also focus on guarding open polyhedra, i.e., bounded polyhedra excluding their boundaries. Consider an orthogonal polyhedron to represent an empty room with solid walls, with the task to place guards who can detect unwelcome intruders. Because an intruder cannot hide within a wall but rather must be located inside the room, there is no need to guard the walls of the room, i.e., the boundary of the polyhedron. A guarding problem can alternatively be viewed as an illumination problem, with guards acting as light sources. Incandescent lights are modeled as point guards, and fluorescent lights are modeled as segment guards. In the latter case, it is more realistic to disregard the endpoints of the edge guards. The amount of light that a point interior to the polyhedron receives is proportional to the total length of the segments illuminating that point. Employing the open edge-guard model ensures that if a point is illuminated, it receives a strictly positive amount of light, and makes the model more realistic.

Notice that these two definitions of illuminated points (visible to an open edge guard or receiving a strictly positive amount of light from closed edge guards) cease to be equivalent if we consider polyhedra with boundary.

Previous work. Although guarding orthogonal polygons is a relatively well-studied topic, few positive results exist for orthogonal polyhedra. To the best of our knowledge, the only results relevant to the problem studied in this paper were given by Urrutia in his survey [3, Sec. 10]: for a polyhedron of e edges, e/6 closed edge guards always suffice, and e/12 guards are sometimes necessary (Figure 1). He also conjectured that the latter is the correct bound, i.e., that e/12 + O(1) suffice. Because no proof of the upper bound was given in [3], another contribution here is that our proof for the (11/72)e bound incorporates the essence of Urrutia's unpublished e/6 proof.

^{*}Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA, nbenbern at mit.edu

[†]MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine, mdemaine}@mit.edu

[‡]Department of Computer Science, Smith College, Northampton, MA, USA, orourke@cs.smith.edu

[§]Department of Music, Harvard University, Cambridge, MA, USA, Department of Computer Science, Tufts University, Medford, MA, USA, School of Computer Science, McGill University, Montreal, QC, Canada, godfried at cs.mcgill.ca

^{||}Department of Computer Science, University of Pisa, Italy, vigliett@di.unipi.it. Partially supported by MIUR of Italy under project AlgoDEEP prot. 2008TFBWL4.



Figure 1: Lower bound example from [3]: k guards are required to guard a polyhedron with a total of 12k + 12 edges.

2 Properties of orthogonal polyhedra

We start with precise definitions of necessary concepts. Given two points x and y, we denote by xy the (closed) straight line segment joining x and y, and by \widetilde{xy} the corresponding *open segment*, i.e., the relative interior of xy.

Orthogonal polyhedra. A *cuboid* is defined as a compact subset of \mathbb{R}^3 bounded by 6 axis-orthogonal planes. The union of a finite non-empty set of cuboids is an *orthogonal polyhedron* if its boundary is a connected 2-manifold.

A face of an orthogonal polyhedron is a maximal planar subset of its boundary, whose interior is connected and non-empty. Faces are orthogonal polygons with holes, perhaps with degeneracies such as hole boundaries touching each other at single vertex, etc. A vertex of an orthogonal polyhedron is any vertex of any of its faces. An *edge* is a minimal positive-length straight line segment shared by two faces and connecting two vertices of the polyhedron. Each edge, with its two adjacent faces, determines a dihedral angle, internal to the polyhedron. Each such angle is 90° (at a convex edge) or 270° (at a reflex edge).

Visibility and guarding. Visibility with respect to a polyhedron \mathcal{P} is a relation between points in \mathbb{R}^3 : point x sees point y (equivalently, y is visible to x) if $xy \setminus \{x\}$ lies entirely in the interior of \mathcal{P} . Note that, according to the previous definition, the boundary of \mathcal{P} occludes visibility; no portion of xy, except the endpoint x, can lie on the boundary of \mathcal{P} . Also, x is assumed to be invisible to itself when it belongs to the boundary. Given a point $x \in \mathcal{P}$, its visibility region V(x) is the set of points that are visible to x. Similarly, the visibility region of a set $X \subseteq \mathbb{R}^3$, denoted by V(X), is the set of points that are visible to at least one point in X.

The Art Gallery Problem we consider in this paper is: given an orthogonal polyhedron \mathcal{P} , efficiently select a (sub)set of its edges e_1, e_2, \ldots, e_k , called the *guarding* set, so that the whole interior of \mathcal{P} is guarded by the interiors of the selected edges. In other words, the interior of \mathcal{P} must coincide with $V(\tilde{e_1}) \cup V(\tilde{e_2}) \cup \ldots \cup V(\tilde{e_k})$. Our goal is also to minimize k, the number of selected edges. We bound k with respect to the total number e of edges of \mathcal{P} , or the number r of its reflex edges.

The notion of ε -guarding implies that each point is guarded by at least one positive-length segment. Guarding in our open polyhedra model is equivalent to ε guarding in that, if a point is guarded, then it is also guarded by a positive-length segment, lying on some guard.



Figure 2: The six vertex types.

Vertex classification. Based on the number of incident reflex and convex edges, the vertices of orthogonal polyhedra form six distinct classes, denoted here by A, B, C, D, E and F, and are introduced as follows. Consider the eight octants determined by the coordinate axes intersecting at a given vertex, and place a sufficiently small regular octahedron around the vertex, such that each of its faces lies in a distinct octant. By definition, the set of faces that fall inside (or outside) the polyhedron is connected: recall that the boundary of a polyhedron is a 2-manifold. Consider all possible ways of partitioning the faces of the octahedron into two non-empty connected sets, up to isometry (refer to Figure 2):

- There is essentially a single way to select 1 face (resp. 7 faces). This corresponds to an A-vertex (resp. a B-vertex).
- There is a single way to select 2 faces (resp. 6 faces). This case does not correspond to a vertex of the orthogonal polyhedron: it implies that the considered point is not a vertex of any face on which it lies.
- There is a single way to select 3 faces (resp. 5 faces). This corresponds to a D-vertex (resp. a C-vertex).
- There are three ways to select 4 faces. One of them implies that the point lies in the middle of a face, hence it does not correspond to a vertex. The other two choices correspond to an E-vertex and an F-vertex, respectively.

Auxiliary results. We now present two useful properties of orthogonal polyhedra that will be employed to prove our main results. Let us denote by A the number of A-vertices in a given orthogonal polyhedron, and so on, for each vertex class.

Lemma 1 In every orthogonal polyhedron with r > 0reflex edges, $3A + D \ge 28$.

Proof. Consider the bounding cuboid of the polyhedron and the set of orthogonal polygons (perhaps with holes, without degeneracies), formed by intersection of the faces of the cuboid and the polyhedron. The vertices of those polygons are either A-vertices or D-vertices of the polyhedron (convex vertices are A-vertices, and reflex vertices are D-vertices). Our strategy is to only look at the vertices belonging to the bounding faces and ensure that there is a sufficient number of them. Namely, we only need to show that there are at least

- (a) 10 A-vertices, or
- (b) 9 A-vertices and 1 D-vertex, or
- (c) 8 A-vertices and 4 D-vertices.

Suppose each face of the bounding cuboid contains exactly one rectangle. If all the vertices of these rectangles coincide with the corners of the bounding cuboid, then the polyhedron is convex, contradicting the assumptions. Hence, there is a vertex x that is not a corner of the bounding cuboid. Let f denote a face containing x. At least one of the vertices, denoted by y, adjacent to x in the rectangle contained in f, is such that \widetilde{xy} does not lie on an edge of the bounding cuboid. Let f' be the bounding face opposite to f, and f'' be the bounding face chosen as shown in Figure 3: out of



Figure 3: An illustration of the proof of Lemma 1.

the 4 faces surrounding f, f'' is the one that lies on the "side" of xy. f and f' contain two disjoint rectangles, and thus exactly 8 distinct A-vertices. Additionally, f'' has two extra A-vertices, lying on an edge x'y' parallel to xy (refer to Figure 3). Collectively, f, f' and f'' contain at least 10 A-vertices, so (a) holds.

On the other hand, if there exists a bounding face f whose intersection with the polyhedron is not a single rectangle, then we need to analyze the following three cases. Let f' be the bounding face opposite to f.

- If f contains at least two polygons (those polygons' boundaries must be disjoint because f is a bounding face), then collectively f and f' contain at least 12 distinct A-vertices, so (a) holds. Indeed, every orthogonal polygon has at least 4 convex vertices.
- If f contains a polygon with at least one hole, then the polygon's external boundary contains at least 4 convex vertices (equiv. A-vertices), and the hole has at least 4 reflex vertices (equiv. D-vertices). f' also contains at least 4 convex vertices (A-vertices). Together f and f' contain at least 8 A-vertices and 4 D-vertices, so (c) holds.
- If f contains just one polygon, which is not convex, then such a polygon has at least 5 convex vertices and one reflex vertex. Together with f', there are at least 9 A-vertices and 1 D-vertex, so (b) holds.

Theorem 2 For every orthogonal polyhedron with e edges in total, r > 0 reflex edges and genus $g \ge 0$,

$$\frac{1}{6}e + 2g - 2 \leqslant r \leqslant \frac{5}{6}e - 2g - 12$$

holds. Both inequalities are tight for every g.

Proof. Let c = e - r be the number of convex edges. Let A be the number of A-vertices, etc.. Double counting

the pairs (edge, endpoint) yields (refer to Figure 2)

$$2c = 3A + C + 2D + 3E + 2F,$$
 (1)

$$2r = 3B + 2C + D + 3E + 2F.$$
 (2)

The angle deficit (with respect to 2π) of A- and Bvertices is $\pi/2$, the deficit of C- and D-vertices is $-\pi/2$, the defect of E- and F-vertices is $-\pi$. Hence, by the polyhedral version of Gauss-Bonnet theorem (see [2, Thm. 6.25]),

$$A + B - C - D - 2E - 2F = 8 - 8g.$$
 (3)

Finally, since all the variables involved are non-negative,

$$9B + 3C + 3E + F \ge 0. \tag{4}$$

Subtracting 3 times (3) from 2 times (4) yields

$$-3A + 15B + 9C + 3D + 12E + 8F \ge 24g - 24.$$

Further subtracting (1) and adding 5 times (2) to the last inequality yields

$$2c - 10r + 24g - 24 \le 0,$$

which is equivalent to $\frac{1}{6}e + 2g - 2 \leq r$.

To see that the left-hand side inequality is tight for every r and g, consider the staircase-like polyhedron with holes depicted in Figure 4. If the staircase has k"segments" and g holes, then it has a total of 6k+12g+6edges and k+4g-1 reflex edges.



Figure 4: A polyhedron that achieves the tight left-hand side bound in Theorem 2.

According to Lemma 1, $3A + D \ge 28$, unless the polyhedron is a cuboid. Then

$$9A + 3D + 3E + F \ge 84. \tag{5}$$

Subtract 3 times (3) from 2 times (5):

$$15A - 3B + 3C + 9D + 12E + 8F \ge 24g + 144.$$

Subtract (2) and add 5 times (1):

$$2r - 10c + 24g + 144 \ge 0,$$

which is equivalent to $r \leq \frac{5}{6}e - 2g - 12$.

To see that the right-hand side inequality is also tight, consider a cuboid with a staircase-like well carved in it, and a number of cuboidal "poles" carved out from the surface of the well (i.e., the *negative* version of Figure 4). If the staircase has k "segments" and g poles, then the polyhedron has a total of 6k + 12g + 18 edges and 5k + 8g + 3 reflex edges.

Notice that the statement of the previous theorem does not hold if we change the definition of orthogonal polyhedron by dropping the condition of connectedness of the boundary. Indeed, consider a cube and remove several smaller disjoint cubic regions from its interior. The resulting shape has unboundedly many reflex edges and just 12 convex edges.

Finally, the next proposition characterizes visibility regions of points belonging to polyhedra.

Proposition 3 The visibility region of any point in a polyhedron or on its boundary is an open set.

Proof. Let x be a point in a polyhedron \mathcal{P} . Let f be a face of \mathcal{P} , not containing x. The region of space "occluded" by f is a closed set O(x, f), shaped like a truncated unbounded pyramid with apex x and base f. The number of faces is finite. Forming the union of all O(x, f), for every face f not containing x, we obtain a closed set O(x).

The region occluded by the faces containing x is the corresponding (unbounded) solid angle, external with respect to \mathcal{P} , which is a closed set. Its union with O(x) is again a closed set, and therefore the complement of O(x) is an open set, which by definition is V(x).

Observe that the visibility regions of open and closed edges are also open sets, since they are unions of open sets.

3 Open vs. closed edge guards

We now establish the relationship between the number of open and closed edge guards required to guard the interior of an orthogonal polyhedron.

Theorem 4 Any orthogonal polyhedron guardable by k closed edge guards is guardable by at most 3k open edge guards, and this bound is tight.

Proof. Given a set of k closed edges that guard the entire polyhedron, we first construct a guarding set of open edges of size at most 3k and then show that this set also guards the entire polyhedron. The construction is simple: for each closed edge uv from the original guarding set, place the open edge \widetilde{uv} into the new guarding set. For the endpoint u, also add a reflex edge \widetilde{uw} with $w \neq v$, if such edge exists, or any other edge incident to u otherwise. Similarly, an incident edge is selected for the other endpoint v. Hence, for each edge of the original guarding set, at most 3 open edges are placed in the new guarding set.

To prove the equivalence of the two guarding sets, we need to show that the volume that was guarded by an endpoint u of the closed edge from the original guarding set, is guarded by some point belonging to the interior of uv or the interior of uw, as chosen above, i.e., $V(u) \subseteq V(\widetilde{uv}) \cup V(\widetilde{uw})$.

Let x be any point previously guarded by $u, x \in V(u)$. By Proposition 3, a ball \mathcal{B} centered at x belongs to V(x). Then we create a right circular cone \mathcal{C} with apex u, whose base is centered at x and is contained in \mathcal{B} . Clearly, $\mathcal{C} \subset V(u)$. Let \mathcal{D} be a small-enough ball centered at u that does not intersect any face of the polyhedron except those containing u (refer to Figure 5). We prove that $\mathcal{D} \cap \mathcal{P} \subseteq V(\widetilde{uv} \cap \mathcal{D}) \cup V(\widetilde{uw} \cap \mathcal{D})$.

If u is an A-vertex, then $\mathcal{D} \cap \mathcal{P} \subseteq V(\widetilde{uv} \cap \mathcal{D})$. If u is a B-vertex (as illustrated), then of the 8 octants determined by orthogonal planes crossing at u, one is external to \mathcal{P} . Out of the 7 octants that need to be guarded, 6 are guarded by $\widetilde{uv} \cap \mathcal{D}$. The same holds for \widetilde{uw} , and together they guard all 7 octants (two of the octants guarded by \widetilde{uv} are missing a face, but those two faces are guarded by \widetilde{uw}).

In all other cases (*u* is a *C*-, *D*-, *E*- or *F*-vertex), either *uv* or *uw* is a reflex edge. Assume without loss of generality that *uv* is reflex. Then, $\widetilde{uv} \cap \mathcal{D}$ sees all of $\mathcal{D} \cap \mathcal{P}$ (refer to Figure 2).



Figure 5: Construction from the proof of Theorem 4.

The boundaries of \mathcal{D} and \mathcal{C} intersect at a circle of radius $\rho > 0$. Let y be the center of that circle. There is a point z on $\widetilde{uv} \cap \mathcal{D}$ or on $\widetilde{uw} \cap \mathcal{D}$ that sees y, and hence the entire open segment \widetilde{uz} sees y. Pick a point t on \widetilde{uz} such that $||ut|| < \rho$. Then t sees x.

A similar argument holds for the visibility region of the other endpoint, v, of uv.

To see that 3 is the best achievable ratio between the number of open and closed edge guards, consider the polygon in Figure 6 and extrude it to an orthogonal prism. Each large dot in that figure represents the projection of some distinguished point located in the interior of the prism. The only (closed) edges that can see more than two selected points are the highlighted edges (on the lower or upper base of the prism). Picking those edges as guards yields the minimum set of guards, and together they guard the entire polyhedron. On the other hand, the relative interior of any edge can see at most



Figure 6: Matching ratio in Theorem 4. Notice that the same example also solves the corresponding problem for 2D polygons.

one point of interest. At least as many open edge guards as there are distinguished points are necessary. \Box

Note that the above analysis does not hold in the case of closed polyhedra, i.e., when the boundary does not obstruct visibility, since we can no longer argue that a single closed edge guard is locally dominated by 3 open edge guards.

4 Upper bound

We now establish an upper bound on the number of open edge guards required to guard an orthogonal polyhedron.

Theorem 5 Every orthogonal polyhedron with e edges in total and r reflex edges is guardable by $\lfloor \frac{e+r}{12} \rfloor$ open edge guards.

Proof. Let e_x and r_x be the number of X-parallel edges and reflex edges, respectively; e_y , e_z , r_y , r_z are similarly defined. Without loss of generality, assume X is the direction that minimizes the sum $e_x + r_x$, so that $e_x + r_x \leq \frac{e+r}{3}$. Of course, a guard on every X-parallel edge suffices to cover all of \mathcal{P} , but we can do much better with a selected subset of these edges. We argue below that selecting the three types of X-parallel edges circled in Figure 7 suffice (as do three other symmetric configurations). Let the number of X-edges of each of the eight types shown be α, \ldots, δ' as labeled in Figure 7.

Hence we could place $\alpha + \beta' + \delta'$ guards, or $\gamma + \beta' + \delta'$ guards, or $\beta + \alpha' + \gamma'$ guards, or $\delta + \alpha' + \gamma'$ guards.

By choosing the minimum of these four sums, we place at most

$$(\alpha + \beta + \gamma + \delta + 2\alpha' + 2\beta' + 2\gamma' + 2\delta')/4$$
$$= \frac{e_x + r_x}{4} \leqslant \frac{e + r}{12}$$

guards.

Next we prove that our guard placement works.

We consider any point p in \mathcal{P} and show that p is guarded by the edges selected in Figure 7. Let ω be the



Figure 7: Possible configurations of X-edges. The Xaxis is directed toward the reader. The circled configurations are those selected in the proof of Theorem 5.

X-orthogonal plane containing p and let Q be the intersection of the (open) polyhedron \mathcal{P} with ω . To prove that p is guarded, we first shoot an axis-parallel ray from p. For our choice of guarding edges, the ray is directed upward. Let q be the intersection point of the ray and the boundary of Q that is nearest to p. Next, grow leftwards a rectangle whose right side is pq until it hits a vertex v of Q. If it hits several simultaneously, let v be the topmost. There are three possible configurations for v, shown in Figure 8, and each corresponds to a selected configuration in our placement of guards (Figure 7). If v lies in the interior of the guarding edge, then p is guarded. If v is an endpoint of such an edge, then we show that p is guarded by a sufficiently small neighborhood of v that belongs to the guarding edge. Every face of \mathcal{P} that does not intersect ω has a positive distance from ω . Let d be the smallest such distance. Then, the points of the guarding edge at distance strictly less than d from v see p.

If a different triplet of guarding edges is chosen, the above construction is suitably rotated by a multiple of 90° .



Figure 8: An illustration of the proof of Theorem 5.

Our placement of guards in the single slices resembles a construction given in [1], in a slightly different model.

By combining the results of Theorem 5 with those of Theorem 2, we obtain two corollaries.

Corollary 6 Let e denote the number of edges of an orthogonal polyhedron and let g denote its genus. Then $\frac{11}{72}e - \frac{g}{6} - 1$ open edge guards are sufficient to guard the interior of the polyhedron.

Corollary 7 Let r denote the number of reflex edges of an orthogonal polyhedron and let g denote its genus. Then $\frac{7}{12}r-g+1$ open edge guards are sufficient to guard the interior of the polyhedron.

5 Conclusions

We have elucidated the relationship between the required number of closed edge guards and open edge guards. We also improved the current state of the art and obtained a better upper bound $(\frac{11}{72}e$ vs. the previously known $\frac{e}{6}$) on the number of edge guards that suffice for coverage.

We remark that, due to the observation following Theorem 2, our methods do not improve on the $\frac{e}{6}$ upper bound when applied to orthogonal shapes with disconnected boundary. Indeed, in this case the $\frac{e+r}{12}$ given by Theorem 5 still holds, but the r to e ratio can be arbitrarily close to 1.

We conclude with a few possible future directions. The same construction used in Theorem 5 could be analyzed more closely to achieve a tighter upper bound. In contrast with the fact that the polyhedra with highest rto e ratio are responsible for the worst cases in our analysis, such polyhedra are nonetheless intuitively easy to guard by selecting a small fraction of their reflex edges. Isolating these cases and analyzing them separately may yield an improved overall bound.

We also conjecture that suitably placing guards on roughly half of the (open) reflex edges solves our Art Gallery Problem in any orthogonal polyhedron, which would imply that $\frac{1}{2}r + O(1)$ guards suffice (this many are needed in Figure 4 when g = 0).

Observe that our construction in Theorem 5 places guards in just one direction. It would be interesting to investigate this restriction of the Art Gallery Problem (i.e., with the additional constraint that edge guards are mutually parallel), perhaps showing that the lower bound given in Figure 1 can be improved in this more restrictive scenario.

On the other hand, refining our construction by placing guards in all three directions, according to some local properties of the boundary, is likely to yield better upper bounds.

References

- J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia, Illumination of orthogonal polygons with orthogonal floodlights. *Internat. J. Comp. Geom.* 8: 25–38 (1998).
- [2] S. Devadoss and J. O'Rourke. *Discrete and Comptuational Geometry*. Princeton University Press, 2011.
- [3] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

Wireless Localization within Orthogonal Polyhedra

Tobias Christ* Michael Hoffmann*

Abstract

In the wireless localization problem, given a polygon $P \subset \mathbb{R}^2$, we have to place guards and fix their angular range such that P can be *described* using these guards. The guards describe P if for every point pair $p \in P$ and $q \notin P$, there is a guard that sees p but does not see q. We consider the analogous problem in 3D: given a polyhedron $P \subset \mathbb{R}^3$, place guards—which now are polyhedral cones—that collectively describe P. Generalizing a known result for 2-dimensional orthogonal polygons, we show that for any given 3-regular orthogonal polyhedron $P \subset \mathbb{R}^3$ with *n* vertices, it suffices to put a *natural* vertex quard onto every other vertex. (A natural vertex guard is a guard that is placed at a vertex v of Pand the defining cone coincides with P in a sufficiently small neighborhood of v.) Furthermore, we show how to describe P with 3n/8 (general) vertex guards.

1 Introduction

Art gallery problems are a classic topic in discrete and computational geometry. The wireless localization (or sculpture garden) problem, introduced by Eppstein et al. [5], differs from the classical setting in two respects: First, the guards are more powerful because they can "see through walls"; and second, their job is harder, because rather than asking them to collectively *see/cover* the entire polygon (which is very easy, not being impeded by walls), the guards must collectively *describe* the polygon instead.

The motivation for this model stems from communication in wireless networks. For illustration, suppose you run a café (modeled as a polygon P) and you want to provide wireless Internet access. But you do not want the whole neighborhood to use your infrastructure. Instead, Internet access should be limited to those people who are located within the café. To achieve this, you can install a certain number of devices, called guards, each of which broadcasts a unique (secret) key in an arbitrary but fixed angular range. The goal is to place guards and adjust their angles in such a way that everybody who is inside the café can prove this fact just by naming the keys received and nobody who is outside the café can provide such a proof. It is convenient to model a guard as a subset of the plane, namely the area where the broadcast from this guard can be received. This area can be described as an intersection or union of at most two halfplanes. The definition directly carries over to the 3-dimensional case, where a guard is a polyhedral cone, that is, an unbounded polyhedron with at most one vertex and a connected 1-skeleton.

We define a guarding of P to be a set of guards with the property that for each pair of points (p,q), where $p \in P$ and $q \notin P$, there is a guard g that distinguishes p and q, meaning $p \in g$ and $q \notin g$. It can be shown that this notion is equivalent to a description of P using a combination of the operations union and intersection over the guards or—in logical terminology—a monotone Boolean formula over the guards, that is, a formula using the operators AND and OR only, negation is not allowed. (See [2], Observation 1. If each pair (p,q)can be distinguished by a guard, we can find a formula in disjunctive normal form for P: For any point p, let G_p the intersection of all guards that contain p. Then $P = \bigcup_{p \in P} G_p$.)



Natural locations for guards are points on the boundary of P. A guard that is placed at a vertex of P is called a *vertex guard*. A vertex guard on a vertex v is called *natural* if its shape is given by the shape of P at v. More precisely, if the intersection of an ε -ball around v with q equals the intersection with P. A guard placed anywhere on an edge e of P and the shape of which is given by the shape of P at e is called a *natural edge* guard (in 2D, edge guards are just the halfplanes defined by an edge, in 3D they are wedges whose only edge is the line through e). In 3D, there is a third class of natural guards. We call the closed halfspace defined by a face f of P a natural face guard on f. Note that both edge and face guards are cones without apex, so their exact position is undefined. We can think of them to be placed anywhere on their only edge (or anywhere on the bounding plane, respectively).

^{*}Institute of Theoretical Computer Science, ETH Zürich, tobias.christ@inf.ethz.ch, hoffmann@inf.ethz.ch

Dobkin et al. [4] showed that n natural edge guards are sufficient for any simple polygon with n edges. Using both natural vertex guards and natural edge guards, n-2 guards are sufficient and can be necessary [2]; using general vertex guards 8n/9 are sufficient [3]. In the most general setting we do not have any restriction on the placement and the angles of guards. The best known upper bound is $\lfloor \frac{4n-2}{5} \rfloor$ and the best lower bound is $\lceil \frac{3n-4}{5} \rceil$ [2]. We believe that the upper bound can be improved to roughly 3n/4 and the lower bound to 2n/3, but this is still work in progress. The classical art gallery problem for orthogonal polygons was considered in [1].

For the 3D case, Dobkin et al. [4] observed that placing a face guard onto every face of a polyhedron suffices. To our knowledge, the 3-dimensional wireless localization problem has not been studied since then. In this work we focus on orthogonal polyhedra and prove that n/2 natural vertex guards suffice to guard a polyhedron with n vertices. If we allow general vertex guards, we can improve the bound to 3n/8. We observe that there are orthogonal polyhedra that cannot be guarded by fewer than n/4 guards.

2 Notation and Basic Observations

An orthogonal polyhedron P is a polyhedron where all faces are orthogonal to one of the coordinate axes. Faces orthogonal to the x-axis (y-axis, z-axis) are called xfaces (y-faces, z-faces, respectively). Consequently, all edges of P are parallel to one of the coordinate axes and are called x-edges, y-edges and z-edges accordingly. Think of the x-axis as being oriented from left to right, the y-axis front to back, and the z-axis bottom up.

A polyhedron is a solid and closed subset of the space. We define its vertex set V(P), its edge set E(P) and its set of faces F(P) in the usual way. Let n = n(P) = |V(P)| be the number of vertices. Furthermore, we restrict our attention to bounded orthogonal polyhedra with the additional property that exactly three edges meet at every vertex. In other words, the graph of P has to be cubic. Eppstein and Mumford [6] use a similar definition and additionally require the polyhedral surface bounding a polyhedron to have the topology of a sphere. They call this class of polyhedra simple orthogonal polyhedra. We do not use this notation, as in this work, we do not need any topological conditions and allow the polyhedra to form handles (that is, their genus might be greater than 0) and to contain cavities (that is, their surface may be disconnected). Therefore, we use the term 3-regular instead. We define the *type* of a vertex v as follows. Assuming v to be the origin, the type of v is the set of octants P locally occupies around v. We call v convex, if only one octant is inside P. There are eight different possible types of convex vertices. We call v reflex if all but



Figure 1: Vertex types.

one octant around v are in P. There are eight possible types of reflex vertices. Furthermore, there are vertex types where exactly three octants are occupied, so two of the adjacent edges are convex and one is reflex. We call such a vertex *semiconvex*. There are 24 different semiconvex types. Finally, there are vertex types where all but three octants are occupied, denoted as *semireflex*. See Genc [7], p. 38, for a classification of possible vertex types of general orthogonal polyhedra.

By definition guards are unbounded polyhedra with at most one vertex. In this context we restrict ourselves to 3-regular orthogonal guards. From now on, a guard is an unbounded orthogonal cone with at most one x-edge, at most one y-edge and at most one z-edge. Consequently, a guard has at most one x-face, at most one y-face and at most one z-face.

The type of an edge $e \in E(P)$, is given by its direction (parallel to the x-, y-, or z-axis) and by which quadrants around e are occupied by P in the plane orthogonal to e. Either one quadrant is occupied, in which case we call e a convex edge or three quadrants are occupied, in which case we call e a reflex edge. For example, we say an edge e is a convex z-(++) edge if e is vertical and locally around e, the points in P are the points with both higher x- and y-coordinate. Or we say e is a reflex x-(+-) edge if e is parallel to the x-axis and P occupies all but one quadrant around e, namely it leaves out the quadrant that lies behind e (higher y-coordinates) and below e (lower z-coordinates). There are 4 convex and 4 reflex edge types in any of the three directions, so totally, there are 24 different edge types. For each type, we fix a direction of the edge: We define the convex x-(++) edges to be directed in negative x-direction (to the left). Similarly, we define the reflex x-(++) edges to be oriented in positive x-direction (to the right). If we rotate P around the x-axis, an x-(++) edge either becomes a x-(+-) or a x-(-+) edge. So rotating several times around all possible axes, each time by $\pi/2$, an edge can change from any type to any other type. We define the directions of all types in such a way that rotating by $\pi/2$ around any of the three coordinate axes flips the orientation of an edge.

Observation 1 The edges of P can be oriented according to their type such that rotating by $\pi/2$ around any coordinate axis reverses the orientation.


Figure 2: All possible edge types and their orientations.

See Figure 2 for a possible orientation of all edge types. There are exactly two ways to orient the convex types such that the observations holds and, independently, exactly two ways to orient the reflex types such that the observation holds. From now on, we think of every edge to be oriented as shown in Figure 2.

Observation 2 At a vertex $v \in V(P)$ either all adjacent edges are pointing to v or all adjacent edges are pointing away from v.

First consider just one convex vertex type. After observing the property for this type, it follows for all other convex vertex types directly. Repeatedly rotating the vertex by $\pi/2$ around any coordinate axis, we can go from one convex type to any other convex type. With each single rotation, the orientation of all three adjacent edges flip. So if the edges were pointing towards the vertex before, they are all pointing away after the rotation and vice versa, see Figure 3.

With this observation we have reproved that the graph of a 3-regular orthogonal polyhedron is bipartite, as observed for simple orthogonal polyhedra by Eppstein and Mumford [6]. (Their proof is somewhat easier, as it is a direct consequence of the fact that the graph is planar and the numbers of edges of every face is even.)

Corollary 1 The graph of a 3-regular orthogonal polyhedron is bipartite.

In 2D, placing a natural vertex guard onto every other vertex of an orthogonal polygon gives a valid guarding, see [5], Theorem 9, where this is proved for simple



Figure 3: How the orientations of the edges adjacent to a convex and a semiconvex vertex flip when the polyhedron gets rotated by $\pi/2$ around a coordinate axis.

polygons. Because guarding a polygon and guarding its complement are equivalent problems (cf. [2], Observation 5), the same holds for polygons with holes.

Theorem 2 [5] An orthogonal polygon with n vertices (possibly containing holes) can be guarded by n/2 guards placing a natural vertex guard onto every other vertex.

For a face f of a polyhedron, let \overline{f} denote the plane that contains f. A set G of guards covers a point r if there is an $\varepsilon > 0$ such that for any point pair $p, q \in B_{\varepsilon}(r)$ with $p \in P$ and $q \notin P$ there is some guard in G that contains p but does not contain q. We say that a set G of guards covers a face $f \in F(P)$, if G covers some point p in the interior of f. (p is in the interior of f if $p \in f$ and $p \notin e$ for any $e \in E(P)$.) and G covers fcompletely, if G covers all points in the interior of f. If $\{g\}$ covers f, then g has a face f_q with $\overline{f_q} = \overline{f}$.

Theorem 3 For any integer $k \ge 2$, there are 3-regular orthogonal polyhedra with 4k vertices that cannot be guarded by fewer than k guards.

Proof. Take a 2-dimensional orthogonal polygon Q with 2k pairwise non-collinear edges. Let P be a (right) prism with base Q. P has 2k + 2 faces, k of which are x-faces. Consider a guarding \mathcal{G} of P. Each face f of P has to be covered by at least one guard. No two x-faces are coplanar, so any guard can cover at most one x-face. Therefore, there are at least k guards in \mathcal{G} .

3 Guarding with n/2 Natural Vertex Guards in 3D

Theorem 4 A 3-regular orthogonal polyhedron P can be guarded with n(P)/2 natural vertex guards.

Proof. Place a guard onto every vertex where all edges are pointing inwards. We show that for every inside/outside point pair (p,q) there is a guard g that distinguishes p and q. Denote the axis-parallel cube spanned by $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ by Q.



Figure 4: The three subcases of Case 1: Vertex v can have any of the types to the right.

Edge orientations and hence guardings are symmetric under a rotation by an angle of π around a coordinate axis. Therefore we may suppose without loss of generality that **either** $q_x \leq p_x$, $q_y \leq p_y$ and $q_z \leq p_z$, or $p_x \leq q_x$, $p_y \leq q_y$ and $p_z \leq q_z$.

First consider the case that $q_x \leq p_x$, $q_y \leq p_y$ and $q_z \leq$ p_z . Look at $Q \cap P$ and pick the point $r = (r_x, r_y, r_z) \in$ $P \cap Q$ which minimizes $r_x + r_y + r_z$. The point r can arise in three different ways, as depicted in Figure 4: If $r \in V(P)$, r is a convex vertex such that P occupies the (+++)-octant. Therefore, there is a guard on it which distinguishes p and q. If $r = e \cap f$ is the intersection of an edge $e \in E(P)$ and a face f of Q, f must be adjacent to q and *e* must be a convex edge orthogonal to it. Therefore, e is pointing outward of Q to a guard g distinguishing p and q. If r is on an edge of Q and in the interior of a face f of P, look at f as a 2D-polygon: Drawing a horizontal and a vertical line through r divides f into four quadrants. f has a convex vertex in each quadrant, in particular in the quadrant opposite to $f \cap Q$. No matter which type v has as a vertex of P, it is going to distinguish p and q.

In the case where $p_x \leq q_x$, $p_y \leq q_y$ and $p_z \leq q_z$, we have to use a slightly different argument. Let Abe the face of Q that is adjacent to p and orthogonal to the z-axis. Consider the 2-dimensional orthogonal polygon P' we get by intersecting P with the plane \overline{A} and picking the connected component of the intersection that contains p. Let r be the point in $P' \cap A$ that maximizes $r_x + r_y$. As in the first case, there are three sub-cases to consider. If $r = A \cap e$, $e \in E(P)$, we observe that e is a convex z-edge of type (--), so it is oriented



Figure 5: Case 2: Following e we find a guard g, which has one of several possible types.

downwards. So we can follow e to its end point outside Q where we find a guard q that distinguishes p and q, see Figure 5. If r is the intersection of an edge of A with a face f of P, then r divides f into four quadrants, in each of which we find a convex vertex of f (thought of as 2D-polygon). In particular, there is a convex vertex v of f in the quadrant opposite to the one containing $f \cap Q$ and there is a guard on v that distinguishes p and q. Finally, if r is a vertex of A, which means that A is completely contained in P, then pick another face B of Q adjacent to p, and repeat the argument. If all faces adjacent to p are completely inside P, then look at the top face C of Q and observe that $C \cap P$ must have at least one reflex vertex r. This vertex r lies on a reflex z-edge of P, which is pointing upward to a guard outside of Q that distinguishes p and q, see the example to the right in Figure 5. \square

4 Improving the Bound to 3n/8

Let P be an arbitrary 3-regular orthogonal polyhedron with n vertices. Every vertex is incident to one x-edge, one y-edge and one z-edge. Thus, there are exactly n/2edges in each direction. In the guarding described above we used one guard per z-edge. In order to reduce the number of guards, we now place a natural edge guard onto roughly half of the z-edges only, namely onto those that are pointing downwards.

Such a guarding for sure covers all x- and y-faces, but we have not done anything about the z-faces yet. An easy solution would be to place a natural face guard g_f onto every z-face f. This would yield a valid guarding, but the number of z-faces could be as large as n/4(even after permuting the coordinate axes). So instead, we replace every natural edge guard g_e on a z-edge epointing down to a vertex v of f by a vertex guard $g_v: g_v := g_e \cap g_f$ if f has the interior of P above and $g_v := g_e \cup g_f$ if f has the interior of P below. So in some sense—we combine the natural edge guards



Figure 6: An increasing event.

and the natural face guards to vertex guards. (Note that these new guards are not necessarily natural vertex guards.) However, some z-faces may not be covered still. We call a z-face f good, if the guards we place on vertices of f cover f completely, and we call f bad, otherwise. If a z-face f is bad, then we put a natural face guard g_f onto f. So we have to make sure that this does not happen too often and that we will not use more than roughly n/8 face guards in this way. Let \mathcal{G} be the set of vertex guards on bad z-faces.

Lemma 5 \mathcal{G} is a valid guarding of P.

Proof. We use a sweep argument. For simplicity, we assume that P has no coplanar z-faces. Imagine sweeping a plane E orthogonal to the z-axis upwards and look at the *intersection polygon* $Q = P \cap E$. Whenever E is coplanar with a z-face f of P (called the *event face*), the intersection polygon Q changes: the new intersection polygon Q' is either bigger or smaller, $Q' = Q \cup f$ or Q' is (the closure of) $Q \setminus f$. We call the first case an *increasing event*, the second case a *decreasing event*.

The claim is that using guards encountered so far only, we are able to guard P as far as we have seen it: At any moment the set $\tilde{\mathcal{G}}$ of guards that lie below the sweep plane E, together with an imaginary face guard g_E that is the closed halfspace below E, is a guarding of the part $\tilde{P} = P \cap g_E$ of P below E. We prove this claim by induction on the number of event faces processed.

At the beginning \tilde{P} is empty. At some point, we hit the first z-face f of P. The vertices of f correspond to zedges of P starting at f and going upwards. According to our rule, there is a guard on every second vertex of f, which we can think of as a 2-dimensional guarding for f extending to the region orthogonally above f.

Increasing Event. Let f be an increasing event face, that is, the interior of P lies above f (Figure 6). We claim that after E has passed f (but no other event face yet), we still have a valid guarding for \tilde{P} . Consider a point pair $p \in \tilde{P}$ and $q \notin \tilde{P}$. We may suppose without

loss of generality that both p and q lie in the closed halfspace below \overline{f} : If one of the points, say, p lies above \overline{f} , then consider the orthogonal projection p' of p onto \overline{f} instead. As all guards in $\tilde{\mathcal{G}}$ are located in the closed halfspace below \overline{f} , none of them distinguishes p and p'.

If p and q are both below \overline{f} , then by induction there is a guard that distinguishes them. If both p and q lie in \overline{f} , then we are in a 2-dimensional situation and find a guard that distinguishes them because our guarding contains a 2-dimensional guarding of Q'. (The vertices of Q' correspond to z-edges. There is a guard on every other z-edge that—intersected with $\overline{Q'}$ —is a natural 2D vertex guard of Q'. See Theorem 2.)

If $p \in f$ and q lies below, then either there is a face guard g_f that does the job or f is a good face. In the latter case, the new vertex guards (i.e., those placed when handling the event face f) collectively cover f. If $p \in f$, then one of these new guards distinguishes p and q. Otherwise, a point p' slightly below p lies within P as well. By induction there is some old guard (i.e., a guard placed before handling the event face f) to distinguish p' from q. As such a guard cannot distinguish between p and p', it also distinguishes p and q. Symmetrically, if $q \in \overline{f}$ and p is below, consider a point q' located slightly below q and note that $q' \notin P$ because the event is increasing. By induction, there is an old guard that distinguishes p and q' but cannot distinguish q and q'. Hence this guard also distinguishes p and q.

Decreasing Event. Consider a point pair $p \in \tilde{P}$ and $q \notin P$. As above, we may suppose without loss of generality that p lies in the closed halfspace below f. However, if q lies above the event face f—that is, the orthogonal projection q' of q onto \overline{f} lies in f—we cannot simply replace q by q', because $q' \in P$. But we know that some guard that was placed when handling the event face f distinguishes q and q', and every guard placed when handling f contains the closed halfspace below \overline{f} . Therefore, if p lies below \overline{f} , then this guard distinguishes p and q. If $p \in \overline{f}$, then recall that we have a 2-dimensional guarding for Q', which must contain a guard that can distinguish p and q'. This guard classifies q' as outside and so it does with q. Hence it distinguishes p and q. We may thus assume that q lies in the closed halfspace below f as well. It follows inductively that there exists an old guard that distinguishes p and q. \square

Lemma 6 Under a random rotation around the z-axis by a multiple of $\pi/2$ and independently, a reflection with respect to the plane z = 0 with probability 1/2, a z-face with 4 or 6 vertices is good with probability at least 1/2.

Proof. Each vertex v of f corresponds to a z-edge e_v of P. e_v either starts at v going upwards or it ends at v. We call v a starting vertex or an ending vertex, respectively. If v is a starting vertex and e_v is pointing

to v, there is a guard g_v on v. Else, if e_v is oriented upwards or v is an ending vertex, there is no guard on v. Under a reflection in the xy-plane, starting vertices turn into ending vertices and vice versa, see Figure 7.

Consider f as a 2D-polygon. If it has 6 vertices, exactly 5 are convex and one is reflex. One of the convex 2D-vertex-types appears twice, the other three appear exactly once and are referred to as *unique*, therefore. If f has 4 vertices, they are all unique. So in any case we have at least three unique (convex) vertices. Moreover, these vertices appear consecutively along the boundary of f, which implies that for at least one of them the incident z-edge is directed towards the vertex.

If there is a guard on some unique vertex of f, then fis good because this guard covers f completely. (Note that f may be bad if there is a guard at some nonunique vertex only, see Figure 7 (B).) If two adjacent unique vertices of f are starting, then—by the remark above—at least one of them has a guard. When reflecting P at the plane z = 0, all ending vertices turn into starting vertices and vice versa. Hence, if two adjacent unique vertices of f are ending, then at least of them has a guard after this reflection and so f is good with probability at least 1/2. It remains to consider the case that the three unique vertices of f follow the pattern starting-ending-starting or ending-starting-ending and neither of the starting vertices, with and without reflection, has a guard (Figure 7 (C)). When rotating around the z-axis by $\pi/2$ or $3\pi/2$, starting vertices remain starting and ending vertices remain ending. But edge orientations flip and so a starting vertex without guard turns into a starting vertex with guard. As a result, both the original face and the reflected variant turn good after such a rotation. So again with probability at least 1/2, the face f appears as a good face.

Theorem 7 Let P be a 3-regular orthogonal polyhedron. Then P can be guarded with 3n(P)/8 guards.

Proof. Guard P as described at the beginning of the section. Whenever we have to use a face guard g_f , we charge it to the edges of f. The edges of a z-face f are also edges of P and each x- or y-edge appears exactly once as an edge of a z-face. So the edges of a z-face f of degree $d \ge 8$ each get charged at most $1/d \le 1/8$. The edges of a face f with degree d = 4 or d = 6 get charged $1/d \leq 1/4$ if we have to use a face guard g_f . If we pick a random rotation around the z-axis and independently decide to reflect P with respect to the plane z = 0 with probability 1/2, we have shown that f ends up as a bad face with probability at most 1/2. So the x- or y-edges gets charged at most 1/4 with probability at most 1/2 and 0 otherwise. So the expected charge of an x- or y-edge is at most 1/8. The vertex guards get charged to their corresponding z-edge. Under a random



Figure 7: (A) a bad face that turns good after reflection; (B) a face that is bad even though there is a vertex v with a guard g_v on it: g_v does not cover the face completely; (C) a face that stays bad after reflecting, but both itself and the reflected version turn good after rotating around the z-axis by $\pi/2$.

rotation, a z-edge gets charged 1 with probability 1/2and 0 otherwise. Therefore, the expected total charge is going to be at most $\frac{1}{8}n + \frac{1}{2}n/2 = 3n/8$, so there is a rotation (possibly combined with a reflection) such that at most 3n/8 guards are used.

References

- J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia. Illumination of orthogonal polygons with orthogonal floodlights. *Internat. J. Comput. Geom. Appl.*, 8(1):25– 38, 1998.
- [2] T. Christ, M. Hoffmann, Y. Okamoto, and T. Uno. Improved bounds for wireless localization. *Algorithmica*, 57:499–516, July 2010.
- [3] T. Christ and A. Mishra. Wireless localization with vertex guards. In Abstracts of the 27th European Workshop on Computational Geometry (EuroCG '11), Morschach, Switzerland, March 2011.
- [4] D. P. Dobkin, L. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Algorithmica*, 10:1– 23, 1993.
- [5] D. Eppstein, M. T. Goodrich, and N. Sitchinava. Guard placement for efficient point-in-polygon proofs. In Proc. 23rd Annu. Sympos. Comput. Geom., pages 27–36, 2007.
- [6] D. Eppstein and E. Mumford. Steinitz theorems for orthogonal polyhedra. In Proc. 26th Annu. Sympos. Comput. Geom., pages 429–438, 2010.
- [7] B. Genc. Reconstruction of Orthogonal Polyhedra. PhD thesis, University of Waterloo, 2008.

Weak Visibility Queries in Simple Polygons

Mojtaba Nouri Bygi *

Mohammad Ghodsi[†]

Abstract

In this paper, we consider the problem of computing the weak visibility (WV) of a query line segment inside a simple polygon. Our algorithm first preprocesses the polygon and creates data structures from which any WV query is answered efficiently in an output sensitive manner. In our solution, the preprocessing is performed in time $O(n^3 \log n)$ and the size of the constructed data structure is $O(n^3)$. It is then possible to report the WVpolygon of any query line segment in time $O(\log n + k)$, where k is the size of the output. Our algorithm improves the current results for this problem.

1 Introduction

Two points inside a polygon are visible to each other if their connecting segment remains completely inside the polygon. The visibility polygon VP(q) of a point qin a simple polygon P is the set of P points that are visible from q. A common approach to this problem is to decompose the polygon into the visibility regions in such a way that all points inside a region have equivalent visibility data [2]. Two visibility polygons are equivalent if they are composed of the same sequence of vertices and edges of the underlying polygon. If all the visibility regions and their corresponding visibility polygons are calculated in the preprocessing phase, for any point q, VP(q) can then be obtained by refining the visibility polygon of the region that contains q.

In a simple polygon with n vertices, VP(q) can be reported in time $O(\log n + |VP(q)|)$ by spending $O(n^3 \log n)$ preprocessing time and $O(n^3)$ space [2, 7]. An improvement was presented in [1] where the preprocessing time and space were reduced to $O(n^2 \log n)$ and $O(n^2)$ respectively, at the expense of more query time of $O(\log^2 n + |VP(q)|)$.

The visibility problem has also been considered for line segments. A point v is said to be *weakly visible* to a line segment pq if there exists a point $w \in pq$ such that w and v are visible to each other. The problem of computing the *weak visibility polygon* (or WVP) of pq inside a polygon P is to compute all points of P that are weakly visible from pq. If P is a polygon without holes, Chazelle and Guibas [3] gave an $O(n \log n)$ time algorithm for this problem. Guibas *et al.* [6] showed that this problem can be solved in O(n) time if a triangulation of P is given along with P. Since P can be triangulated in O(n) [4], the algorithm of Guibas *et al.* runs in O(n) time [6]. Another linear time solution was obtained independently in [8].

The weak visibility problem in the query version has been considered by few. It is shown in [2] that a simple polygon P can be preprocessed in $O(n^3 \log n)$ time and $O(n^3)$ space such that given an arbitrary query line segment inside the polygon, $O(k \log n)$ time is required to recover k weakly visible vertices. This result was later improved in [1] where the preprocessing time and space were reduced to $O(n^2 \log n)$ and $O(n^2)$ respectively, at the expense of more query time of $O(k \log^2 n)$.

In this paper, we improve these results by showing that the weak visibility polygon of a line segment pq can be reported in an output sensitive time of $O(\log^2 n + k)$ after preprocessing the input in time and space of $O(n^3 \log n)$ and $O(n^3)$ respectively.

2 Preliminaries

In this section we introduce some basic terminologies used throughout the paper. For a better introduction to these terms, we refer the readers to Guibas *et al.* [6], Bose *et al.* [2], and Aronov *et al.* [1]. For simplicity, we assume that no three vertices of the polygon are collinear.

2.1 Visibility Decomposition

Let P be a simple polygon with n vertices. Also let p and q be two points in the polygons. A visibility decomposition of P is to partition P into a set of visibility regions, such that for each region, the same sequence of vertices and edges of P are visible from any point inside the region.

Two visibility regions are *neighboring* if they are separated by an edge. In a simple polygon, two neighboring visibility regions differ only in one vertex in their visibility sequences. This fact is used to reduce the space complexity of maintaining the visibility sequences of the regions [2]. This is done by defining the *sink regions*. A sink is a region with the smallest visibility sequence

^{*}Department of Computer Engineering, Sharif University of Technology, nouribaygi@ce.sharif.edu

[†]Computer Engineering Department, Sharif University of Technology, and Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. ghodsi@sharif.edu. This author's research was partially supported by the IPM under grant No: CS1389-2-01

compared to all of its adjacent regions. It is therefore sufficient to only maintain the visibility sequences of the sinks, from which the visibility sequences of all other regions can be computed. By constructing a directed dual graph (see Figure 1) over the visibility regions, one can maintain the difference between visibility sequences of neighboring regions[2].



Figure 1: Decomposed visibility regions and its dual graph [2].

The number of visibility regions in a simple polygon is $O(n^3)$, and the number of sink regions is $O(n^2)$ [2].

2.2 Linear time algorithm for computing WVP

Here, we explain the O(n) time algorithm of Guibas et al. [6] for computing WVP(pq) of a line segment pqinside a simple polygon P with n vertices, as described in [5]. For any line segment pq, we can cut P into two polygons P_1 and P_2 along the supporting line of pq (see Figure 2). It can be seen that WVP(pq) is the union of the WVPs of the two sub-polygons from pq. So here we assume that pq is an edge of P.



Figure 2: P is divided by uv into two sub-polygons.

Let SPT(p) denote the shortest path tree in P rooted at p. We traverse SPT(p) using a depth-first search and check the turn at every vertex v_i in SPT(p). If the path $SP(p, v_j)$ makes a right turn at v_i , then, we find the descendant of v_i in the tree with the largest index j (see Figure 3). We compute the intersection point z of $v_j v_{j+1}$ and $v_k v_i$, where v_k is the parent of v_i in SPT(p), in O(1) (because there is no vertex between v_j and v_{j+1}), and finally remove the counter-clockwise boundary of P from v_i to z by inserting the segment $v_i z$.

Let P' denote the remaining portion of P. We follow the same procedure for q, except that this time we check the turn at every vertex and see whether the path make its first left turn. After finishing the procedure, we output the remaining portion of P' as WVP(pq).



Figure 3: In both cases, the from the root makes its first right turn at v_j [5].

3 The Proposed Algorithm

In this section, we show how to modify the presented algorithm, so that the WVP can be computed efficiently in an output sensitive manner. First, we show how to compute the shortest path trees in an output sensitive manner, and then we present the first version of our algorithm. Finally, in Section 3.3, we improve this algorithm and present the final result.

3.1 Computing the shortest path trees

In our algorithm, we use both of the shortest path trees of p and q. In [6], it is shown how to compute the Euclidean shortest paths inside a simple polygon P of n vertices from a given point p to all other vertices in O(n) time. But, this algorithm requires O(n) of query time which is way beyond our goal. To overcome, we show how to preprocess a simple polygon, so that for any given point, we can compute any part of its shortest path tree in an output sensitive way.

The shortest path tree SPT(p) is composed of two kinds of edges: the primary edges, which are from the root p to its direct visible vertices, and the secondary edges that connect other two vertices of polygons (see Figure 4).

We can compute the primary edges using the same output sensitive algorithm of computing the visibility polygon [2]. More precisely, with a processing cost of $O(n^3 \log n)$ time and $O(n^3)$ space, we can in query time of $O(\log n)$ have a pointer to the sorted list of the visible vertices from p in $O(\log n)$ time.



Figure 4: The shortest path tree from p and its different edge types.

We also need to access the list of the secondary edges of a node in constant time. For this, we compute all possible values of secondary edges of a vertex in preprocessing time and in query time, we detect the appropriate list without any further cost.

Depending on the number of possible parents of a vertex v, we recognize two kinds of secondary edges: The 1st type of secondary edges (1st type for short) are those connected to a primary edge, and the 2nd type are the ones that connect other two vertices of the polygon.

For the 2nd type edges, as there are O(n) possible parents for v, and for each parent, there may be O(n)edges emitting from v, we need $O(n^2)$ space to store all possible combinations of the 2nd type edges emitting from v. In total, we need $O(n^3)$ space to store all these edges. We can also compute a local shortest path tree in each case (a *SPT* with the parent of v as its root), and compute the sorted list of edges in $O(n \log n)$, or in total $O(n^3 \log n)$ time.

The parent of a 1st type edge is the root of the tree. As the root can be in any of the $O(n^3)$ different visibility regions, computing all possible combinations of the 1st type edges emitting from a vertex, requires to consider all these parents (remember that if the root of two *SPTs* are in the same region, then the combinatorial structure of the two trees are the same). We can compute the first type edges for each region in $O(n^4 \log n)$ time and store them in $O(n^4)$ space. In Section 3.3 we will show how to improve this result by a linear factor.

Theorem 1 Given a simple polygon P, we can preprocess it into a data structure with $O(n^4)$ space and in $O(n^4 \log n)$ time so that for any query point p, the shortest path tree from p can be reported in $O(\log n + k)$, where k is the size of the tree that is to be reported.

Proof. First, we use Bose's algorithm for computing the visibility polygon of point p. For this, we need $O(n^3)$ space and $O(n^3 \log n)$ time in the preprocessing phase.

For the secondary edges, we need $O(n^4 \log n)$ time and $O(n^4)$ space to compute and store the 1st type edges, and $O(n^3 \log n)$ time and $O(n^3)$ space to store the 2nd type ones.

In query time, we can locate the visibility region of p in $O(\log n)$ and have the sorted list of the visible vertices from p. Therefore, we can use the primary edges of SPT(p) without paying any further costs (remember that each visible vertex from p corresponds to a primary edge in SPT).

As we have computed the 1st type edges of the SPT for all the regions, we can access a pointer to the sorted list of these edges in O(1). Similarly, at any node of the tree, we have the list 2nd type edges from that node. Therefore, the cost of traversing the SPT would be the number of visited nodes of the tree, plus the initial $O(\log n)$ cost, i.e., $O(\log n + k)$, where k is the number of the traversed edges of SPT.

3.2 Computing the query version of *WVP*

In this section, we use the linear algorithm of Guibas et al. [6] for computing WVP of a simple polygon and show how to compute the query version of this problem. We build the data structure explained in previous section, so that we can compute the SPT of any point inside the polygon in query time.

This algorithm is not output sensitive by itself. See the example of Figure 5. As stated in Section 2.2, first we traverse SPT(p) using DFS and check the turn at every vertex of SPT(p). Consider vertex v. As we traverse the shortest path from p to v, or SP(p,v), we must check all the children of v and this checking can costs O(n). But when we traverse SPT(q), v would be omitted, therefore, the time we spend for processing its children would be useless.



Figure 5: Processing vertex v imposes redundant O(n) time.

To achieve an output sensitive algorithm, we store some additional information about the vertices of the polygon. We say that a vertex v of a simple polygon is *left critical* (LC for short) with respect to a point p, if SP(p, v)makes its first left turn at v or one of its ancestors. In other words, each shortest path from p to a non-LC vertex is a convex chain that makes only clockwise turns at each node. Having the critical state of all vertices with respect to a point p, we say that we have the *critical information* with respect to p.

Having the critical information of p and q, we can change the algorithm for computing WVP as follows: In the first round, we traverse SPT(p) using DFS. At each vertex, we check whether this vertex is left critical with respect to q. If so, we are sure that the descendants of this vertex are not visible from pq, so we postpone its processing to the time we reach it from q, and check the other branches of SPT(p). Otherwise, we proceed with the algorithm and check whether SPT(p) makes a right turn at this vertex. In the second round, we traverse SPT(q) and perform the normal procedure of the algorithm.

Lemma 2 All the vertices that we traverse in SPT(p)and SPT(q) are vertices of WVP(pq).

Proof. Assume that we meet v when we are traversing SPT(p) and $v \notin WVP(pq)$. Also, assume that u is the parent of v in SP(pv). Then, u or one of its ancestors must be LC with respect to q, otherwise the Guibas *et al.* algorithm will detect it as a WVP vertex. So, as one of the ancestors of v is LC, we would not reach v when traversing SPT(p). The same argument applies to SPT(q).

As the combinatorial structure of shortest path trees of all points in a visibility region are the same, we just need to compute the critical information of a point ain each region S, and use this information for all points of that region. In the preprocessing phase, for each visibility region we compute critical information of a point inside it, and assign this information to that region. In query time and upon receiving a line segment pq, we locate p and q. Using the critical information of their regions, we apply the above algorithm and compute WVP(pq).

So far, we have assumed that pq is a polygon edge. The following lemma generalizes the position of pq in P.

Lemma 3 If pq is a line segment inside the simple polygon P, we can decompose P into two sub-polygons P_1 and P_2 , such that they both have pq as an edge. In addition, we can use the critical information and the secondary edges data of the visibility regions of P for these sub-polygons.

Proof. We build the ray shooting structure in P, in O(n) time and space [3]. In query time, we find the

intersection points of the supporting line of pq with the border of P. Locating these intersection points among the vertices of P, we can create two simple polygons, P_1 and P_2 , in $O(\log n)$ time. Each of these two polygons has pq on its edge. As the visibility regions of the generated polygons are a subset of the visibility regions of the original polygon, and we have computed the critical information and the SPT edges for all the regions of P, we have the needed data for p and q in both P_1 and P_2 . See an example in Figure 6.



Figure 6: If the query line segment pq is inside the polygon, we split it along the supporting line of pq.

For simplicity, we can translocate pq a little higher (or lower) from its supporting line to p'q'. As the visibility regions of p and p' (also q and q') are the same, WVP(pq) and WVP(p'q') have the same combinatorial structures. We need to filter the primary edges originating from p and q to those that are in P_1 (or P_2). As we have the sorted list of these edges, this filtering can be done in $O(\log n)$ by a simple range searching. By traversing these primary edges at each vertex of P_1 , we can use the stored critical information and secondary edges of that vertex. Depending on which side of the line pq we are on, we use the critical information of p or q in the weak visibility computations.

Now, we analyse the time and space of the above algorithm. As there are $O(n^3)$ visibility regions, we need $O(n^4)$ space to store the critical information of each vertex. For each region, we compute SPT of a point, and by traversing the tree, we update the critical information of each vertex with respect to this region. We assign an array of size O(n) to each region to store these information. We also build the structure described in Section 3.1 for computing SPT in time $O(n^4 \log n)$ and $O(n^4)$ space. In query time, we locate the visibility regions of p and q in $O(\log n)$. As we traverse SPTs of p and q, by Lemma 2, each vertex that we see is on WVP(pq). Because the processing time we spend in each vertex is O(1), the total query time is $O(\log n + |WVP(pq)|)$.

Theorem 4 Using $O(n^4 \log n)$ time to preprocess a simple polygon P and maintain a data structure of size $O(n^4)$, it is possible to report WVP(pq) in time $O(\log n + |WVP(pq)|)$.

3.3 Improving the algorithm

To improve the result of Theorem 4, we will modify two parts of our algorithm. First, we show that it is sufficient to compute the critical information of the sink visibility regions (see Section 2.1), from which we can deduce the critical information of all other regions. Also, in computing *SPT* in Section 3.1, we will show that if we compute 1st type of secondary edges of the sink regions, we can compute these edges for the non-sink regions in query time. As there are $O(n^2)$ sinks in a simple polygon, the processing time and space of our algorithm would reduce to $O(n^3 \log n)$ and $O(n^3)$ respectively.

In query time, if both p and q belong to sink regions, we have critical information of both regions and we proceed the algorithm as stated before. On the other hand, if one of these points lie on a non-sink region, we show how to obtain the secondary edges and the critical information for that region in $O(\log n + |WVP(pq)|)$.

Lemma 5 Consider two visibility regions that share a common edge. If we have the 1st type secondary edges of a region for each vertex visible from it, these edges are the same for its neighboring region, except for one edge.



Figure 7: Combinatorial changes of *SPT* by moving between neighboring regions.

Proof. When we cross the border of two neighboring regions, a vertex becomes visible, or invisible [2]. In Figure 7 for example, when p crosses the border specified by u and v, a 1st type secondary edge of u becomes a primary edge of p, and all edges of v become 1st type

secondary edges. We can see that no other vertex would be affected by this movement. Processing these changes can be done in constant time, since it includes the following changes: removing a secondary edge of u (uv), adding a primary edge (pv) and moving an array pointer (edges of v) from 2nd type edges to 1st type edges. Note that we know the exact position of these elements, so we do not have the overhead time of finding them in their corresponding lists. Finally, we can identify the sole edge which involves with these changes in the preprocessing time (the edge corresponding to the crossed critical constraint), so, the time we spend in the query time would be O(1).

Lemma 6 In the path from a sink to another visibility region, we can handle the changes of the critical information of the point in constant time.



Figure 8: The critical information of v w.r.t p, as p moves between the two regions. a) v is LC but not u, b) u and v are not LC, c) both u and v are LC, d) u is LC but not v.

Proof. Suppose that we want to maintain the critical information of p and we are crossing the critical constraint defined by the edge uv. Depending on the critical status of u and v w.r.t. p, four possible situations may occur (see Figure 8). In the first three cases, the critical status of v will not change and no further action is required. In the forth case, however, the critical status of u will change. To handle this case, we modify the way we store the critical status of each vertex w.r.t. p. More precisely, at each vertex v we store the number of LC vertices we met, or *critical numbers*, in

the path SP(p, v) (see Figure 9). Computing and storing the critical numbers along the critical info will not change our time and space requirements. Now consider



Figure 9: We store the number of LC vertices we met from p in SPT(p).

the forth case in Figure 8. When v becomes visible to p, it is no longer LC w.r.t. p. So, we change the critical number of v to 0, but instead of changing the critical numbers of its children, we store -1 in v as its critical number, indicating that the critical numbers of all the vertices of its subtree must be subtracted by 1. The actual propagation of this subtraction will happen when we are traversing SPT(p). We also modify the query time algorithm to reflect this change. If we are computing WVP(pq), and because v is LC w.r.t. q, we stopped at the path SP(p, v), we store a pointer to this path at v. When we are traversing SP(q, v) and we find out that v is no LC w.r.t. q, we resume the stored pass.

In the preprocessing time, we construct the dual planar graph of the visibility regions. We use the dual directed graph that was built by algorithm of Bose *et al.* [2] (Figure 1). In this graph, every node represents a visibility region, and an edge between two nodes corresponds to a gain of one vertex in the visibility set in one direction, and a loss in the other. By Lemma 5 and 6, we also know that these two neighboring regions have the same critical information and secondary edges, except for one vertex. We associate this vertex with the edge. We also compute the critical information and 1st type secondary edges of all the sink regions.

In query time, we locate the region containing point p, and follow any path from this region to a sink. As each arc represents one vertex seen by the query point p and therefore seen by pq, the number of arcs that we pass would be O(|WVP(pq)|). When traversing the path from sink back to the region of p, we update the critical information and the secondary edges of the visible vertices in each region. Upon coming back to the original region, we would have the critical information and the secondary edges of this region. We perform the same procedure for q. Having the critical information and the 1st type edges of p and q, we can compute WVP(pq) with the algorithm of Section 3.2. Putting all together, we have the following result

Theorem 7 A simple polygon P can be preprocessed in $O(n^3 \log n)$ time and $O(n^3)$ space such that given an arbitrary query line segment inside the polygon, it takes $O(\log n + |WVP(pq)|)$ time to list all vertices of WVP(pq).

4 Conclusion

In this paper, we showed how to answer weak visibility queries in a simple polygon in an efficient way. We presented an algorithm to report WVP(pq) of any line segment pq in $O(\log n + |WVP(pq)|)$ time by spending $O(n^3 \log n)$ time to preprocess the polygon and maintaining a data structure of size $O(n^3)$.

Currently, we are working on a different approach for the same problem, to construct a data structure of size $O(n^2)$ which can be computed in time $O(n^2 \log n)$ so that the weak visibility polygon WVP(pq) from any query line segment $pq \in P$ can be reported in $O(\log^2 n + |WVP(pq)|)$ time. Also, we are investigating whether our techniques can be extended to the cases of polygons with holes.

Acknowledgement

The authors would like to thank the anonymous reviewers for their helpful comments.

References

- B. Aronov, L. Guibas, M. Teichmann and L. Zhang. Visibility queries and maintenance in simple polygons. *Discrete and Computational Geometry*, 27(4):461-483, 2002.
- [2] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. *Computational Geometry: Theory and Applications*, 23(3):313-335, 2002.
- [3] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. Discrete and Computational Geometry, 4:551-581, 1989.
- B. Chazelle. Triangulating a simple polygon in linear time. Discrete and Computational Geometry, 6:485-524, 1991.
- [5] S. K. Ghosh. Visibility Algorithms in the Plane. Cambridge University Press, New York, NY, USA, 2007.
- [6] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209-233, 1987.
- [7] L. Guibas, R. Motwani, and P. Raghavan. The robot localization problem in two dimensions. SIAM J. Comput., 26(4):11201138, 1997.
- [8] G. T. Toussaint. A linear-time algorithm for solving the strong hidden-line problem in a simple polygon. *Pattern Recognition Letters*, 4:449-451, 1986.

The Possible Hull of Imprecise Points*

William Evans[†]

```
Jeff Sember<sup>†</sup>
```

Abstract

We pose the problem of constructing the possible hull of a set of n imprecise points: the union of convex hulls of all sets of n points, where each point is constrained to lie within a particular region of the plane. We give an optimal algorithm for the case when n = 2, and the regions are a point and a simple (possibly nonconvex) polygon. We then describe how the algorithm leads to an optimal algorithm for the case when $n \ge 2$, and each region is a simple polygon.¹

1 Introduction

Let $S = \{s_1, \ldots, s_n\}$ be a planar point set.² If we are not given the locations of these points, but are told only that each point s_i lies within a particular region of uncertainty R_i , then the points are *imprecise*. The convex hull of a set of imprecise points cannot be determined since it is one of possibly many *feasible hulls* (each of which is the convex hull of a *feasible* set $\{s_1 \in R_1, \ldots, s_n \in R_n\}$).

Kreveld and Löffler investigate the problem of finding the feasible hull with maximal or minimal area or boundary length [3]. The problem of determining the intersection of all feasible hulls has also been investigated [5], [6], [2], [1], [7].

We define the *possible hull* of a set of imprecise points (or their corresponding regions of uncertainty) as being the union of the feasible hulls of the points. To motivate this problem, consider the scenario where each island in a group of islands contains a sensor whose exact location is uncertain, and that each pair of these sensors can detect any object that passes between them. To avoid being detected, a boat traveling near the islands would need to remain outside of their possible hull.

One reason that possible hulls have received little attention until now is that when the regions of uncertainty are convex, the possible hull is simply the convex hull of the regions [5]. In this paper, we investigate possible hulls of more general uncertain regions. We present an algorithm for constructing the possible hull of a point and a simple (possibly nonconvex) polygon, and describe how this algorithm can be used as a subroutine to construct the possible hull of two or more simple polygons. See Figure 1.



Figure 1: Possible hulls of pairs of uncertain regions.

2 Properties

We will denote the convex hull of a point set S by CH(S), and the possible hull of uncertain regions $\mathcal{R} = \{R_1, \ldots, R_n\}$ by $PH(\mathcal{R})$ (or, when clear from the context, by PH). Formally,

$$PH(\mathcal{R}) = \bigcup_{\{s_1 \in R_1, \dots, s_n \in R_n\}} CH(\{s_1, \dots, s_n\}).$$

From this definition, we can derive the following additional properties of possible hulls.

Lemma 1
$$PH(\{A\}) = A$$

Lemma 2 $PH(\{A, B\}) = \bigcup_{a \in A, b \in B} \overline{ab}.$

Lemma 3
$$\bigcup_{R_i \in \mathcal{R}} R_i \subseteq PH(\mathcal{R}).$$

Lemma 4 If \mathcal{A} and \mathcal{B} are nonempty sets of uncertain regions, then $PH(\mathcal{A} \cup \mathcal{B}) = PH(\{PH(\mathcal{A}), PH(\mathcal{B})\}).$

Proof. Let $Q = PH(\{PH(\mathcal{A}), PH(\mathcal{B})\})$. Suppose p is a point within $PH(\mathcal{A} \cup \mathcal{B})$. Then there exists a feasible set S of $\mathcal{A} \cup \mathcal{B}$ such that $p \in CH(S)$. Let S_a and S_b be the subsets of S corresponding to the subsets \mathcal{A} and \mathcal{B} . By using the definition of convex hull, it is easy to show that (i) $CH(S) = CH(CH(S_a) \cup CH(S_b))$, and (ii) there exist points a and b within $CH(S_a) \cup CH(S_b)$ such that $p \in \overline{ab}$. Now, without loss of generality, either (i) $a, b \in$ $CH(S_a)$, or (ii) $a \in CH(S_a)$ and $b \in CH(S_b)$. If (i),

^{*}Research supported by NSERC and Institute for Computing, Information and Cognitive Systems (ICICS) at UBC

[†]Department of Computer Science, University of British Columbia, [will,jpsember]@cs.ubc.ca

¹An applet demonstrating these results can be found at http://www.cs.ubc.ca/~jpsember/uh.html.

²All sets in this paper are assumed to be multisets.

then $\overline{ab} \subseteq \operatorname{CH}(S_a)$, and since (by definition) $\operatorname{CH}(S_a) \subseteq PH(\mathcal{A})$, $\overline{ab} \subseteq PH(\mathcal{A})$, which implies (by Lemma 3) that $\overline{ab} \subseteq Q$. If (ii), then since $\operatorname{CH}(S_a) \subseteq PH(\mathcal{A})$ and $\operatorname{CH}(S_b) \subseteq PH(\mathcal{B})$, Lemma 2 implies $\overline{ab} \in Q$. Hence $PH(\mathcal{A} \cup \mathcal{B}) \subseteq Q$.

If p is a point in Q, then by Lemma 2, $p \in \overline{ab}$, where $a \in PH(\mathcal{A})$ and $b \in PH(\mathcal{B})$. There must then exist feasible sets S_a of \mathcal{A} and S_b of \mathcal{B} where $a \in CH(S_a)$ and $b \in CH(S_b)$. Note that \overline{ab} is within $CH(S_a \cup S_b)$, and since $S_a \cup S_b$ is a feasible set of $\mathcal{A} \cup \mathcal{B}$, $CH(S_a \cup S_b)$ is within $PH(\mathcal{A} \cup \mathcal{B})$; hence $Q \subseteq PH(\mathcal{A} \cup \mathcal{B})$. \Box

Lemma 5 The possible hull of any set of two or more connected uncertain regions is simply connected.

Proof. Let $\mathcal{R} = \{A, B\}$ be a set of connected uncertain regions (Lemma 4 implies that the proof extends by induction to sets of more than two regions). By Lemma 2, every point of PH is connected within PH to both Aand B; and by Lemma 3, both A and B lie within PH. Hence PH is connected, and we need only show that it has no holes. We will do this by showing that every exterior point of PH is the source of a ray exterior to PH. Let q be any point exterior to PH. First, observe that if no lines through q that are tangent to Aexist, then (i) every line through q will intersect A, and (ii) at least one line through q will intersect A to both sides of q. Since the same argument applies to B, if no such lines exist for A or B, then some segment \overline{ab} exists (where $a \in A$ and $b \in B$) that contains q, implying that $q \in PH$, a contradiction. Hence, we can assume that there exist directed lines L_1 and L_2 through q that are right-tangent to (without loss of generality) A. Let W_1 (resp., W_2) be the wedge lying on or to the right (resp., left) of both L_1 and L_2 . Note that W_1 contains A. Note also that W_2 cannot intersect B, otherwise some segment \overline{ab} exists that contains q. Let R be any ray from q lying in W_2 . Observe that no point $r \in R$ can lie on a segment \overline{ab} , since for any choice of $a \in A$, the portion of ray \overrightarrow{ar} lying at or beyond r lies within W_2 (Figure 2). Hence, by Lemma 2, R does not intersect PH.



Figure 2: Lemma 5

Corollary 6 Let (p_1, \ldots, p_k, p_1) be a cyclic sequence of points. If each consecutive pair (p_i, p_{i+1}) are endpoints

of a segment known to lie within PH, and P is a simple polygon whose edges lie on these segments, then the interior of P lies within PH.

Theorem 7 The possible hull of any set of two or more connected uncertain regions is star-shaped.

Proof. Let $\mathcal{R} = \{A, B\}$ be a set of two connected uncertain regions (Lemma 4 can be applied to prove the claim for sets of more than two regions). We will prove that PH is star-shaped by showing that it has a nonempty kernel.

Let $I = CH(A) \cap CH(B)$. If $I \neq \emptyset$, then let s be any point in I. Observe that there must exist points $a_1, a_2 \in A$, and $b_1, b_2 \in B$ where s lies on both $\overline{a_1 a_2}$ and $\overline{b_1 b_2}$. Let q be any point in PH. By Lemma 2, there exist points $a' \in A$, $b' \in B$ such that $q \in \overline{a'b'}$. Without loss of generality we can assume that a_1, b_1 , and a' are on or to the left of \overline{sq} , and that a_2, b_2 , and b' are on or to the right of \overline{sq} . There are now two cases (Figure 3): s, b_1 , and a_2 are either to the left or to the right of $\overline{a_1 b_2}$. In both cases, we can construct a cyclic sequence of points defining a polygon (per Corollary 6) that lies within PH, and which contains \overline{sq} . (In the former case, the sequence is $(a_1, b_2, a_2, b', a', b_1)$; and in the latter case, it is (b_1, a_2, b', a') .) Since this holds for any $q \in PH$, s is in the kernel of PH.



Figure 3: Theorem 7

If $I = \emptyset$, then there exists line L_1 right-tangent to Aand left-tangent to B, and line L_2 left-tangent to A and right-tangent to B. Let s be the point where L_1 and L_2 cross, and q be any point in PH. By Lemma 2, $q \in \overline{ab}$, for some $a \in A, b \in B$. Note that there exist points $a' \in A$ and $b' \in B$ such that $s \in \overline{a'b}$ and $s \in \overline{ab'}$. We can again construct a sequence of points that defines a polygon that lies within PH and contains \overline{sq} . If s lies to the right of \overline{ab} , this sequence is (b, a, b', a'); otherwise, it is (a, b, a', b').

From this point on, we will assume that each region of \mathcal{R} is a polygon (or a point, which can be viewed as a degenerate polygon). We will refer to an edge of each such polygon as a *native segment*, and to a segment connecting vertices of distinct polygons of \mathcal{R} as a *bridge segment*.

Theorem 8 If \mathcal{R} is a set of uncertain polygons with a total of n vertices, then $PH(\mathcal{R})$ is a star-shaped polygon with at most n vertices.

Proof sketch. PH is star-shaped, by Theorem 7. It can be shown (we omit the details) that every point on the boundary of PH lies on either a native segment or a bridge segment of \mathcal{R} ; hence, PH is a polygon. To bound its complexity, it can also be shown that each boundary vertex v that is not already a vertex of \mathcal{R} can be associated with a subset of the boundary of one of the polygons A of \mathcal{R} that (i) is disjoint from the subset associated with any other vertex of PH, and (ii) contains a vertex of A (in the interior of PH) to which we can charge v.

3 Point and Polygon

By Lemma 5, the possible hull of a point s and a polygon P, $PH = PH(\{s, P\})$, is equal to the possible hull of s and the boundary of P. Hence, it will suffice for our algorithm to construct the possible hull of a point and a simple polygonal chain.

Our algorithm is reminiscent of Melkman's algorithm for finding the convex hull of a simple polygonal chain [4] for two reasons: first, both are on-line algorithms; and second, both look for points where the chain enters and emerges from the interior of the hull, and rely upon the simplicity of the chain to perform this efficiently.

We motivate our algorithm with the following observation: the possible hull of a point s and a chain P is equal to a union of triangles, where each triangle's vertices are s and the endpoints of an edge of P.

Let (p_1, \ldots, p_n) be the ordered vertices of P. We will denote the connected subset of P from a to b by $(a \ldots b)$. We start with point u initialized to p_2 , and the current possible hull H initialized to $PH(\{s, (p_1 \ldots p_2)\}) = \triangle sp_1p_2$. We advance u along P, processing each new edge (or part of an edge) in one of two ways. If the edge is exterior to H, then we expand H by adding its associated triangle; and if the edge is interior to H, then we skip the edge and advance u until it emerges from H. In either case, at the start of each iteration, u is a point that is on both P and the boundary of H. When u reaches p_n , H will equal $PH(\{s, P\})$.

We assume the vertices of H (which, by Theorem 8, is a simple polygon) have a ccw ordering. For added flexibility, we will associate with H a variable orientation whose values are ccw or cw. If a and b are adjacent vertices of H, with b ccw from a, then we will consider bto follow a when H has ccw orientation, and to precede a when H has cw orientation. Our algorithm has these steps:

- 1. Initialize H to be $\triangle sp_1p_2$, and u to be p_2 .
- 2. If $u = p_n$, stop.

- 3. Set the orientation of H to ccw (resp., cw) if s lies to the left (resp., right) of \overline{uv} , where v is the vertex of P following u.³
- 4. If the points of P immediately following u lie in the interior of H, go to step 6.
- 5. Expansion step. Let $T = \triangle suv$ (\overline{uv} 's contribution to the hull). Starting with x = u, advance x along H, deleting those edges that lie within T, until the first of three events occurs:
 - (i) H has no more edges (Figure 4). Replace H with T, advance u to v, and go to step 2.



Figure 4: Expansion step, case (i), before and after changes to H.

(ii) x reaches the point where edge \overline{ab} of H intersects \overline{sv} . Replace \overline{ab} with edges \overline{uv} , \overline{vx} , and \overline{xb} (Figure 5). Advance u to v, and go to step 2.



Figure 5: Expansion step, case (ii), before and after changes to H.

(iii) x reaches the point where edge \overline{ab} of H intersects \overline{uv} . Replace \overline{ab} with \overline{ux} and \overline{xb} (Figure 6). Advance u to x, and go to step 2.



Figure 6: Expansion step, case (iii), before and after changes to H.

 $^{^{3}\}mathrm{In}$ each of the figures that follow, H has ccw orientation according to this rule.

- 6. Interior step. Starting with x = u, advance x along P until the first of two events occurs:
 - (i) x reaches p_n ; stop.
 - (ii) x reaches the point where P emerges from H's interior (Figure 7). Split \overline{cd} , the edge of H containing this point, into edges \overline{cx} and \overline{xd} . Advance u to x, and go to step 2.



Figure 7: Interior step, case (ii).

We will use induction to show that at the start of each iteration of the algorithm, the following invariants hold:

- 1. *H* is the possible hull of *s* and $(p_1 \ldots u)$.
- 2. u lies on the boundary of H.

The invariants clearly hold for the base case, since the initial hull, H, is triangle $\triangle sp_1 u$ (where $u = p_2$), and is thus equal to $PH(\{s, (p_1 \dots u)\})$.

Suppose the invariants hold for every iteration until ureaches a particular position along P, and an expansion step is to be performed. If every edge of H lies within T (case i), then T is the possible hull of $(p_1 \dots v)$, and the invariants are satisfied. Suppose instead that some edge of H does not lie within T. Consider the boundary points of H following u. Since s lies in the kernel of H, the first such point where the boundary crosses an edge of T must lie on \overline{sv} (case ii), or in the interior of \overline{uv} (case iii). In the former case, modifying the boundary of H as stated has the effect of expanding H to include $\triangle suv (=$ T); and in the latter, it has the effect of expanding Hto include $\triangle sux (\subset T)$. Since u is advanced to v in the former and x in the latter, we are thus adding exactly that portion of the hull contributed by those points of P between the old and new u, which satisfies invariant (1); and since the new u is not interior to H, invariant (2) is also satisfied.

Now consider the case where an interior step is to be performed. It is easy to show that if H is the possible hull of s and a set J, then $H = PH(\{s, H \cup J\})$. Hence, we can ignore points on $(u \dots x)$. The only change we make to H is to split edge \overline{cd} at x. As this does not actually change the boundary of H, and x (the new location of u) lies on this boundary, both invariants are satisfied.

Since the invariants hold for each iteration of the algorithm, we can claim: **Theorem 9** The above algorithm generates the possible hull of a point and a simple polygonal chain.

We now examine the running time of the algorithm. To simplify the analysis, we assume that s is not collinear with any two vertices of P. Step 4 takes constant time, since the points of P immediately following u lie in the interior of H iff the vertex of H following u is to the right (or, if the hull has cw orientation, to the left) of \overline{uv} .

In step 6, we must determine which edge of H contains the point x where P emerges from H's interior. To do this efficiently, we start by characterizing each boundary edge of H as being either a *polygonal* edge (lying on an edge of P) or a *radial* edge (lying on a ray from s through a vertex of P). By assumption, no edge can be both.

Lemma 10 At the start of any iteration in which an interior step occurs, the following conditions hold: (i) exactly one of the edges of H incident with u is a radial edge; and (ii) if this radial edge is not incident with s, then x (the point where P emerges from H's interior) must lie on this edge as well; otherwise, x must lie on an edge incident with s.

Proof. At the start of an interior step, the edges of H incident with u cannot both be polygonal edges, otherwise (since an interior step is about to occur) this would imply that u is incident to three edges of P, which is impossible. Suppose instead that that they are both radial edges. Note that each radial edge of H is induced by a distinct vertex of P (unless a radial edge has just been split in step 6(d); but each such step is immediately followed by an expansion step that removes one of these two edges). Hence, s and two distinct vertices of P must be collinear, contradicting our general position assumption.

To prove (ii), we first note that since P is simple, xmust lie in the interior of \overline{cd} , a radial edge of H. Let \overline{ab} be the radial edge of H incident with u. Assume by way of contradiction that \overline{ab} and \overline{cd} are distinct edges, and that at least one of them is not incident with s. This edge must then be adjacent to (distinct) polygonal edges y_1 and y_2 (see, for example, edge \overline{ab} in Figure 8). Now observe that $(u \dots x)$ partitions H into two pieces, and since $\overline{ab} \neq \overline{cd}$, y_1 and y_2 must lie on opposite sides of $(u \dots x)$. We now have a contradiction, since the interiors of paths $(p_1 \dots u)$ and $(u \dots x)$ must intersect, which implies that P is nonsimple.

Lemma 10 implies that in order to find x while moving along edges of P during an interior step, we need to check for intersections of P with at most two edges of H: the single radial edge incident with the point of entry u, and (if that edge is also incident with s) the other edge incident with s.



Figure 8: Lemma 10.

Let us determine the total number of vertices processed by the algorithm. These include the vertices of P, plus any vertices that ever appear in H. Consider the start of a particular iteration, where H is the current hull, u is the current position on P, and v is the vertex of P following u. We will show that at most three vertices are introduced to H by the addition of triangle $T = \triangle suv$.

Each new vertex (other than v) is a point where the boundaries of H and T cross, and hence must lie on either \overline{uv} or \overline{vs} (since $\overline{su} \subset H$). Suppose for the sake of contradiction that two new vertices, p and q, lie in the interior of \overline{uv} . We can assume that \overline{uv} first enters H at p, then exits H at q. Since $\overline{uv} \subset P$, and P is simple, p and q must lie on radial edges of H. These radial edges must lie on rays \vec{sa} and \vec{sb} respectively, where a and b are vertices of P preceding u (Figure 9). The path $(a \dots b \dots u)$ (or $(b \dots a \dots u)$) in P cannot cross rays \overrightarrow{pa} or \overrightarrow{qb} (otherwise p or q would lie in H's interior), nor can it cross \overline{pq} (since P is simple). We now have a contradiction, since this implies that $(a \dots b)$ is not connected to $(u \dots v)$ within P. Hence, at most one new vertex lies in the interior of \overline{uv} ; and since s is in the kernel of H, at most one of the new vertices lies in the interior of \overline{vs} .



Figure 9: p and q cannot both be new vertices of H.

Theorem 11 The above algorithm generates the possible hull of a point and a simple polygonal chain of n vertices in O(n) time.

Proof. We store the vertices of H and P in doublylinked lists, so that inserting or removing a vertex, or accessing a vertex's neighbor, can be done in constant time. Every step of the algorithm can be done in constant time, except for the expansion and interior steps, which can be done in time proportional to the number of vertices that are: (i) visited on the chain; (ii) inserted into the hull; or (iii) removed from the hull. Since a vertex can be removed from the hull only once, the total running time of the algorithm is bounded by the number of vertices processed by the algorithm (which, as we have shown, is O(n)) and the number of iterations (which is also O(n), since each advances u to a distinct vertex of H or P).

4 Possible Hull of Polygons

In this section, we provide an overview of an algorithm to construct the possible hull of a pair of uncertain polygons A and B (a more detailed presentation, which includes a correctness proof, can be found in [7]). It employs the algorithm of the previous section as a subroutine to achieve an optimal running time.

Our algorithm starts with a polygon H equal to $CH(A \cup B)$, then modifies H's boundary until H is equal to $PH(\{A, B\})$. If a boundary edge of $CH(A \cup B)$ is a polygonal segment or a bridge segment, then by Lemmas 2 and 3, and the fact that $PH(\{A, B\}) \subseteq CH(A \cup B)$, it lies on the boundary of $PH(\{A, B\})$ as well. Otherwise, its vertices must be nonadjacent vertices from the same polygon (e.g., a_i and a_j). As we manipulate H, both a_i and a_j will remain vertices of H, but the path $(a_i \ldots a_j)$ on H's boundary will change. We will refer to this path as a *pocket* of H, and to segment $\overline{a_i a_j}$ as the pocket's *lid*.

Our algorithm has these steps:

- 1. Initialize H to $CH(A \cup B)$.
- 2. For each pocket lid $\overline{a_i a_j}$ of H, perform the following *hull contraction* steps:
 - (a) Using the algorithm of the previous section, construct J, the possible hull of $(a_i \dots a_j)$ (on the boundary of A) and any point s from B.
 - (b) Replace $\overline{a_i a_j}$ with path $(a_i \dots a_j)$ on the boundary of J.
- 3. Repeat the hull contraction steps with the roles of A and B reversed.
- 4. Perform the following *hull expansion* steps:
 - (a) Set u to h_1 , any vertex of $CH(A \cup B)$.
 - (b) Determine ray T_u as follows. If u is a vertex of A, then set T_u to the ray from u that is left-tangent to B; otherwise, set T_u to the ray from u that is left-tangent to A.
 - (c) If the vertex v of H following (i.e., in ccw direction) u is not left of T_u , then go to (f).
 - (d) Let x be the point where the boundary of H next crosses T_u .

- (e) Replace $(u \dots x)$ with \overline{ux} .
- (f) Advance u to the next convex vertex of H. If $u \neq h_1$, go to (b).
- 5. Repeat the hull expansion steps, substituting cw for ccw, and right for left.

The hull contraction steps use the algorithm of the previous section to replace each pocket lid with a portion of the boundary of a possible hull associated with the pocket (Figure 10). This contracts H by 'taking bites' out of the convex hull of the two polygons. Corollary 6 implies that after this modification, each pocket lies within PH. The hull expansion steps traverse the boundary of H, find tangent rays that potentially contain bridge segments of PH, and modify H to incorporate these segments. This has the effect of expanding H, by adding back some portions that were removed in the hull contraction steps. It can be shown that after the hull expansion steps have been performed for both ccw and cw directions, $H = PH(\{A, B\})$.



Figure 10: Hull contraction step: pocket lid $\overline{a_i a_j}$ replaced by $(a_i \dots a_j)$ of J.

Step 1 can be performed in O(n) time, where n is the number of vertices of A and B: first by constructing the convex hulls of both A and B (in O(n) time, e.g., by using Melkman's algorithm [4]); then by using the rotating calipers method [8] to construct $CH(A \cup B)$.

In the hull contraction steps, for each pocket lid $\overline{a_i a_j}$, we construct the corresponding subset of the boundary from A, then calculate the possible hull of this boundary and an arbitrary point of B. Since each edge of Aappears in only one of these subsets, and the possible hulls can be constructed in time linear in the size of the subset (Theorem 11), we can perform these steps in O(n) time.

Step 2(b) plays a crucial role in the algorithm. It ensures that throughout the hull expansion steps, each pocket $(a_i \ldots a_j)$ is a sequence of points that have monotonically increasing polar angles with respect to a point $s \in B$ (the pocket would not necessarily have this property if, for example, it was instead initialized to path $(a_i \ldots a_j)$ on the boundary of A). The monotonicity property implies that the tangent rays T_u in step 4(b) can be found by using the rotating calipers method [8], which (it can be shown) implies that the running time of each hull expansion step is O(n). Hence:

Theorem 12 The possible hull of a pair of polygons with n total vertices can be constructed in O(n) time.

The running time of our algorithm is clearly optimal, since it matches the input size. We can adapt the algorithm to the case where there are more than two polygons:

Theorem 13 The possible hull of k polygons with a total of n vertices can be constructed in $O(n \log k)$ time, and this running time is optimal in the worst case.

Proof. Lemma 4 implies that we can apply our O(n) algorithm for pairs of polygons recursively, in a divideand-conquer manner, to construct the possible hull of k polygons. In doing so, we increase the running time by a factor that is logarithmic in the height of a binary tree of k elements. It is worst-case optimal, since if the input consists of n/3 small triangles distributed along a circle, the problem reduces to constructing the convex hull of O(n) points (each of which lies on the hull).

Acknowledgment

The authors would like to thank David Kirkpatrick for his helpful discussions and valuable insights.

References

- A. Edalat and A. Lieutier. Foundation of a computable solid modelling. *Theor. Comput. Sci.*, 284(2):319–345, 2002.
- [2] A. Edalat, A. Lieutier, and E. Kashefi. The convex hull in a new model of computation. In *CCCG*, pages 93–96, 2001.
- [3] M. Löffler and M. J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.
- [4] A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Inf. Process. Lett.*, 25(1):11–12, April 1987.
- [5] T. Nagai, Y. Seigo, and N. Tokura. Convex hull problem with imprecise input. In *Revised Papers from the Japanese Conference on Discrete and Computational Geometry*, pages 207–219, London, UK, 2000. Springer-Verlag.
- [6] T. Nagai and N. Tokura. Tight error bounds of geometric problems on convex objects with imprecise coordinates. In JCDCG '00: Revised Papers from the Japanese Conference on Discrete and Computational Geometry, pages 252–263, London, UK, 2001. Springer-Verlag.
- [7] J. Sember. Guarantees Concerning Geometric Objects with Uncertain Imputs. PhD thesis, University of British Columbia, Forthcoming 2011.
- [8] M. I. Shamos. Computational geometry. PhD thesis, Yale University, 1978.

A Slow Algorithm for Computing the Gabriel Graph with Double Precision

David L. Millman

Vishal Verma *

Abstract

When designing algorithms, time and space usage are commonly considered. In 1999, Liotta, Preparata and Tamassia proposed that we could also analyze the precision of an algorithm. We present our first steps towards the goal of efficiently computing the Gabriel graph of a finite set of sites, while restricting ourselves to only double precision.

1 Introduction

Computers use finite precision arithmetic to test geometric relationships between objects. Sometimes, a computer cannot provide a sufficient number of arithmetic bits to guarantee that the tests are correct. It is natural then to analyze the number of bits required to correctly run an algorithm. One such model of analysis was proposed by Liotta, Preparata and Tamassia [4]. They define the degree (or arithmetic complexity) of an algorithm in terms of the arithmetic degree of its predicates. One can then attempt to minimize the degree of an algorithm, just like time and memory. This type of analysis is known as *degree-driven algorithm design*, and it tells us the amount of arithmetic precision required to run the algorithm safely. In Section 2, we define degree and arithmetic precision more formally. Arithmetic precision is of special interest when running geometric algorithms, that assume general position, on practical datasets that are frequently close to being degenerate. To avoid these issues of instability we have been working on degree driven algorithms for constructing geometric structures. In this paper we describe our first step towards constructing the Gabriel graph robustly.

Given a finite set of sites S, an edge (s_i, s_j) with $s_i, s_j \in S$ is in the Gabriel graph of S if the edge maintains the Gabriel property, that is, the closed disk with diameter $\overline{s_i s_j}$ contains no points of S besides s_i and s_j . It is know that the Gabriel graph [2] is a subgraph of the Delaunay triangulation. Matula and Sokal [6] showed how to compute the Gabriel graph directly from the Delaunay triangulation in time proportionate to the number of sites in S.

Computing the Delaunay triangulation requires four times the precision of the input coordinates, and Matula and Sokal's Gabriel graph algorithm uses six-fold precision. Liotta [3] showed how to implement Matula and Sokal's algorithm using only two-fold precision, however, it still requires four-fold precision for computing the Delaunay triangulation. A natural question that follows is, can we compute the Gabriel graph with only two-fold precision?

The answer is yes! In Section 4 we show that we can compute the Gabriel graph with two-fold precision (albeit rather slowly).

2 Definitions and Notation

We begin by recalling how one can analyze arithmetic complexity. Assume that the coordinates of our input can be scaled to *b*-bit integers. Thus, we can think of the sites of *S* as lying on the $U \times U$ grid, notated as \mathbb{U} . The primitives of a geometric algorithm are called *predicates*, which are tests of the signs of multivariate polynomials with variables from the input coordinates. We say that the *degree* of a predicate is the degree of the polynomials to which it corresponds (for a degree *d* predicate we sometimes say it uses *d*-fold precision). Furthermore, we define the *degree of an algorithm* by the highest degree predicate it evaluates.

Consider, for example, testing if point q is closer to point p_1 or p_2 with $p_1, p_2, q \in \mathbb{U}$. We can write this predicate as $\operatorname{sign}(||q-p_1||^2 - ||p-p_2||^2)$. Which expands to a degree 2 polynomial, thus, this predicate is degree 2, (i.e., it uses two-fold precision). Another example (used in Section 4) tests if the straight line path from p_1 to p_2 to q, with $p_1, p_2, q \in \mathbb{U}$, forms a counterclockwise orientation. This Orientation (p_1, p_2, q) predicate, tests the sign of the determinant of the homogeneous coordinates of p_1, p_2 and q, and is also degree 2.

Next, recall the point/line duality [1] that maps a point $p = (p_x, p_y)$ to a line $p^* := (y = p_x x - p_y)$ and a line l := y = mx + b to a point $l^* := (m, -b)$. The set S^* is the set of lines, dual to the set of sites of S. We notate the arrangement of the lines in S^* is $A(S^*)$ and the Gabriel graph of S as G(S).

3 Arrangements of Dual Lines

It is know that for a set of line segments, defined by their endpoints, computing an arrangement requires four times the input precision and computing its trapezoidation requires five times the input precision [5]. In

^{*}Department of Computer Science, University of North Carolina at Chapel Hill. [dave,verma]@cs.unc.edu

this section, we show that for a set of lines, defined as duals of points, computing an arrangement and its trapezoidation can be solved with double precision.

We begin by observing that for non-parallel lines p^* and q^* , the the *x*-coordinate of the point $\ell^* = p^* \cap q^*$ is the slope of line $\ell = \overleftarrow{pq}$, which is $(p_y - q_y)/(p_x - q_x)$.

Observation 1 The x-coordinate of the intersection of two dual lines is represented by a rational polynomial of degree 1 over degree 1.

For three dual lines, p^* , q^* , and r^* , where p^* and q^* intersect r^* , by Observation 1 and clearing fractions, we compare the *x*-ordering of the intersection points with degree 2. We call this the **OrderOnALine** (p^*, q^*, r^*) predicate.

By using the OrderOnALine predicate in an incremental construction of an arrangement (such as [1, Chapter 8.3]) we achieve a degree 2 construction. Furthermore, we can use the OrderOnALine predicate to add the verticals into the arrangement and get its trapezoidation.

Lemma 1 For n dual lines S^* , we can compute the arrangement of S^* and its trapezoidation in $O(n^2)$ time and degree 2.

4 Gabriel Graphs

Next, we describe how to construct the Gabriel graph in $O(n^2)$ using degree 2, and begin by defining the primitives of our construction. Let D(p,q) be the closed disk with \overline{pq} as the diameter. We say that a site *s* kills the edge (p,q) if *s* lies in D(p,q). Let *m* be the midpoint of \overline{pq} . The degree 2 predicate IsKiller(p,q,s) compares the squared distance between *m* and *s* and *m* and *p* to determine if *s* kills edge (p,q).

Given the arrangement $A(S^*)$ and a site $s_i \in S$ we would like to compute the circular orderings of the sites in $S \setminus \{s_i\}$ around s_i . Consider the line s_i^* , each vertex $v_j \in A(S^*)$ that lies on s_i^* corresponds to a line though s_i and some other site $s_j \in S$. As mentioned in Section 3, the slope of $\overleftarrow{s_i s_j}$ is the x-coordinate of v_j , thus, by walking along s_i^* in $A(S^*)$ we find a set of lines, though s_i ordered by slope, which gives the circular ordering of the sites of $S \setminus \{s_i\}$ around s_i .

Constructing a circular ordering for a site is a purely topological operation on the arrangement $A(S^*)$ and uses degree 0. For each site s, computing the circular ordering takes time proportional to the number of vertices that lie on s^* , which is O(n).

Lemma 2 Given $A(S^*)$, for site $s \in S$, we can compute the circular ordering of the sites in $S \setminus \{s\}$ around s in O(n) time and degree 0.

Once we have computed a circular ordering of $S \setminus \{s\}$ around each $s \in S$, in O(n) time we can compute the



Figure 1: s_j lies in the part of $D_l(s, s_i)$ that is to the left of $s\vec{s}_k$. This part of $D_l(s, s_i)$ is a subset of $D_l(s, u_k)$, which itself is a subset of $D_l(s, s_k)$.

Gabriel edges incident at s. The key idea behind this step is captured in Lemma 3.

We number the circularly ordered sites in $S \setminus \{s\}$ in a counterclockwise manner starting with any $s_0 \in S \setminus \{s\}$ *i.e.* ss_{i+1}^{-1} is the first ray counterclockwise from $s\vec{s}_i$ at s. Let $D_l(s, s_i)$ denote the closed semicircular disk that has $\overline{ss_i}$ as the diameter and lies to the left of $s\vec{s}_i$. We say s_j kills the edge (s, s_i) from the left if and only if s_j lies in $D_l(s, s_i)$. Then,

Lemma 3 If s_j lies in $D_l(s, s_i)$ and $\forall k \in \{i, i + 1, \ldots, j-1\}$, $s_k \notin D_l(s, s_i)$, then s_j also lies in $D_l(s, s_k)$, $\forall k \in \{i, i+1, \ldots, j-1\}$

Intuitively, the above lemma says that if s_j is the first site (in a counterclockwise sense) that kills (s, s_i) from the left, then s_j kills all $(s, s_k), i \leq k \leq j - 1$, from the left. Figure 1 gives a brief idea of the lemma and the proof.

Proof. Since $i \leq k \leq j-1$ and $s_k \notin D_l(s, s_i)$, the segment $\overline{ss_k}$ intersects the circular part of the boundary of $D_l(s, s_i)$ at some point u_k . For every point $p \in D_l(s, u_k), \angle sps_k \geq \angle spu_k > \pi/2$. Thus p also lies in $D_l(s, s_k)$. Hence $D_l(s, u_k) \subset D_l(s, s_k)$.

Let H be the closed half plane that lies on left of the line $s\vec{u}_k$. For every point $p \in D_l(s, s_i) \cap H, \angle spu_k \ge \angle sps_i > \pi/2$. Thus $(D_l(s, s_i) \cap H) \subset D_l(s, u_k)$. Using this with the subset relation from the previous paragraph we have $(D_l(s, s_i) \cap H) \subset D_l(s, s_k)$. Since s_j lies in $(D_l(s, s_i) \cap H)$ it also lies in $D_l(s, s_k)$.

Let L be the circular linked list of sites of $S \setminus \{s\}$ circularly ordered around s. Algorithm 1 efficiently identifies sites $s' \in L$ such that the edge (s, s') is killed from the left by some site in L. Such vertices are marked *dead* by the algorithm. A similar algorithm is used to identify the sites s'' such that the edge (s, s'') is killed from the right. The edges that are killed neither from left nor right belong to the Gabriel graph.

We now give a brief overview of Algorithm 1. Given a site $u \in L$, we define $left_victims(u)$ as a subset of $S \setminus \{s\}$ such that for each site $v \in left_victims(u), u$ is the first (when walking left along the list L) site to kill the edge (s, v) from left. Lemma 3 says that the set $left_victims(u)$ is contained in a continuous sublist of L that starts on the right of u and only contains sites wsuch that (s, w) is killed from left by u. This observation is used in the inner while loop of Algorithm 1 to find a sublist L_u such that: (a) each site in L_u has a killer in $(u \cup L_u)$; and (b) the union of the *left_victims* of the sites in $(u \cup L_u)$ is a subset of L_u . Due to (a), we know that the sites in L_u can be killed from left and hence they are marked *dead*. Due to (b), for any remaining site $v \in L \setminus L_u$, if the edge (s, v) is killed from left then v belongs to the set of left_victims of some site in $L \setminus L_u$. Thus, to find the remaining $left_victims$, we process the smaller list $L \setminus L_u$.

Algorithm 1: KillFromLeft(S, s, L)

Make a copy L_{left} of L; Initialize the unseen values of each site in L to *true*; Initialize the dead values of each site in L to *false*; $u = \text{any site in } L_{left};$ while $u \rightarrow unseen$ do $u \rightarrow \text{unseen} = false;$ killer = u; $current = u \rightarrow right;$ while $killer \neq current$ do if $killer \in D_l(s, current)$ then $current \rightarrow dead = true;$ $current = current \rightarrow right;$ else $killer = killer \rightarrow right;$ end end L_u = the sublist of L_{left} , that is to the right of u and left of *current*; Delete L_u from L_{left} ; $u = u \rightarrow \text{left};$ end

Testing if $killer \in D_l(s, current)$ uses degree 2 predicates isKiller and Orientation, thus, the above algorithm runs in O(|L|) time and is degree 2.

Lemma 4 Given the circular ordering of $S \setminus \{s\}$ around s, in O(n) time and degree 2, we can find the Gabriel edges incident at s.

For completeness, we describe the three steps for constructing the Gabriel graph of a set of n sites S with degree 2. First, compute $A(S^*)$, which by Lemma 1 takes $O(n^2)$ time and degree 2. Second, for each site $s_i \in S$ compute the circular ordering of the sites of $S \setminus \{s_i\}$, which in total, by Lemma 2, takes $O(n^2)$ and degree 0. Third, for each site $s_i \in S$, use the circular orderings to compute the set of Gabriel edges in which s_i is a member, which in total, by Lemma 4, takes $O(n^2)$ and degree 2.

Corollary 5 We can compute the Gabriel graph in $O(n^2)$ time using degree 2.

5 Conclusion and Open Problems

Even though an $O(n^2)$ construction is too slow for practical applications, Corollary 5 tells us that we can at least compute the Gabriel graph with degree 2 and do better than brute force. In contrast, we simply cannot compute the Delaunay triangulation with degree 2. Two questions follow.

Firstly, can we compute the Gabriel graph in subquadratic time with degree 2? It may be of interest to note that the grid size does not appear in the running time of the algorithm. Thus, the algorithm still terminates if we let the step size of the grid shrink to zero.

Secondly, since we cannot compute the Delaunay triangulation with degree 2, can we compute a triangulation that is in some sense close to Delaunay? With degree 2 we can compute the convex hull and Gabriel graph, but which edges should we add to complete the triangulation?

References

- M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Applications. Springer-Verlag New York, Inc., 3rd edition, 2008.
- [2] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):pp. 259–278, 1969.
- [3] G. Liotta. Low degree algorithms for computing and checking gabriel graphs. Technical report, Providence, RI, USA, 1996.
- [4] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. SIAM J. Comput., 28(3):864–889, 1999.
- [5] A. Mantler and J. Snoeyink. Intersecting red and blue line segments in optimal time and precision. In *Discrete* and *Computational Geometry*, number 2098 in LNCS, pages 244–251. Springer Verlag, 2001.
- [6] D. W. Matula and R. R. Sokal. Properties of Gabriel Graphs Relevant to Geographic Variation Research and the Clustering of Points in the Plane. *Geographical Analysis*, 12(3):205–222, 1980.

An Experimental Analysis of Floating-Point Versus Exact Arithmetic^{*}

Martin $Held^{\dagger}$

Willi Mann[†]

Abstract

In this paper we investigate how sophisticated floatingpoint codes that are in real-world use - VRONI for computing Voronoi diagrams, FIST for computing triangulations, and BONE for computing straight skeletons – can benefit from the use of the Core library (for exact geometric computing) or the MPFR library (for multiprecision arithmetic). We also discuss which changes to the codes were necessary in order to get them to run with these libraries. Furthermore, we compare our codes to codes provided by the CGAL project. By means of GMP-based (brute-force) verifiers we check the numerical validity of the outputs generated by all codes. As expected, the output precision of VRONI increases when MPFR is used, at a cost of an average slow-down by a multiplicative factor of 70. On the other hand, FIST demonstrates that a careful engineering can enable a code that uses floating-point arithmetic to run flawlessly, provided that the input coordinates are interpreted as genuine floating-point numbers. To our surprise, we could not get VRONI and BONE to work with CORE. It is similarly surprising that their CGAL counterparts did not fare well at all: we recorded drastically increased CPU-time consumptions combined with decreased accuracy of the numerical output.

1 Introduction

Robustness problems that occur for geometric codes when executed on a floating-point (fp) arithmetic are notorious. Typically, robustness problems are caused by numerical quantities being approximated, up to some quantitative error. While most errors tend to be benign, some errors may cause a program to end up in a state with no graceful exit, i.e., it crashes.

Various alternatives to standard floating-point computations have been advocated in recent years in an attempt to such robustness problems, such as the use of multi-precision arithmetic or exact geometric computing (EGC). The Core library, CORE [1], is an implementation of state-of-the-art EGC algorithms and techniques. It is written in C++, and was designed to be used easily as an alternative arithmetic backend to existing C/C++ programs. CORE is a generalpurpose tool that allows the correct evaluation of the sign of real predicates and, thus, is a way to ensure that the combinatorial part of an algorithm is computed exactly. The CGAL project [5] makes use of this EGC approach. Shewchuk [12] offers a small collection of geometric predicates that also support an exact evaluation based on standard fp-arithmetic. Another option is given by the MPFR library [3]: it is a "multi-precision floating-point library with correct rounding", and can be considered as an intermediate step between IEEE 754 double-precision fp-arithmetic [10] and CORE.

Just how easy is it really to interface an existing geometric code with MPFR or CORE? And what do we gain or lose by resorting to MPFR, CORE or CGAL? In this paper we carry out an experimental case study that attempts to provide an answer to this question beyond personal beliefs or wide-spread myths. We consider three problems of imminent practical interest – the computation of triangulations, Voronoi diagrams and straight skeletons of polygons – and take three codes that compute these structures on a fp-arithmetic: the C codes FIST [6] and VRONI [7, 8], and the C++ code BONE [9]. All three codes were engineered to be reliable, and FIST and VRONI have been used extensively in industry and academia for more than a decade.

In Sec. 2 we discuss the modifications of our codes required to adapt them to a use with MPFR 3.0.1 or CORE 2.1, and report on problems encountered. (Due to lack of space we omit details on our results for BONE; they are similar to those for VRONI.) Section 3 documents the results of our run-time and verification tests.

2 Preparations

2.1 Modifications Required for CORE

CORE 2.1 data types do not work with C functions like printf() and scanf(), which we wanted to preserve in order to allow FIST to be compiled as a standard C program. For scanf() the problem can be worked around by implementing a custom version of scanf() that interprets the format specifiers for floating-point data types as specifiers for the CORE Expr type. Supporting printf() required more work because the arguments of a printf() command need to be converted to pointers. (Variable-argument functions cannot take C++ objects as arguments.)

Memory management also needed to be changed: The

^{*}Work supported by Austrian FWF Grant L367-N15.

[†]Univ. Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; {held,wmann}@cosy.sbg.ac.at

C functions malloc() and free() do not call constructors and destructors of C++ objects. We replaced them by the C++ operators new and delete.

And, of course, all precision thresholds used in comparisons of a numerical value with zero were set to zero. We were once again reminded of the fact that an ε -based comparison of some variable x with zero should be encoded as " $|x| \leq \varepsilon$ " rather than as " $|x| < \varepsilon$ ". (Otherwise, setting $\varepsilon := 0$ does not work.)

We also learned quickly that algorithmically equivalent code fragments may result in substantially different expression trees and, thus, runtimes for a CORE-based execution. For instance, in order to compute an (approximate) normal vector of a 3D facet whose vertices might be not perfectly co-planar it is advisable to first test whether the facet is truly planar. If yes then any three vertices that are not collinear allow to compute the correct normal vector. Otherwise, no correct normal vector exists and an approximate normal obtained by averaging normals defined by triples of vertices of the facet should be computed by using standard floatingpoint arithmetic, rather than by resorting to CORE and blowing up the size of the expression trees by making the normal dependent on all vertices of the facet.

2.2 Difficulties with CORE

Our work helped to reveal two major problems in CORE 2.0.8. The first problem concerns the parsing of the input: Some values in the interval (-0.1, 0.1) were not parsed correctly. The CORE developers fixed the problem for positive numbers, and we extended the fix to negative numbers; it has become part of CORE 2.1.

Another problem is caused by the conversion from real to integer types. The CORE type for real numbers, Expr, supports a method called intValue(). The documentation shipped with CORE describes this method (and a few others) by the sentence "The semantics of these operations are clear." So we felt it safe to assume that intValue() behaves like the (int) conversion known from C. Unfortunately, it turned out the CORE conversion sometimes rounds up and sometimes rounds down. (It bases its rounding decision on a finite approximation of binary digits.) As a work-around we resorted to using the floor() function which works reliably on Expr numbers.

So far, we have been unable to execute CORE-based versions of VRONI or BONE on even trivial inputs. The apparent problem is that both codes cause large expression trees, where several minutes of CPU time do not suffice for CORE to evaluate the sign of one expression tree. (We created test cases and sent them to the developers of CORE, but no fix has been released yet.)

2.3 Adding MPFR Support

MPFR was not shipped with an integrated C++ wrapper, and the existing wrappers did not work with MPFR 3.0.1 when we tried them. (This has changed in the meantime.) So we wrote our own wrapper that supports precisely the operations needed in our applications, without adding any extra magic that might hurt the run-time performance.

As MPFR allows to set the precision at run-time, we have to adjust the precision thresholds used in the comparisons of fp-numbers. After some tests we ended up using the following formula¹:

$$\varepsilon_{\text{prec}} := \varepsilon_{\text{fp}} / 2^{100 \cdot \left(\sqrt{\text{prec}/53} - 1\right)},$$

where $\varepsilon_{\rm fp}$ is the standard threshold used for the fpcomputations and prec is the precision (number of bits) requested. (The standard IEEE 754 precision assigns 53 bits of precision to the mantissa; see [10].)

We note that setting the default precision of MPFR in main() with the mpfr_set_default_prec library call does only affect variables created after this library call. In particular, the precision of global variables is not set adequately by default — and missing the correct setting of even just a few global variables turned out to downgrade the precision of the output quite significantly. So we modified our wrapper to ensure that the target variable in assignments has the default MPFR precision set.

2.4 Using Shewchuk's Predicates in FIST

Since correct sidedness tests are important for FIST we inserted a compile-time option that allows us to replace FIST's standard determinant evaluation by Shewchuk's 2D orientation predicate [12]. It was easy to integrate his code into FIST but one caveat remains: Shewchuk explicitly warns that his predicates will not work correctly if extended-precision registers are used. We circumvented this problem by running our tests on an x86-64 hardware; see the discussion in Sec. 3.2.

3 Tests and Experimental Results

3.1 Speeding up the CORE Library

The default constructor in CORE initializes an object that represents the constant zero. A closer inspection revealed that it always creates a new node representing zero. However, FIST very often uses variables in a way that causes their default constructor to be called: The programming language C (prior to the 1999 standard of C) forces variables to be declared prior to the body of the function, which often happens without an assignment in FIST. The default constructor is

¹Thanks go to Stefan Huber for coming up with this formula.

also called frequently when arrays containing Expr objects are enlarged. A simple modification of this feature of CORE 2.0.8 resulted in an increase in speed of the CORE-based version of FIST by about 31%. (Our patch has been integrated into CORE 2.1.)

3.2 Influence of Compiler Options

In general, it is easier to debug non-optimized builds than optimized builds because compilers instructed to optimize may reorder instructions on machine-code level to gain performance. This often leads to a nonmonotonic control flow with respect to the C sources. However, a key aspect of optimized builds is the attempt to keep variables inside registers across multiple statements in the source code.

On the x86 architecture, the floating-point registers are 80 bit wide, with a mantissa of 64 bit [11]. However, floating-point variables of type double as defined by IEEE 754-1985 [10] only have a mantissa of 53 bit. So, whenever a floating-point value is moved from a floating-point register to main memory, a loss of precision occurs. Or, in other words, the precision of numerical data computed depends on which variables and which intermediate results were kept in the registers. As a result, the output may change drastically once optimization is turned on.

Tests with FIST on our test data – see below – revealed that the outputs of the optimized build (using gcc -0) and the debug build (using gcc -00) differ for about 15% of the inputs. We simply took the triangles (in the order computed by FIST) as the output. Hence, different outputs may nevertheless describe the same triangulation. Still, this means that at least one comparison in FIST returned different results on 15% of our inputs, depending on the compiler options.

There are multiple ways around this problem:

- Force the use of SSE instructions. This is not supported on all x86 CPUs. Many compilers including gcc use this as default on x86-64.
- Link the executable with a flag that limits the precision of the FPU. (E.g., -mpc64 for gcc).
- Use a compiler flag that forces the write-back of all calculated values from the registers to main memory. (E.g., -ffloat-store for gcc.)

3.3 Test Results for Triangulations of Polygons

3.3.1 Comparison of Different Arithmetic Backends

The following tests were conducted on the first author's set of polygonal areas which currently consists of 21 175 polygons with and without holes. The tests were run on an x86-64 hardware, an Intel Core i5 CPU 760 clocked at 2.80GHz. The test machine has 8 GiB of main memory, but virtual memory was limited to 6 GiB by the ulimit command. All codes were compiled with gcc 4.4.3.

We ran FIST with six different arithmetic back-ends:

- ordinary IEEE 754 double-precision fp-arithmetic (fistFp),
- Shewchuk's predicates (fistShew),
- CORE (fistCore),
- three precisions of MPFR: 53 bits (fistMp53), 212 bits (fistMp212), and 1000 bits (fistMp1000).

Figure 1 shows the run-time plots of fistFp, fistMp212, and fistCore. (The plots for the other three variants have been omitted due to lack of space.) The use of Shewchuk's exact predicates (fistShew) does not change the runtime behavior and results in a negligible speed penalty compared to fistFp: fistFp averages $0.155 \cdot n \log n$ microseconds, while fistShew averages $0.157 \cdot n \log n$ microseconds. All MPFR-based versions are about 24 times slower than fistFp on average, with fistMp212 averaging $3.786 \cdot n \log n$ microseconds, while fistCore is about 50 times slower. Increasing the precision requested for the MPFR-based versions causes no significant speed penalty. It is noteworthy, though, that the runtime of all MPFR-based versions varies much more significantly than for fistFp, fistShew, and fistCore.

In a second test we examined that numbers of data sets that were triangulated differently by the six variants of FIST. As it could be expected, there is no difference between fistFp and fistMp53. It is also not surprising that the use of exact predicates will cause some differences, despite the fact that considerable efforts were put into making FIST reliable. The difference between fistFp (fistMp53, resp.) and fistShew is small, though: Only 0.34 percent of the inputs were triangulated differently by fistShew. Contrary to our initial assumption, using MPFR with larger precisions in conjunction with FIST does not form an intermediate step between fistFp and fistCore: fistCore triangulates 10.38% of the inputs differently, while the outputs of fistMp212 and fistMp1000 deviate for 10.55% respectively 10.44% of the inputs.

3.3.2 Verification of Triangulations Computed

Recall that different outputs need not indicate different or even incorrect triangulations. Since the number of differences was too large to be analyzed by hand, we wrote a code for verifying triangulations. A polygonal area is considered to be triangulated correctly only if the segments of the triangles neither intersect each other nor intersect the segments of the polygonal area. Note that we also consider triangulation edges that coincide (partially) with edges of the polygon or that pass through vertices of the polygon – termed "overlay" problems – as errors. Additionally, we check whether any segment of the triangulation passes outside of the polygon.

We use the Bentley-Ottmann algorithm [4] to find intersections. As arithmetic back-end, we use exact arithmetic based on the mpq_t data type provided by the GMP package (GMP 5.0.1, [2]). In an attempt to cut down the verification efforts, and lacking better means



Figure 1: Run-time plots for fistFp, fistMp212, and fistCore. The y-axis corresponds to the run-time divided by $n \log n$, where n is the number of vertices shown on the x-axis.

for establishing reference triangulations, we simply assume that all CORE-based outputs correspond to correct triangulations. Under this assumption it suffices to check all outputs that differ from outputs of fistCore.

In our first attempt to check the triangulations our GMP-based verifier took the input coordinates as exact values. (That is, 0.1 was regarded as 1/10.) We were shocked to learn that the verifier reported about 4.92% of the triangulations to be faulty — uniformly for all variants of FIST which are not based on CORE. A more detailed analysis revealed that only fistFp, fistShew, and fistMp53 suffered from genuine intersection problems in their outputs, whereas the outputs of fistMp212 and fistMp1000 contained only overlay problems.

Still, nearly 5% faulty triangulations seemed too bad to be true. We implemented a viewer that does not suffer from floating-point errors and examined a few faulty triangulations: The errors reported by our verifier were only visible in the viewer when we used a zoom factor of at least 10^{15} , which is approximately the reciprocal value of the precision of double precision fp-numbers.

So we switched to using the double-precision fpapproximations of the input coordinates as the true input numbers for our verifier, and again tested the triangulations reported to be faulty for fistFp: This test revealed no faulty triangulation at all. We conclude that the errors found with the first version of the verifier were only caused by input errors, i.e., by the small differences between real numbers and their fp-approximations. Since all real-world applications of FIST that we are aware of are based entirely on fpnumbers, triangulations computed by fistFp can rightfully be assumed to be correct from the point of view of these applications.

3.4 Test Results for Voronoi Diagrams of Segments

In the sequel, we report on tests of four variants of VRONI – VRONI based on fp-arithmetic (vroniFp), and VRONI based on MPFR with precisions 53, 212, 1000 (vroniMp53, vroniMp212, vroniMp1000) – and compare them to the Voronoi code shipped with CGAL 3.8.

We tested three variants of CGAL's Voronoi code. The first variant is fully based on doubleprecision fp-arithmetic (cgvdFp), the second variant uses CGAL::Quotient<CGAL::MP_Float> as predicate kernel (cgvdQu)and the third uses CORE::Expr as predicate kernel (cgvdEx). All variants use Segment_Delaunay_graph_filtered_traits_2 template parameter to the underlying segment Delaunay graph class.

To ensure that CGAL and VRONI work on precisely the same input, we scale the input data to fit into the unit square as it is done per default in VRONI. For the same reason we add four dummy points outside of the bounding-box of the input explicitly to the input data for CGAL, at the positions specified by VRONI. (These points guarantee that each Voronoi cell of the actual input is bounded.) In order to ensure that the performance of CGAL is not influenced by file I/O we parse an input file and store the data in an intermediate data structure. Then we call the insert() method on the segment Delaunay graph, and construct the Voronoi diagram object based on the segment Delaunay graph. Only the insertions and the construction of the segment Delaunay graph are timed in our tests.

Our test bed consists of 18787 input files with polygons, polygon areas, and polygonal chains. In order to ensure that all tests could be carried out within an acceptable time period we considered only inputs with at most 100000 segments and limited the cpu-time consumption to 30 minutes per input. All tests were conducted on an Intel i7 CPU X 980 clocked at 3.33GHz. The test machine was equipped with 24 GiB of main memory.

The run-time performances of the variants tested are shown in Fig. 2. As expected, vroniFp is by far the fasted variant, averaging about $0.6 \cdot n \log n$ microseconds for inputs with 2000 or more segments, while the MPFR-based variants all are 50–70 times slower. However, the variation of the run-time for different inputs of the same size is much smaller for the different variants of VRONI than for CGAL: Once the input size is large enough to make the timing reliable for VRONI there are



Figure 2: Run-time plots for vroniFp, vroniMp212, and cgvdEx. The y-axis corresponds to the run-time divided by $n \log n$, where n is the number of vertices shown on the x-axis.

few data points outside of a small and dense band. On the contrary, the run-times of all CGAL variants vary by at least a multiplicative factor of 20. On average, CGAL performs slightly better than the MPFR-based variants of VRONI, but due to the big variance, there are several data sets that cause CGAL to perform worse than MPFRbased variants of VRONI. Interestingly, switching CGAL entirely to fp-arithmetic speeds up the code by only a factor of 1.5 on average. In any case, vroniFp completely outperforms CGAL on all data sets, being roughly 50–80 times faster than the CGAL variants. For about 0.36% of the inputs the CGAL variants did not finish within the limits imposed on runtime and memory.

The multiple outliers in the CGAL plots made us wonder whether there exist inputs for which the run-time complexity is worse than $O(n \log n)$. And, indeed, specific tests showed that smooth polygonal approximations of elliptical arcs or some free-form curves may cause all CGAL variants tested to consume $\Omega(n^2)$ time, with and without exact kernel, with and without filtered traits. (Unfiltered traits cause CGAL's built-in consistency checker to declare the solution as invalid quite frequently, which we also confirmed by our tests.)

Since the CGAL code was not designed to be run with conventional fp-arithmetic it is not surprising that cgvdFp fails frequently with fp-exceptions. (It crashed on 937 inputs.) It is more surprising, though, that the smallest input on which cgvdFp crashed also results in cgvdQu and cgvdEx computing Voronoi nodes which are numerically clearly wrong.

This observation made us wonder whether we had stumbled upon some isolated bug in CGAL, and we started to investigate the numerical accuracy of the Voronoi diagrams computed. We use an output format that stores the coordinates of the input sites and the coordinates of the Voronoi nodes calculated along with a reference to the defining sites. (We do not unscale the coordinates as this would introduce an unnecessary source of error.) We use a decimal precision of 60 digits in the output format. For each node, our verifier

• calculates differences in the distance to the defining sites of the node as the squared minimum distance divided by the squared maximum distance;

• checks whether any site *s* is closer to the node than its closest defining site, and reports the squared distance to *s* divided by the squared minimum distance.

In order to avoid introducing new errors in the verifier, all distance computations are based on GMP's mpq_t data type. Since brute-force all-pairs distance computations are used, we could afford to run our verifier only on comparatively small inputs with up to 2000 segments.

Due to the use of fp-numbers in the output files of CGAL and VRONI we cannot expect any variant to have no error – but we can hope for small errors. The larger the error, the more such a ratio differs from one. For each variant all ratios different from one are sorted in increasing order and the square roots of the sorted ratios are subtracted from one. (These computations are based on standard fp-arithmetic.) In order to avoid plotting millions of points, we compute the averages of groups of 100 consecutive error values (in the sorted order) and plot the resulting errors.



Figure 3: Error values that correspond to inconsistent distances to the defining sites of a node.

Figures 3+4 depict these sorted sequences of error values for vroniMp212 (lowest, red curve), vroniFp (middle, green curve), and cgvdQu (top, blue curve). Fig. 3 shows the errors that correspond to inconsistent distances to the defining sites of a node, and Fig. 4 shows the errors that represent violations of clearance disks by other sites.



Figure 4: Error values that represent violations of clearance disks by other sites.

It is evident and not surprising that VRONI runs into numerical errors, and that using MPFR clearly helps VRONI to improve the accuracy of its output. But it is surprising to learn that all VRONI variants produce outputs which, on average, seem to be numerically much more accurate than any of the CGAL variants: vroniFp shows significantly fewer and smaller errors than cgvdQu. The numerical errors of cgvdEx are even more severe than those of cgvdQu, and cgvdFp is completely unreliable, given the large number of crashes observed.

Hence, if the numerical accuracy provided by the standard floating-point arithmetic is deemed insufficient in a real-world Voronoi application then it seems advisable to use VRONI in conjunction with MPFR, given the current state-of-the-art of Voronoi implementations. However, we note that the exact CGAL variants might determine correct Voronoi topologies even though the Voronoi nodes are less accurate than what can be achieved on a standard fp-arithmetic. (We had no means to assess and check this quality criterion.)

3.5 Test Results for Straight Skeletons

Due to lack of space we summarize our results for the computation of straight skeletons as follows: the MPFR-based versions of BONE are about 10-20 times slower than boneFp, with boneFp averaging about $30 \cdot n \log n$ microseconds on our test platform. CGAL's straight-skeleton code exhibits both a quadratic run-time as well as a quadratic memory consumption and, thus, is only feasible for very small inputs.

4 Conclusion

In this paper we discussed the problems that arose when we attempted to interface FIST, VRONI and BONE with the MPFR library or the CORE library. While MPFR was fairly easy to integrate into our C/C++ codes, the integration of CORE required significantly more efforts and non-trivial changes to the codes. Furthermore, the CORE-based version of FIST suffered from a substantial performance hit. Our tests suggest that the MPFR library should be considered if the numerical precision of a geometric code such as VRONI is of concern: It does indeed succeed in boosting the precision without causing the increase in runtime to become completely unbearable. As discussed, the numerical precision of the output of a geometric code may depend substantially on the compiler settings.

To our surprise, FIST run on a standard floating-point arithmetic performed flawlessy: all triangulations computed by FIST were correct. Also to our surprise, the CGAL alternatives to VRONI and BONE hardly are an alternative in practice; we observed a tremendously increased runtime (compared to VRONI and BONE) and a decreased numerical precision of the output.

Of course, our tests constitute case studies for only three specific applications with a small number of codes. Hence, generalizations of our findings need not be legitimate without probing the grounds. In particular, the mere fact that CGAL performed poorly in our test applications cannot be construed as an indication for a general weakness of CGAL for other applications.

References

- [1] CORE. http://cs.nyu.edu/exact/core_pages/.
- [2] GMP. http://gmplib.org/.
- [3] MPFR. http://www.mpfr.org/.
- [4] J. Bentley and T. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.
- [5] CGAL. http://www.cgal.org/.
- [6] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 30(4):563–596, Aug 2001.
- [7] M. Held. VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments. *Comput. Geom. Theory* and Appl., 18(2):95–123, Mar 2001.
- [8] M. Held and S. Huber. Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight-Line Segments. *Comput. Aided Design*, 41(5):327–338, May 2009.
- [9] S. Huber and M. Held. Theoretical and Practical Results on Straight Skeletons of Planar Straight-Line Graphs. In Proc. 27th Annu. ACM Sympos. Comput. Geom., pages 171–178.
- [10] IEEE. IEEE 754-1985, Standard for Binary Floating-Point Arithmetic, 1985.
- [11] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Vol. 1, April 2011. http: //www.intel.com/products/processor/manuals/.
- [12] J. Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete Comput. Geom.*, 18(3):305–363, Oct 1997.

On Inducing *n*-gons

Marjan Abedin^{*}

Ali Mohades^{*}

Marzieh Eskandari[†]

Abstract

In this paper, we establish a lower bound on the number of inducing simple n-gons in grid-like arrangements of lines. We also show that the complexity associated with counting the number of inducing n-gons in an arrangement of collinear segments is #P-complete.

1 Introduction

Arrangement of lines in the plane is among the most studied structures in combinatorial and computational geometry. Consider an arrangement of n lines. An *inducing n-gon* is a simple polygon with n sides such that extension of each side induces a line of the arrangement, and extensions of all sides induce the whole arrangement. It means that each line in the arrangement should exactly contain one side of the inducing n-gon.

An interesting question is to find out whether an arrangement includes a simple n-gon inducing the whole arrangement [3], and a more appealing question is to find an upper or lower bound on the number of these n-gons that an arrangement can tolerate [2]. The first question has been responded affirmatively for simple arrangements [1, 5] while the second one still remains open. In addition to the above problems, the complexity of counting inducing n-gons in arrangements is another appealing issue to those interested in complexity theory.

In this paper, we establish a lower bound on the number of inducing n-gons in a grid-like arrangement of lines as defined formally in Section 3. This class of arrangements is interesting because despite the well-shaped appearance of the arrangements it seems to be hard to count all the inducing n-gons.

Inducing n-gons are discussed in arrangements of lines and pseudo-lines [3]. In this work, the complexity of counting these n-gons in arrangements of collinear segments is studied.

This paper is organized as follows. In Section 2, we present a method to count a subset of inducing n-gons in a special class of arrangements. The results of Section 3 are utilized to present a lower bound on the number of inducing n-gons in grid-like arrangements of lines.

In Section 4, it is shown that the complexity of counting the number of inducing n-gons in arrangements of collinear segments is #P-complete.

2 Arrangements of *n* lines with at least factorial number of inducing *n*-gons

This section is concentrated on a specific class of arrangements of n lines, where n = 3m and m is an integer. We present a method to show that an arrangement in this class contains at least factorial number of inducing n-gons.

2.1 Initialization

Consider an arrangement of 3m lines arranged in three sets of m parallel lines. Call the sets R, L and B, and consider the set B to be horizontal. The lines of each set are parallel to one side of an empty hypothetical triangle with horizontal base. Label the lines of each set in an ascending order, from the inner to the outer line. The intersection point between two lines is denoted by the names of those lines, x_{b_i,r_j} for the intersection of the lines b_i and r_j .

Intersections of any triple of lines, each one from different set, forms a triangle. We shall regard the area inside the biggest triangle as arrangement-core, and denote the triangles formed by the intersection of b_1 , r_i and l_i as T_i . Call all the segments on l_i s and r_i s of T_i s mountain range-LR. Similarly, all the segments on b_i s and l_i s of T_i s and likewise all the segments on b_i s and r_i s of T_i s are designated. We only explain the method for the mountain range-LR because of the symmetrical appearance of the arrangement.

In each mountain range, there are m mountains and m-1 narrow corridors, which play a fundamental role in the following section. Let M_i denote the *i*th mountain, and label the mountains and corridors from the inner to the outer one in an ascending order. Half of each corridor is on the right and the other half is on the left side of the arrangement; see Fig. 1(a). There is a specific edge in all the *n*-gons constructed by the method, call it ceiling-edge. Depending upon the parity of m, the ceiling-edge is defined differently that is described in the following section.

^{*}Laboratory of Algorithms and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of Technology, {m.abedin,mohades}@aut.ac.ir

[†]Department of Mathematics, Alzahra University, eskandari@alzahra.ac.ir



Figure 1: (a) The second corridor is filled with gray and M_4 is bolded. (b) An inducing 12-gon.

2.2 The method

All the inducing *n*-gons constructed by the method, contain one of the three mountain ranges. Through contribution of the segments on the mountain range-LR, there are 2m lines induced. Therefore, we extend the selected mountain range and close each corridor with a segment on a line of *B* to induce all the lines and obtain an inducing *n*-gon.

Assume *m* is even. The ceiling-edge for even *m* is a segment of a line of *B*, e.g. b_z , with two endpoints x_{b_z,r_1} and x_{b_z,r_m} or two endpoints x_{b_z,l_1} and x_{b_z,l_m} . Because of the symmetry, let us consider the ceiling-edge with two endpoints x_{b_z,r_1} and x_{b_z,r_m} on the right side, on b_z $(z \ge \lceil \frac{m}{2} \rceil)$. Lemma 1 explains why it is necessary that the ceiling-edge lies on a line of *B* with an index greater than $\lceil \frac{m}{2} \rceil$.

Having fixed the ceiling-edge, extend the right segments of M_1 and M_m to reach it. As the ceiling-edge is placed on the right, extend $\frac{m}{2} - 1$ of the right halfcorridors, and close each one of them with a segment on a so far unused line of B above b_z . Furthermore on the other side, extend $\frac{m}{2}$ of the left half-corridors and close each one with a segment on a remaining unused line of B. As presented in Fig. 1(b), the ceiling-edge is on b_3 and the extended corridors are closed on b_2 , b_1 , b_4 respectively. In summary, the even half-corridors are closed on the right, and the odd half-corridors are closed on the left alternatively starting from the first left half-corridor.

In conclusion, select m-z odd corridors from the left to be extended bellow b_z then there are (m-z)! possible choices for them to be closed on the left. Similarly, for the remaining z-1 half-corridors on both left and right, there are (z-1)! different configurations to be extended up to above b_z and to be closed. This method uses each line exactly once and all the inducing *n*-gons are different.

There are $\binom{m}{2}(m-z)!(z-1)!$ number of inducing *n*-gons by fixing the ceiling-edge on b_z $(z \ge \lceil \frac{m}{2} \rceil)$. Therefore, our method constructs $\sum_{m=1}^{m} \binom{m}{m-z}(m-z)!(z-1)!$ number of different inducing *n*-gons by taking all the possible places for the ceiling-edge into account, $z \ge \lceil \frac{m}{2} \rceil$. Disregarding the algebraic simplification, the

result equals $\left(\frac{m}{2}\right)!\left(\frac{m}{2}-1\right)!\left[\left(\frac{m}{2}\right)-1\right].$

The discussed points for even m are also true for odd m with some changes:

- The ceiling-edge is a segment on b_z , $z \ge \lceil \frac{m}{2} \rceil$, with two endpoints x_{b_z,l_1} and x_{b_z,r_m} or two endpoints x_{b_z,r_1} and x_{b_z,l_m} . Because of the symmetry, consider the ceiling-edge with two endpoints x_{b_z,l_1} and x_{b_z,r_m} on the right side of the arrangement. Then it is necessary to extend the left segment of M_1 and the right segment of M_m to reach the ceiling-edge, and the right half-corridors should be extended and closed above the ceiling-edge to avoid crossing it.
- Since *m* is odd, extend and close $\frac{m-1}{2}$ of the even corridors on the left, and do the same for $\frac{m-1}{2}$ of the odd corridors on the right alternatively starting from the first right corridor. According to the position of the ceiling-edge, whether it is on the left or on the right, some of the corridors should be closed above the line containing the ceiling-edge, and the remaining ones ought to be closed below it.

The lower bound presented for even m is also true here.

Lemma 1 The ceiling-edge has to lie on b_z , $z \ge \lceil \frac{m}{2} \rceil$.

Proof. By contradiction, while closing some corridors on the left or right, there would be an intersection among the ceiling-edge and the extended corridors. Therefore, there are more than one side of an inducing polygon on the ceiling-edge or other lines, and of course the result is not an inducing n-gon. It is also possible to have some self-intersections.

2.3 Generalization of the arrangements

The presented lower bound in the previous section is preserved for generalized arrangements of n lines, where n = km and k and m are both integers. Lines in the arrangements are divided into k sets of parallel lines, each set of size m. In addition, each pair of sets are intersecting, and the lines of each set are parallel to one side of an empty hypothetical k-gon. The arrangement-core for a generalized arrangement is the limited space with the exterior line of each set. To take advantage of the benefits attributed to the method discussed in Section 2.2, each set of lines should only intersect its adjacent sets inside the arrangement-core, and the intersections with other sets lie outside of this area.

It is important to note that the method in the previous section presents a lower bound on the number of inducing n-gons inside the arrangement-core, which indicates that the method ignores the intersections beyond this region. This is an observation which is used to establish a lower bound on the number of inducing n-gons in the following section.

3 Inducing *n*-gons in grid-like arrangements

A grid-like arrangement is an arrangement with two sets of parallel lines. If the numbers of lines in two sets are not equal, then there is no inducing n-gon. Otherwise the numbers of horizontal and vertical sides of inducing n-gons are not equal, and this cannot happen. Thus, consider a grid of size n with two sets of parallel lines each one of size m, where m is an integer. Without loss of generality, consider the lines of one set parallel to the x-axis and those of the other set parallel to the y-axis.

An obvious lower bound on the number of inducing n-gons in the grid is (m - 1)!. This amount is obtained by considering the biggest bounded segment on the bottommost line as a fixed edge, closing each vertical corridor with a segment on a horizontal line and finally joining them with vertical segments on the vertical lines. The inducing n-gons obtained by this method, form monotone orthogonal n-gons in the direction of the x-axis. The former results in Section 2.2 are utilized to improve this bound.

Consider an arrangement of 2kz lines, the same as the arrangements described in Section 2.3, where k is a divisor of m and z is an integer. The arrangement contains 2k sets of parallel lines each one of size z. The 2ksets are arranged such that the arrangement-core forms a 2k-gon, in which each set intersects only its adjacent sets. So it can be concluded that the segments of nonadjacent sets, inside the arrangement-core, are parallel because they do not intersect each other. It is a transformation of the grid to an arrangement of segments bounded with the arrangement-core. In other words, the whole m lines in one set of the grid are divided into k sets in such a way that each set contains $\frac{m}{k}$ lines. These k sets are arranged parallel to the non-consecutive sides of a hypothetical 2k-gon.

Although a grid contains extra intersections in comparison with the arrangement of segments inside the arrangement-core, simply ignore those additional intersections; see Fig. 2(a). In other words, it is a lower bound on the number of inducing *n*-gons where the *n*-gons do not bend on the extra intersections; see Fig. 2(b).

Based on the above discussion, the presented lower bound in Section 2.2 is also true for grids. As there is no overlap between inducing *n*-gons obtained by the two methods, the lower bound is equal to $(m-1)! + \sum_{k \in K} \left(\frac{m}{2k}\right)! \left(\frac{m}{2k} - 1\right)! \left[\left(\frac{m}{k}\right) - 1\right]$, where *K* is the set of all divisors of *m* which are greater than two. Note that for a fixed *k*, the ceiling-edge contains exactly k-1segments. Therefore, different *k* do not lead to identical *n*-gons while there is at least one difference between their ceiling-edges. This bound can become more precise by taking inducing *n*-gons in other directions into account although we ignore them.



Figure 2: (a) Arrangement of segments inside the arrangement-core and the related grid, extra intersections are removed. (b) The related inducing n-gons.

4 Complexity of counting inducing *n*-gons in an arrangement of collinear segments

An arrangement of collinear segments, ACS, is a collection of line segments in the plane. The arrangement includes some maximal subsets of collinear segments, i.e. a maximal subset *family*. A single segment can also be a family if there is no other segment collinear with it. Note that an arrangement of lines is a special case for this arrangement, as there are n families in an arrangement of n lines.

An inducing n-gon in ACS is a polygon with n sides for which there is a bijective relation between its sides and the families. It means that each family of ACS should contain exactly one side of the inducing n-gon. Obviously, if there are less than n intersections between the families of ACS, there is no inducing n-gon.

The class #P contains all counting problems associated with the polynomial-balanced and polynomial-time decidable relations [4]. As our problem satisfies these two properties, it is in #P. We demonstrate that counting the number of inducing *n*-gons in an arrangement of collinear segments is #P-complete. Let us reduce the #P-complete problem #RPM to #n-IP, where #RPMis the number of perfect matchings in a regular bipartite graph, and #n-IP is the number of inducing *n*-gons in an ACS.

Theorem 2 Complexity of counting #RPM in a k-regular bipartite graph is #P-complete, for any fixed k > 2 [6].

Given a k-regular bipartite graph G = (U, V, E) such that |U| = |V| = m. The goal is to construct an arrangement of collinear segments such that the inducing n-gons in ACS somehow correspond to the perfect matchings in G. Consider some guide-lines which form a (m + 1) times (m + 1) grid-like arrangement A, and also suppose m vertical guide-segments such that each segment is limited to the second bottommost guide-line. Each guide-segment is placed inside a vertical corridor of A and divides it into two vertical corridors, a small

corridor and a big corridor. For each node in U and V consider a big corridor and a horizontal guide-line of A respectively. See Fig. 3(a) and consider the following segments:

- *Main-segments*: Bounded horizontal segments with big corridors.
- *Small-segments*: Bounded horizontal segments with small corridors.
- *Helping-segments*: The segments on both guidesegments and vertical guide-lines bounded between the second bottommost horizontal guide-line and the topmost small segment.
- *Final-segments*: The biggest segment on the bottommost guide-line and the connecting segments of its endpoints to the left and right most helpingsegments.

The reduction is as follows. For each edge in G which connects u_i to v_j , add a main-segment inside the big corridor associated with u_i and lies on the guideline attributed to v_j , $1 \leq i, j \leq m$. Add m small-segments inside of each small corridor. Put one of the small-segments above the topmost horizontal guide-line, and each horizontal corridor of A should contain exactly one small-segment except the bottommost corridor. The small-segments in each horizontal corridor of A should be collinear, to form horizontal families. Add the helping-segments and the final-segments; see Fig. 3(b) as overall arrangement. We claim that #n-IP is equal to $m! \ \#\text{RPM}$, where n = 4m + 2. It can be shown by the following lemma.

Lemma 3 All the inducing n-gons in the designed ACS are monotone in the direction of the x-axis.

Proof. By contradiction, there are more than one side of the polygon on at least one of the helping-segments. \Box

In each inducing *n*-gon there are exactly *m* mainsegments. There is no pair of main-segments selected in a big corridor as an inducing *n*-gon in the constructed ACS is monotone, Lemma 3. This point indicates that there is no pair of vertices in *V* matched with a vertex in *U*; see Fig. 3(c).

To obtain an inducing n-gon, the main-segments are joining via small-segments, helping-segments and finalsegments. For a fixed set of selected main-segments in an inducing n-gon, related to a perfect matching of G, there are m! possible ways to choose small-segments to obtain m! different inducing n-gons.

The reduction is now complete and obviously is in P. It is to mean that if someone can obtain the #n-IP efficiently, he could easily find the #RPM which is a #P-complete.



Figure 3: (a) On the left, a 3-regular bipartite graph. On the right, dash lines/segments are as guidance and the related main-segments and small-segments to the graph are bolded. (b) The main-segments, small-segments, helping-segments and final-segments are blue, violet, orange and green respectively. (c) On the left, a perfect matching. On the right, the designed arrangement of segments and a related inducing n-gon to the perfect matching is bolded.

5 Conclusion

We establish a lower bound on the number of inducing *n*-gons in grid-like arrangements. It is interesting to find the exact number of inducing *n*-gons in this class of arrangements. We also demonstrate that the complexity of counting the number of inducing *n*-gons in an arrangement of collinear segments is complete for the class #P. We conjecture that the complexity of counting inducing *n*-gons in arrangements of lines is also #Pcomplete.

6 Acknowledgement

The authors would like to appreciate Fatemeh Zare, Farnaz Sheikhi, Mansoor Davoodi, Zahra Liaghat and Amin Gheibi for their patience and helps throughout writing this paper.

References

- E. Ackerman, R. Pinchasi, L. Scharf and M. Scherfenberg. Every simple arrangement of n lines contains an inducing simple n-gon. The American Mathematical Monthly, 118(2): 164-167, 2009.
- [2] E. Ackerman, R. Pinchasi, L. Scharf and M. Scherfenberg. On Inducing Polygons and Related Problems. Computational Geometry: Theory and Applications (CGTA), 5757: 47-58, 2009.

- [3] P. Bose. Properties of arrangement graphs. The Journal of Computational Geometry and Applications, 13(6): 447-462, 2003.
- [4] C. H. Papadimitriou. Computational Complexity. University of California, Addition-Welsey Publishing Company, 1994.
- [5] L. Scharf and M. Scherfenberg. Inducing n-gon of an arrangement of lines. In 25th European Workshop on Computational Geometry, Bruxells, Belgium, 129-132, 2009.
- [6] S. P. Vadhan. The Complexity of Counting in sparse, Regular, and Planar Graphs. SIAM Journal on Computing (SICOMP), 31(2): 398-427, 2001.

Weak Matching Points with Triangles

Fatemeh Panahi*

Ali Mohades^{*}

Mansoor Davoodi*

Marziyeh Eskandari[†]

Abstract

In this paper, we study the weak point matching problem for a given set of n points and a class of equilateral triangles. The problem is to find the maximum cardinality matching of the points using equilateral triangles such that each triangle contains exactly two points and each point lies at most in one triangle. Under the non-degeneracy assumption, we present an $O(n^{3/2})$ time algorithm using the TD-Delaunay graph and a graph matching algorithm. Also, we show that the lower bound for the number of matched points is $\lfloor 2n/3 \rfloor$ which is optimal in the worst case.

1 Introduction

The point matching problem is a challenging problem in computational geometry and graph theory and has many applications in geometric shape matchings and computational biology [3]. The problem of point matching with planar geometric objects, recently studied in [1], is a special variant of point matching problems. Given a set P of points in the plane and a class C of 2D geometric objects, the problem is to find a set of C-type objects, called C-matching of P, in which each object contains exactly two points of P and each point lies in at most one object. The problem is a generalization of geometric graph matching where the objects are segments. Alternatively, what we refer to as objects can be circles, squares or rectangles as well.

Assume that the number of points is even. A Cmatching is called *perfect* if all points in P are covered, and it is *strong* if the matched geometric C-objects are non-overlapping. In addition, the matching is called *weak* if we do not know whether it is strong [4]. Álbrego et al. studied properties of C-matching problem for two classes of circles and isothetic squares in perfect and strong matching [1]. Assuming the class of objects to be circles, they proved some bounds for the cardinality of matching in strong and/or perfect matching. The weak perfect matching problem for line segments was studied by Rendl and Woeginger [12]. They proposed an $O(n \log n)$ time algorithm for orthogonal segments, where n is the number of points in P. They proved that the problem is NP-complete if the segments are not allowed to cross. Aloupis et al. investigated matching problems for non-crossing objects [3]. They showed that the problem is NP-complete for lines and line segments in general, but polynomial-time when segments form a convex polygon. Also, a bichromatic version of the problem and a non-intersecting constraint have been studied for strong matching when the objects are segments by Dumitrescu and Steiger [8] and Kaneko and Kano [9], respectively.

Albrego et al. studied the matching problem for circles and squares [1], [2]. Under the non-degeneracy assumption, they showed that there always exists a weak perfect matching for the class of axis-aligned square objects, and proposed a $2\lceil n/5 \rceil$ bound for the cardinality of matching for the strong one. They presented a 2[(n-1)/8] bound for circles, as well. The classes of rectangles and squares have been studied by Bereg et al. [4]. Without the general position assumption, they proposed an $O(n \log n)$ optimal time algorithm for squares in the weak matching realization and an $O(n^2 \log n)$ time algorithm for the strong one. Also, they showed that a weak rectangle matching of maximum cardinality can be computed in $O(\beta n^{1.5})$ time, where β is the minimum of the number of different xcoordinates and the number of different y-coordinates in P. In addition, they proved that there exists an optimal worst case |2n/3| cardinality of matching for axisaligned rectangles in the strong matching and proved that the problem of determining whether a given set of points has a perfect strong matching is NP-hard for the class of squares.

In this paper, we study the problem of weak point matching using equilateral triangles with a horizontal base which lies below its non-adjacent vertex. We denote this problem of Weak Triangle Matching by WTM. The approach that we present is also applicable for homothets of any fixed triangle, by applying a shear transformation. To solve the problem, we use a *shrinkability* property [2] and reduce WTM to a graph matching problem. When two points of P named p and q are matched in a solution to a matching problem, a C-type object contains exactly p and q. Thus, the object can be shrunk such that p and q lie on its boundary. This property is called "shrinkability" of geometry object matching. Having this property, we reduce the problem of

^{*}Laboratory of Algorithms Computational and Department Geometry, of Mathematics and Computer Science, Amirkabir University of Technology, {fatemehpanahi,Mohades,mdmonfared}@aut.ac.ir,

 $^{^\}dagger Department$ of Mathematics, Alzahra University, Tehran, Iran. <code>eskandari@alzahra.ac.ir</code> ,

matching with geometric objects to a graph matching problem. The corresponding graph for the WTM problem is similar to a Θ_k -graph which has been used in the geometric spanner context [11]. Indeed, the graph is a special form of a 2-spanner, Θ_6 -graph, introduced by Bonichon et al. [6] and called half- Θ_6 -graph [7]. They proved that the half- Θ_6 -graph is the same as triangulardistance Delaunay graph and can be computed in optimal $O(n \log n)$ time for a set of n points in the plane.

In the next section, we propose an $O(n^{3/2})$ time algorithm for finding the maximum-cardinality matching for the WTM problem. Later in section 3, we will show that the number of matching points with our proposed algorithm will at least be $\lfloor 2n/3 \rfloor$ points for every given point set, which is optimal in the worst case.

2 Weak Point Matching With Equilateral Triangles

The problem of matching with geometric objects has been studied for classes of segments, circles, squares and rectangles. It would be interesting to study the same for convex polygons as well. In this paper, we study equilateral triangles. For the class of arbitrary triangles, the problem will be trivial, because each segment can be assumed to be a triangle with a height sufficiently small. We consider the x-axis aligned equilateral triangles. They are equilateral triangles, one of the edges of which is parallel to the x-axis. We assume that the triangle is located above this edge.

For both strong and weak versions of the problem, there are counterexamples that show a perfect triangle matching does not always exist. But we show in this section that there is an $O(n^{3/2})$ time algorithm which can compute a weak triangle matching of maximum cardinality for a set of n points.

For a given set of points $P = \{p_1, p_2, \ldots, p_n\}$, the problem of weak triangle matching called WTM is to find a set of x-axis aligned triangles such that each triangle includes exactly two points of P. Fig. 1 shows two solutions of the WTM problem for a set of eight points.



Figure 1: An example of the WTM problem and two distinct solutions for it (dashed triangles and solid triangles).



Figure 2: The three directions d_1 , d_2 and d_3 and the cones in the covering of a point.

Throughout this paper, we consider three axes d_1 , d_2 and d_3 which have angles of $\pi/6$, $5\pi/6$, and $9\pi/6$ with x-axis, respectively. We assume that the points of the set P are in general position, which as we define it, means that there are no two points with the same coordinates in the directions d_1 , d_2 or d_3 . Also, we denote the orthogonal projection of a point p onto d_i by $d_i(p)$, for i=1, 2 and 3. For a point p, we partition the plane into six regular cones with the apex p. see Fig. 2. The three odd cones with their bisectors being d_1, d_2 and d_3 will be denoted A_1 , A_2 and A_3 respectively; the remaining three will be called B_1 , B_2 and B_3 . We say that the point q is in the covering of p in the direction d_i , if it lies in A_i , for i=1, 2 and 3.

Let T be an axis-aligned equilateral triangle including p and q. We can shrink T to find a smaller such triangle so that p and q lie on its boundary. In addition, for the smallest covering x-axis aligned equilateral triangle at least p or q lies on one of its vertices. We denote such a triangle by T(p,q). Without loss of generality, we assume that each triangle which contributes to WTM has a point on one of its vertices and the other point is on its boundary. Also, for two points p and q in P, we say that T(p,q) is a *candidate triangle* for the weak triangle matching problem if it contains no other points of P. Letting p be a vertex of a candidate triangle, the other point should be in the covering of p. With regard to the general position assumption, we have the following observation.

Observation 1 Any point p in P can be a vertex of at most three candidate triangles.

To solve the WTM problem, we define a geometric graph and reduce the problem to a graph matching problem. To this end, we construct the geometric graph G(P) for a point set P. Vertices of G(P) are exactly the point set P, and there is an edge between two vertices pand q if and only if T(p, q) is a candidate triangle. Fig. 3 displays a point set and its corresponding geometric graph.

To compute the geometric graph, G(P), we can use the algorithm of Θ_k -graphs for k=6 [11]. This type



Figure 3: The geometric graph in the WTM problem for a set of points.

of graphs are the linear approximation of complete Euclidean graphs. Chew showed that the Delaunay triangulation using triangle distance function (called TD-Delaunay graph) is a 2-spanner graph [7]. To construct the TD-Delaunay graph it is sufficient to replace the empty equilateral triangle with the circle in the empty circle test in constructing the standard Euclidean Delaunay triangulation. Also, it is proved that the size of the TD-Delaunay graph is linear and can be computed using the sweep line approach in $O(n \log n)$ time for a set of n points. The final result in this context was presented by Bonichon et al. [6]. They introduced a specific subgraph of Θ_6 -graph, called the *half*- Θ_6 -graph, and proved that it is equal to the TD-Delaunav graph. Based on the mentioned concepts, we can conclude the following result.

Proposition 1 For a given set P of n points in the plane, the geometric graph G(P) is a connected graph with O(n) edges and can be computed in $O(n \log n)$ time.

Since an edge in G(P) corresponds with a candidate triangle in P, solving the problem in P is equal to finding the maximum graph matching in G(P). The maximum graph matching for a graph G = (V, E)can be solved using Micali and Vazirani's algorithm in $O(|V|\sqrt{|E|})$ time [10]. Taking into account the linear size of G(P), we conclude this section with the following theorem:

Theorem 2 For a set of n points in the plane, the maximum cardinality weak point matching with x-axis aligned equilateral triangles can be solved in $O(n^{3/2})$ time and O(n) space.

3 Lower Bound for the number of matched points for the WTM

In the previous section, we showed that there is an algorithm that finds a maximum cardinality matching for a given point set. In this section, we show that the weak triangle matching for the points in general position always covers at least $\lfloor 2n/3 \rfloor$ points. If the points are not in general position, the worst case is the one in which each point has the same coordinate as another point, in direction d_1 , d_2 or d_3 as illustrated in Fig. 4.



Figure 4: An example for a set of points which are not in general position.

In this case, only the extreme points can be matched. Without the general position assumption, the lower bound for the number of the points which can be matched in an arbitrary point set, P, with the cardinality of n is $O(\sqrt{n})$. If we assume that the points are in general position, the problem of finding the lower bound for the number of matched points with WTM becomes interesting. The following lemmas present some properties of the corresponding graph to find a lower bound.

Lemma 3 For each two vertices p and q in G(P), there are vertices r_1, r_2, \ldots, r_k $(k \ge 0)$ inside T(p,q) such that, the path $pr_1r_2 \ldots r_kq$ is between p and q and each r_i , $(1 \le i \le k)$ lies in T(u, v) where u and v are the adjacent vertices of r_i on the path $pr_1r_2 \ldots r_kq$.

Proof. If T(p,q) is a candidate triangle, there is an edge between p and q. So, the lemma holds for k=0. Otherwise, there is a vertex inside T(p,q), e.g. r_1 , which $T(p,r_1)$ is a candidate triangle and there is an edge between p and r_1 . For the vertices q and r_1 , if $T(r_1,q)$ is a candidate triangle, there is an edge between them. So, the lemma holds for k=1. Otherwise, similarly there is a vertex inside $T(r_1,q)$, e.g. r_2 , which $T(r_1,r_2)$ is a candidate triangle and there is a path between r_2 and q. Consequently, the path $pr_1r_2 \ldots r_kq$ lies inside T(p,q) and each r_i , $(1 \le i \le k)$ lies in T(u,v), where u and v are the adjacent vertices of r_i on the path.

Lemma 4 For an arbitrary point, q, consider the six mentioned cones, A_i and B_i , for i=1, 2, 3. If there are two points, p_1 and p_2 , such that q lies inside $T(p_1, p_2)$, then one of them, e.g. p_1 , cannot be in the covering of qand the other point, p_2 , cannot be in the cone containing p_1 and its two adjacent cones.

Proof. If both two vertices, p_1 and p_2 , are in the covering of q, then q cannot be inside $T(p_1, p_2)$, because there exists a line that separates q and $T(p_1, p_2)$. For example, if p_1 lies in A_1 and p_2 lies in A_2 , $T(p_1, p_2)$ completely lies above the horizontal line that passes through q. So, suppose that p_1 is not in the covering of q and lies in one of B_i cones, e.g. B_1 . If q lies inside $T(p_1, p_2)$,

then $d_3(p_2) > d_3(q)$ which implies that p_2 cannot be in B_1 or in its adjacent cones, A_1 and A_2 .

Let C(q) be the number of connected components which are created by removing a vertex q from G(P). We will have the following lemmas.

Lemma 5 For any vertex q in the corresponding graph of the point set P, G(P), $C(q) \leq 3$.

Proof. Consider the point q and its six mentioned cones. For contradiction, assume that there are at least four components after removing q, so there is a vertex in each component, e.g. p_1 , p_2 , p_3 and p_4 , which connect to q by an edge. See Fig. 5. According to lemma 4, for two points p_i and p_j , for $1 \le i, j \le 4$, if q is inside $T(p_i, p_j)$, there is at least one of the cones, A_1, A_2 or A_3 between p_i and p_j , otherwise, there is a path between them which does not pass through q. In this case, there are four vertices lying in 6 regions. So, there are at least two vertices which are in the same or two adjacent cones. This means that by removing q, there is a path between at least two vertices of p_i which does not contain q. It implies that these two vertices which are p_1 and p_4 in Fig. 5, cannot be in two disjoint components, after removing q, which would be a contradiction.



Figure 5: The vertices adjacent to q cannot be in more than three components.

Lemma 6 Suppose that the vertices $p_1, p_2, \ldots, p_{i-1}$ have been removed from G(P), and G'(P) be the resulted graph. If by removing a vertex, e.g. p_i from G'(P), more than two connected components are added, then there would be two vertices r and s connected to p_i , such that T(r, s) contains some vertex like q where $q \in \{p_1, p_2, \ldots, p_{i-1}\}$ and C(q) < 3 but T(r, s) does not contain p_i .

Proof. According to lemma 5, $C(p_i) \leq 3$. So, each two vertices adjacent to p_i which are in two disjoint components by removing p_i from G(P), the vertex p_i is inside the triangle of them. So, it is expected that removing p_i from G'(P) adds two connected components. Unless,

there are two vertices adjacent p_i like r and s such that T(r, s) does not contain p_i , furthermore, by removing p_i from G(P), the vertices r and s are in the same component, while by removing $p_1, p_2, \ldots, p_{i-1}, p_i$, vertices r and s are in two disjoint components. See Fig. 6. It means that in G'(P) there is no edge between r and s. According to lemma 3, there is a path with length of more than one between r and s, and the vertices on the path are inside T(r, s). This path is disjoint from sp_ir , because r and s are in two disjoint connected components. There should be a vertex on the path like q, which has been deleted before. According to lemma 3, q is inside T(u, v) where u and v are adjacent vertices of q on the path between r and s. The path between u and v passing through p_i , implies that the number of created components by removing q from G(P) cannot be three. So, C(q) < 3.



Figure 6: The adjacent vertices of p_i in lemma 6.

Suppose that we want to remove the vertices of a set from the corresponding graph, one-by-one. Note that, the sequence created by the number of added connected components by removing each vertex, varies with the order of removing vertices. For example, in Fig. 3, there are two possible orders for removing the two points, p_1 and p_2 . For these two removing orders, p_1, p_2 and p_2, p_1 , the sequences of the number of the added connected components are 1, 1 and 0, 2, respectively. It is clear that the total number of created connected components is independent of the removing order. The lemmas 5 and 6 show that by removing each vertex, at most two connected components are added, unless there is a vertex which has been removed before and the number of connected components created by removing it from G(P)is less than three. We are going to show that there is a vertex removing order, which guarantees at most two connected components are added by removing each vertex. The following lemma concludes this discussion.

Lemma 7 By removing the vertices of the set $S = \{p_1, p_2, \ldots, p_k\}$ from G(P), at most 2k + 1 connected components are created.

Proof. As we discussed before this lemma, different orders of removing vertices of S generate different sequences of the number of added components. However,
the total number of created components is the same. To prove the lemma, we show that there is an order for removing the vertices of S such that at most two connected components are added by removing each vertex. If such an order exits, the number of connected components by removing k vertices, will be at most 2k + 1. Let $Pr(p_i)$ be the priority of removing p_i . For any p_i and p_j in S, if $Pr(p_i) > Pr(p_j)$, we remove p_i before p_j . For each two vertices of G(P), p_i , p_j , if p_i has two adjacent vertices like r and s such that T(r, s) contains p_i , but not p_j , let $Pr(p_i) > Pr(p_j)$. See Fig. 7. Lemma 6 shows that if we follow this priority, in each step, at most two connected components will be added. A problem occurs when there is some vertex like p_t such that $Pr(p_t) > Pr(p_i)$ and $Pr(p_i) > Pr(p_t)$. It means that there is a sequence of vertices which have the cycle of priority. For solving this problem, consider all vertices of S which lie on such priority cycles. First, we remove the vertices which lie on more than one priority cycles. For example in Fig. 7, these vertices are p_t and $p_{t'}$. After removing such vertices, we arbitrarily remove one of the vertices on each of the priority cycles which have no common vertex with any other priority cycles. As these vertices are on a cycle of the graph, by removing them no connected components are added. After removing one of the vertices of the priority cycles, the priority of the other vertices on the priority circles, will become explicit. The other vertices of S will have the arbitrary priority. Since there is an order for removing the vertices of S such that at most two connected components are added by removing each vertex, the number of connected components created by removing k vertices is at most 2k+1.



Figure 7: The priority of removing p_i and p_j where $Pr(p_i) < Pr(p_j)$ and the priority cycles.

A basic condition for graphs that have a perfect matching was found by Tutte in 1947. Berge in 1958 observed that it implies a min-max formula for the maximum cardinality $\alpha(G)$ of a matching in a graph G, the *Tutte-Berge* formula. A connected component of a graph is called odd if it has an odd number of vertices. Let $C_o(G)$ denote the number of odd components of G. Then, based on Tutte-Berge formula [5], for each graph G = (V, E),

$$\alpha(G) = \min_{S \subset G} (|V(G)| + |S| - C_o(G - S))$$

Tutte–Berge formula and lemma 7 lead to find a lower bound for the number of matched points in WTM.

Theorem 8 Maximum cardinality of weak triangle matching for any set of n points in the plane in general position matches at least $\lfloor 2n/3 \rfloor$ points.

Proof. Let $|S| = k_S$ and G be the corresponding graph of P. According to lemma 7, $C_o(G - S) \le C_(G - S) \le 2k_S + 1$. Based on the Tutte–Berge formula

$$\alpha(G) = \min_{S \subset G} (|V(G)| + |S| - C_o(G - S)) \ge \min_{S \subset G} (n + k_S - 2k_S - 1) = \min_{S \subset G} (n - k_S - 1)$$

We consider two following cases:

• |S| < n/3 $M_1 = \min_{S \subset G} (n - k_S - 1) > n - n/3 - 1 > 2n/3 - 1 \Rightarrow$ $M_1 \ge 2n/3$

•
$$|S| \ge n/3$$

 $C_o(G-S) \le 2k_S + 1 \Rightarrow \forall S, \exists F_S \ge 0,$
 $C_o(G-S) = 2k_S + 1 - F_S,$
 $|S| + C_o(G-S) \le n \Rightarrow k_S + 2k_S + 1 - F_S \le n \Rightarrow$
 $3k_S + 1 - F_S \le n \Rightarrow F_S \ge 3k_S + 1 - n,$
 $M_2 = \min_{S \subset G} (|V(G)| + |S| - C_o(G-S)) =$
 $\min_{S \subset G} (n + k_S - (2k_S + 1 - F_S)) =$
 $\min_{S \subset G} (n - k_S - 1 + F_S) \ge$
 $\min_{S \subset G} (n - k_S - 1 + 3k_S + 1 - n) =$
 $\min_{S \subset G} (2k_S) \ge 2n/3$
Therefore,
 $\alpha(G) = \min(M_1, M_2) \ge 2n/3 \ge \lfloor 2n/3 \rfloor$

Fig. 8 depicts a point set P and its corresponding geometric graph which has n = 3k vertices. As illustrated in the figure, the triangles $T(r_i, s_i)$, $T(r_i, r_j)$ and $T(s_i, s_j)$, for $1 \leq i, j \leq k$, are not candidate triangles. Therefore, the candidate triangles are $T(r_i, m_i)$ and $T(s_i, m_i)$, for $1 \leq i \leq k$, and also, $T(r_i, m_{i-1})$, $T(s_i, m_{i-1})$ and $T(m_i, m_{i-1})$, for $2 \leq i \leq k$. Each edge has an end point at the central vertices, m_1, m_2, \ldots, m_k . Clearly, only one of the edges incident to m_i can be in a matching. It shows that the ratio of the points that can be covered by a maximum cardinality weak triangle matching is 2/3, so, the proposed lower bound is tight.



Figure 8: Set of n points which at most $\lfloor 2n/3 \rfloor$ can be matched.

4 Conclusion

The problem of matching points with classes of objects such as circles, squares and rectangles has been recently studied in computational geometry and graph theory. In this paper, we studied the weak point matching for the class of equilateral triangles as an open problem of previous studies. We showed that the maximum cardinality of this kind of matching can be computed using a convex distance function based on equilateral triangles. In addition, we discussed the lower bound of the size of weak triangle matching. We proved that for every point set, at least 2/3 of the points can be matched and we showed that this lower bound is tight. These results are also true for homothets of any fixed triangle. However, the time optimality of the algorithm remains as an open problem. Another future work is to study the strong version of the problem.

5 Acknowledgments

This research was started at the third Winter School on Computational Geometry organized by the members of the Laboratory of Algorithms and Computational Geometry of Amirkabir University of Technology. The authors thank the participants and the invited speakers: Michiel Smid and Helmut Alt for their lively disscusion.

References

- B. M. Ábrego, E. M. Arkin, S. Fernández-Merchant, F. Hurtado, M. Kano, J. S. B. Mitchell, J. Urrutia. Matching points with geometric objects: Combinatorial results. Proc. 8th Jap. Conf. Discrete Comput. Geometry , JCDCG04, Springer-Verlag, 2005.
- [2] B. M. Ábrego, E. M. Arkin, S. Fernández-Merchant, F. Hurtado, M. Kano, J. S. B. Mitchell, J. Urrutia. Matching points with squares. *Discrete and Computational Geometry archive*, 41(1):77–95, 2009.

- [3] G. Aloupis, J. Cardinal, S. Collette, E.D. Demaine, M.L. Demaine, M. Dulieu, R.F. Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Matching Points with Things. *JLNCS* 6034/2010, 456–467, 2010.
- [4] S. Bereg, N. Mutsanas, E. Wolff. Matching Points with Rectangles and Squares. *Computational Geome*try, Theory and Applications, 42: 93–108, 2009.
- [5] J. A. Bondy and U. S. R. Murty. Graph Theory with Applications. *American Elsevier Publishing, New York*, 1976.
- [6] N. Bonichon, C. Gavoille, N. Hanusse, D. Ilcinkas. Connections between Theta-Graphs, Delaunay Triangulations, and Orthogonal Surfaces. WG 2010, 266–278, 2010.
- [7] L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–21917, 1989.
- [8] A. Dumitrescu, W. Steiger. On a matching problem in the plane. *Discrete Math*, 211:183-195, 2000.
- [9] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the planer, a-survey. *Discrete and Computational Geometry*, 25:551–570, 2003.
- [10] S. Micali, V.V. Vazirani. An O(√|V|.|E|) algorithm for finding maximum matching in general graphs. in: Proc. 21st IEEE Symp. Found. Comp. Sci.(FOCS80), 17-27, 1980.
- [11] G. Narasimhan, M. Smid. Geometric Spanner Networks. *Cambridge University Press*, 2007.
- [12] F. Rendl, G. Woeginger. Reconstructing sets of orthogonal line segments in the plane. *Discrete Math*119:167– 174, 1993.

Index

Α

Abedin, Marjan 495 Abel, Zachary 77, 83, 89 Abu-Affash, A. Karim 39 Aichholzer, Oswin 21, 229 Alam, Md. Ashraful 169 Alon, Noga 285 Aloupis, Greg 229, 361 Anari, Nima 367 Arkin, Esther M. 163 Asano, Tetsuo 315

В

Ben-Moshe, Boaz 33, 399 Benbernou, Nadia M. 461 Berardi, Matthew 181 Bhosle, Amit 319 Biedl, Therese 291 Bisadi, Pouya 337 Bokowski, Jürgen 199 Bose, Prosenjit 93, 241 Bremner, David 193 Bygi, Mojtaba Nouri 473

С

Calinescu, Gruia 141 Cardinal, Jean 443 Carmi, Paz 39 Chambers, Erin 59 Chan, Timothy M. 135, 431 Chen, Dan 425 Christ, Tobias 467 Custard, Grant 267

D

Damian, Mirela 361 Das, Ananda Swarup 123, 129, 343 Das, Gautam 375 Davoodi, Mansoor 501 De Carufel, Jean-Lou 93, 147 De Rezende, Pedro J. 309 De, Minati 331, 347, 375 Demaine, Erik D. 77, 83, 89, 153, 229, 235, 461
Demaine, Martin L. 77, 89, 229, 461
Deza, Antoine 193, 267
Dillabaugh, Craig 147
Dimitrov, Nikolay 117
Doerr, Benjamin 315
Douieb, Karim 105
Dujmovic, Vida 229
Durocher, Stephane 303, 355
Díaz-Báñez, José Miguel 15

Ε

Eastman, Matthew 105 Eisenstat, Sarah 235 Elbassioni, Khaled 437 Elkin, Elazar 399 Elkin, Michael 33 Eskandari, Marzieh 495, 501 Evans, William 479

F

Fabila-Monroy, Ruy 21 Fazli, Mohammadamin 367 Fernandez Anta, Antonio 163 Flatland, Robin 361

G

Gemsa, Andreas 205 Ghali, Sherif 405 Ghodsi, Mohammad 367, 473 Gioan, Emeric 187 Gonzalez, Teofilo 319 Gonzalez-Aguilar, Hernan 21 Grant, Elyot 431 Gu, Chen 217 Guerra Filho, Gutemberg 309 Guibas, Leonidas 217, 279 Gupta, Prosenjit 123, 129 Heeringa, Brent 181 Held, Martin 261, 489 Heredia, Marco A. 15, 21 Hershberger, John 211 Hoffmann, Michael 467 Horiyama, Takashi 65 Hsu, Chia-Hong 381 Hua, William 193 Huber, Stefan 261 Huemer, Clemens 21 Hurtado, Ferran 229

Ito, Hiro 443

J

L

Jansens, Dana 241 Jiang, Xiaoye 217, 279 Jin, Kai 111 Ju, Tao 59

κ

Kalavagattu, Anil Kishore 343 Kao, Mong-Jen 43 Karloff, Howard 141 Katz, Bastian 43 Katz, Matthew 39 Khalilabadi, Pooya Jalaly 367 Kirkpatrick, David 27 Kiyomi, Masashi 393 Korman, Matias 361, 443 Kostitsyna, Irina 27 Kostochka, Alexandr 285 Kothapalli, Kishore 129, 343 Kouhestani, Bahram 455 Krug, Marcus 43 Kurdia, Anastasia 461

L

Lange, Carsten 273 Langerman, Stefan 443 Lee, Der-Tsai 43 Letscher, David 59 Levi, Harel 399 Liao, Chung-Shou 381 Liu, Lu 59 Lubiw, Anna 229 Lund, Benjamin 223

Μ

Mahdavi, Salma Sadat 455 Maheshwari, Anil 55, 105, 147, 331 Malestein, Justin 181 Mann, Willi 489 Matsui, Hiroaki 77 Matulef, Kevin 111 Mehrabi, Saeed 355 Mehrabian, Abbas 373 Milenkovic, Victor 99 Millman, David L. 485 Mitchell, Joseph S. B. 163 Mohades, Ali 455, 495, 501 Mohamad, Mustafa 49 Mondal, Debajyoti 303, 355 Morin, Pat 425 Mosteiro, Miguel A. 163

Ν

Nandy, Subhas 331, 347, 375 Nickerson, Bradford 337 Nishat, Rahnuma Islam 303 Nöllenburg, Martin 43, 205

0

O'Rourke, Joseph 71, 153, 461 Okayama, Yosuke 393 Omri, Eran 33 Ozkan, Ozgur 361

Ρ

Panahi, Fatemeh 501 Peláez, Canek 15 Pilaud, Vincent 199 Polishchuk, Valentin 27 Purdy, George 223

R

Rand, Alexander 157 Rappaport, David 49, 361 Rauf, Imran 437 Ray, Saurabh 437 Rivin, Igor 169 Rote, Günter 77, 229 Rowekamp, Brandon 417 Rutter, Ignaz 43, 205

S

Sack, Jörg-Rüdiger 55

Sacks, Elisha 99 Safari, Mohammadali 367 Sarioz, Deniz 297 Saumell, Maria 241 Saveliev, Peter 411 Scheffer, Christian 325 Schewe, Lars 193 Schulz, André 229 Sellarès, J. Antoni 15 Sember, Jeff 479 Shahbaz, Kaveh 55 Shoji, Wataru 65 Siddiqi, Kaleem 249 Skala, Matthew 355 Smid, Michiel 105, 331 Smith, Justin 223 Sol, Kevin 187 Souvaine, Diane L. 229 Speckmann, Bettina 387 Srinathan, Kannan 123, 129, 343 Steiger, William 13 Stephen, Tamon 267 Stolpner, Svetlana 249 Streinu, Ileana 169 Subsol, Gérard 187 Sun, Jian 279 Sun, Timothy 287Suri, Subhash 211

W

Wagner, Dorothea 43 Weissman, Ayal 399 Welzl, Emo 423 Whitesides, Sue 249, 303 Wilkinson, Bryan T. 135 Winslow, Andrew 229, 449 Wu, Yujun 99 Wuhrer, Stefanie 361

Х

Xia, Ge 175 Xie, Feng 267

Υ

Yildiz, Hakan 211

Ζ

Zarrabi-Zadeh, Hamid 55 Zhang, Liang 175 Zheng, Rong 255

Т

Theran, Louis 181 Toth, Csaba 223, 449 Toussaint, Godfried 49, 449, 461

U

Uehara, Ryuhei 77, 393 Urrutia, Jorge 15, 21, 461

V

Vahrenhold, Jan 325 Valtr, Pavel 21 Van Renssen, André 241, 387 Ventura, Inmaculada 15 Verdonschot, Sander 241 Verma, Vishal 485 Viglietta, Giovanni 461 Vilcu, Costin 71 Vogtenhuber, Birgit 21 Vu, Khuong 255