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 $A = \{a_1, a_2, \ldots, a_m\}$ denote points on a line, such that $a_i < a_{i+1}$ for all $i, 1 \leq i \leq m-1$. Similarly we use $B = \{b_1, b_2, \ldots, b_n\}$ 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:

$$d_1(a, b) = |a - b| \tag{1}$$

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 $P_0, P_1, P_2, ...$ 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).

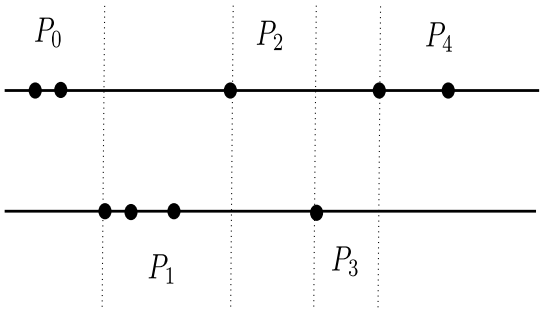

Figure 1: Partitioning of the set $A \cup B$

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 algorithm 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.

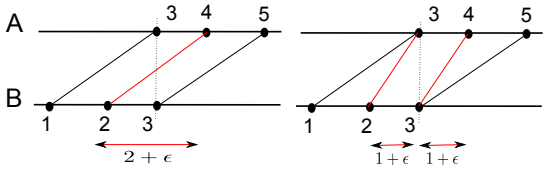

Figure 2: The identity matching on the left is optimal for d_1 with $d_1(A, B) = 6$. However it is not optimal for d_2 where $d_2(A, B) = 12$. The matching on the right is optimal for d_2 with $d_2(A, B) = 10$. As can be seen the edge $(2, 4)$ is only optimal for d_2 if $\epsilon > \sqrt{2}$ or $\epsilon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

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} ← d(a_1, b_1)
for i = 2 to m do
   W_{i,1} ← W_{i-1,1} + d(a_i, b_1)
end for
for j = 2 to n do
   W_{1,j} ← 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* ← min(W_{i-1,j}, W_{i,j-1}, W_{i-1,j-1})
      W_{i,j} ← W* + d(a_i, b_j)
   end for
end for
{W_{mn} stores the weight of the optimal many-to-many matching}
return W_{mn}
```

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

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:*

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

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 right edges is greater than or equal to 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. □

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 B that 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:

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

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

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

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 $\{a_i, b_j\}$ 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 current 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. □

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

Table 1: Experimental Results of running Algorithm 2

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

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

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

$m = n$	$m \times n$	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

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

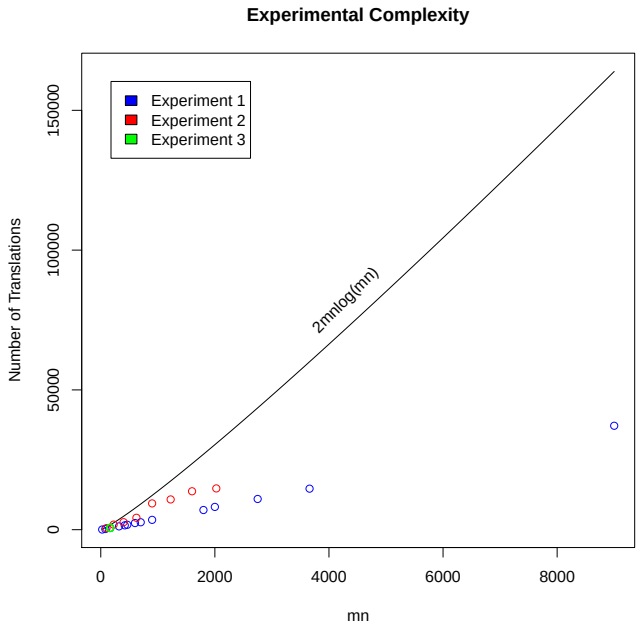

Figure 3: Summary of experimental results shows that all our data fall under the curve $2mn\log(mn)$

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

[1] 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-to-many 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.