

Matching Points with Things

Greg Aloupis¹, Jean Cardinal¹, Sébastien Collette^{1,*}, Erik D. Demaine²,
Martin L. Demaine², Muriel Dulieu³, Ruy Fabila-Monroy⁴, Vi Hart⁵,
Ferran Hurtado⁶, Stefan Langerman^{1,**}, Maria Saumell⁶, Carlos Seara⁶,
and Perouz Taslakian¹

¹ Université Libre de Bruxelles, CP212, Bld. du Triomphe, 1050 Brussels, Belgium
{galoupis,jcardin,secollet,slanger,ptaslaki}@ulb.ac.be
Supported by the Communauté française de Belgique - ARC.

² MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St.,
Cambridge, MA 02139, USA
{edemaine,mdemaine}@mit.edu

³ Polytechnic Institute of NYU, USA
mdulieu@gmail.com

⁴ Instituto de Matemáticas, Universidad Nacional Autónoma de México
ruy@ciencias.unam.mx

⁵ Stony Brook University, Stony Brook, NY 11794, USA
vi@vihart.com

⁶ Universitat Politècnica de Catalunya, Jordi Girona 1–3, E-08034 Barcelona, Spain
{ferran.hurtado,maria.saumell,carlos.seara}@upc.edu

Partially supported by projects MTM2009-07242 and Gen. Cat. DGR 2009SGR1040.

Abstract. Given an ordered set of points and an ordered set of geometric objects in the plane, we are interested in finding a non-crossing matching between point-object pairs. We show that when the objects we match the points to are finite point sets, the problem is NP-complete in general, and polynomial when the objects are on a line or when their number is at most 2. When the objects are line segments, we show that the problem is NP-complete in general, and polynomial when the segments form a convex polygon or are all on a line. Finally, for objects that are straight lines, we show that the problem of finding a min-max non-crossing matching is NP-complete.

1 Introduction

Finding a matching between pairs of planar objects, that is connecting them by a set of non-crossing line segments, is a natural problem that has been frequently studied in computational geometry. It is well known, for instance, that given two sets of n points in the plane, say n red points and n blue points, there always exists a non-crossing matching between red and blue points. In particular, it is not difficult to show that the minimum Euclidean length matching is non-crossing. Kaneko and Kano [22] survey a number of related results. Algorithms for finding minimum sum and minimum bottleneck distance red-blue matchings are given in [15,27].

* Chargé de Recherches du FRS-FNRS.

** Maître de Recherches du FRS-FNRS.

In this paper, we investigate related questions for general planar objects instead of points. Again, matchings are represented by line segments, but here the endpoints can be placed anywhere inside the corresponding matched objects. Note that as a consequence of the aforementioned result on points, there always exists a non-crossing matching between two sets of objects. Here we consider the problem where we are given object *pairs* (i.e. a point and the geometric object it must be matched to) and need to find a set of non-crossing matching edges, if one exists. This can be seen as a 1-regular graph drawing problem with constraints on the location of vertices.

Related work. Problems on matchings have an important role in combinatorial graph theory, both for theoretical and applied aspects; hence a lot of research is devoted to the study of these problems (for example, see [24]).

Suppose we are given an embedding of a graph in the Euclidean plane, where the vertices are points in the plane, edges are rectilinear line segments, and weights on these edges represent the Euclidean distance between the vertices they connect. Elementary geometry tells us that the sum of any pair of opposite sides of a convex quadrilateral is strictly smaller than the sum of the diagonals. Remarkably, this implies that the minimum weight matching in any realization of the complete graphs K_{2n} and $K_{n,n}$ consists of pairwise non-crossing segments. These geometric graph problems can be solved using generic algorithms for weighted graphs. However, in the planar case just mentioned, Vaidya [27] proved that it is possible to obtain specialized algorithms with better running times (the title of his paper is especially suggestive: *Geometry helps in matching*). In particular, in [27] the running time of the generic algorithm for the bipartite case was reduced from $O(n^3)$ to $O(n^{2.5} \log n)$. This was later improved to $O(n^{2+\epsilon})$ by Agarwal et al. [1]. Similar results have been obtained for other matching variations, such as *bottleneck matching* or *uniform matching*, in the work of Efrat, Itai and Katz [15]. The authors consider matchings as an approach for the problem of matching a point set A with a point set B , where A must be moved in some way to coincide as much as possible with B or one of its subsets. This is a fundamental problem in pattern recognition [5,7,8,10,11,12,19,20,21].

The non-crossing requirement in our problems is quite natural in geometric scenarios (see for example [25,2,3]), and the family of geometric problems that we consider has several applications; these applications include geometric shape matching [4,13,17,18], colour-based image retrieval [13], music score matching [26], and computational biology [14,16].

Our results. Throughout the paper, we let $P := \{p_1, p_2, \dots, p_n\}$ be a set of points in the plane and $T := \{t_1, t_2, \dots, t_n\}$ be a set of planar objects. A *matching*

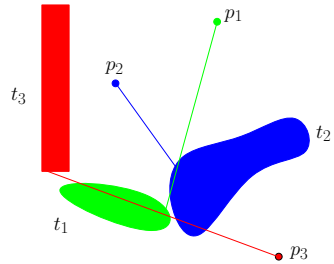


Fig. 1. A non-crossing matching for a set $P = \{p_1, p_2, p_3\}$ of points and a set $T = \{t_1, t_2, t_3\}$ of planar objects

for a pair (P, T) consists of a set of line segments, called *edges*, of the form $\{p_1m_1, p_2m_2, \dots, p_nm_n\}$, where $m_i \in t_i$. A matching is said to be *non-crossing* if no pair of matching edges properly cross. This is illustrated in Figure 1.

We consider the problem of deciding whether a non-crossing matching exists for a given pair (P, T) . In cases where a non-crossing matching always exists, we consider the problem of finding the matching that minimizes either the length of the longest edge, or the sum of the lengths of all the edges.

In Section 2, we study the case where the objects t_i are finite point sets. We prove that the decision problem is NP-complete in general, but becomes polynomial when every t_i has size at most two, or when all the t_i s are on a line. In Section 3 we consider T to be a set of line segments and prove that the (P, T) matching problem is NP-complete. We also consider special cases, such as the case when the line segments form a convex polygon surrounding all points in P (Section 4), or the case when segments belong to a single line (Section 5). We show that these special cases have polynomial solutions. Finally, in Section 6, we consider the problem of matching points with lines. In this variation, a non-crossing matching always exists, but the optimization problems are NP-hard.

2 Matching Points with Finite Point Sets

We first prove that if the objects t_i are pairs of points, then we can decide whether there exists a non-crossing matching in polynomial time. On the other hand, if the sets t_i may contain three points or more, the problem becomes NP-complete. This situation is similar to that of the k -satisfiability problem (k -SAT). In k -SAT we are given a boolean formula f of the form $C_1 \wedge C_2 \wedge \dots \wedge C_m$ (where each C_i is an OR clause of k variables), and we are required to find a truth assignment of its variables that satisfy the formula. It is well-known that 2-SAT has a polynomial-time solution whereas k -SAT is NP-complete for $k \geq 3$. The 2-SAT problem can be solved in polynomial time by exploiting the fact that, if in a clause a variable is set to false, it forces the other variable to be set to true. This dependency between the variables can be represented by an *implication graph*.

An implication graph for the formula f is a directed graph having two vertices for each variable x_i of f , one of these vertices is labeled x_i while the other is labeled $\neg x_i$. The vertex x_i represents setting x_i to true while $\neg x_i$ represents setting x_i to false. Dependencies between literals in f are represented by directed edges. Thus if $(x_i \vee \neg x_j)$ is a clause in f , in the implication graph there would be a directed edge from $\neg x_i$ to $\neg x_j$ and a directed edge from x_j to x_i . These edges represent the fact that if x_i is set to false then x_j must also be set to false in order for the formula to be satisfied. Likewise if x_j is set to true then x_i must be set to true.

There exists a truth assignment satisfying f if and only if no strong component of the implication graph contains both a vertex and its negation. The implication graph can be constructed in $O(m)$ time, where m is the number of clauses. The previous condition can be verified in $O(m)$ time, and in general the strong

components of a directed graph of v vertices and e edges can be computed in $O(e+v)$ time. A similar implication graph can be constructed for our problem when t_i is a pair of points. Using this graph, it is possible to decide in $O(n^2)$ time whether (P, T) has a non-crossing perfect matching.

Theorem 1. *Given an ordered set P of points and an ordered set T of pairs of points, there is an algorithm that decides in $O(n^2)$ time whether (P, T) has a non-crossing matching.*

Proof. Assume that the elements of each t_i are labeled arbitrarily “ \mathcal{T}_i ” and “ \mathcal{F}_i ” (thus $t_i = \{\mathcal{T}_i, \mathcal{F}_i\}$). We think of each p_i as a boolean variable, so that if we match p_i with \mathcal{T}_i then p_i is set to “true”, and if p_i is matched with \mathcal{F}_i , it is set to “false”. We construct a directed implication graph G as follows: For each p_i we have vertices p_{i,\mathcal{T}_i} and p_{i,\mathcal{F}_i} in G . For every $i, j = 1, 2, \dots, n$, we add the edge $(p_{i,X}, p_{j,Y})$ (X equal to \mathcal{T}_i or \mathcal{F}_i , and Y equal to \mathcal{T}_j or \mathcal{F}_j) to G if and only if the line segments $\overline{p_i, X}$ and $\overline{p_j, \neg Y}$ intersect. For example if $\overline{p_i, \mathcal{T}_i}$ intersects $\overline{p_j, \mathcal{F}_j}$, we add the edge $(p_{i,\mathcal{T}_i}, p_{j,\mathcal{T}_j})$ to G (since if p_i is matched to \mathcal{T}_i , p_j must be matched to \mathcal{T}_j as well). So (P, T) has a non-crossing complete matching if and only if for every p_i , p_{i,\mathcal{T}_i} and p_{j,\mathcal{F}_j} lie in different strong connected components of G . Since G is constructed in $O(n^2)$ time and has $O(n^2)$ edges, the overall complexity of the algorithm is $O(n^2)$. □

2.1 Matching Points with Triples

By a reduction from Planar 3-SAT, we can prove that the problem of matching points with triples is NP-complete, even when the points within each triple are horizontally collinear. Details are omitted due to space limitations.

Theorem 2. *Given an ordered set P of points and an ordered set T of triples of points, it is NP-complete to decide whether (P, T) has a non-crossing matching. The problem remains NP-complete even if each triple is horizontally collinear.*

2.2 Matching Points with k -Tuples on a Line

Theorem 3. *Given an ordered set P of points and an ordered set T of k -tuples of points on a line, we can decide in $O(k^3n^2)$ time whether (P, T) has a non-crossing matching.*

Proof. Without loss of generality, assume all the tuples are on a horizontal line L . Assume also that all points are on one side of L ; otherwise we may consider each problem separately as the matching edges on each side of L do not interact. We now show how to build a dynamic programming table that solves the problem.

In any solution to the problem, if a matching edge e is part of the solution, then there is no matching edge that intersects e . Therefore, we can consider the regions on each side of e (sub-problems) separately and determine whether they in turn have a valid solution. Thus, a sub-problem (P', T') is defined as follows (see Figure 2): given a simple quadrilateral A with one face adjacent to L , we want to decide if it is possible to find a non-crossing matching completely contained in the region A for all the points contained in A , i.e., we want to solve

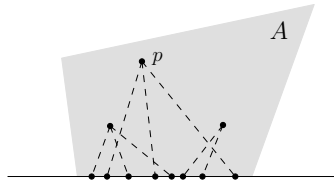


Fig. 2. Definition of a sub-problem

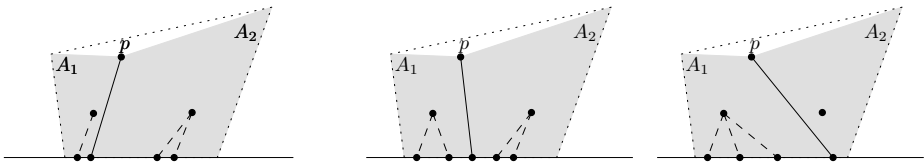


Fig. 3. In this example, there are three pairs of sub-problems to consider to decide if p can be matched

the problem with $P' = P \cap A$ and T' containing the subsets of the tuples of T contained in A . If A does not contain at least one point of P (sub-problem of size 0), it is trivially true that there is a non-crossing matching. Otherwise, to solve the sub-problem we consider the topmost point p in A . It has at most k possible matching edges. If it has no possible matching edge, i.e., if all points that p could be matched to in T are out of A , then there is no valid matching.

Each of the possible matching edges defines two new independent sub-problems (see Figure 3) in the quadrilaterals A_1 and A_2 , whose sizes are strictly smaller than that of the original problem, as there is one less point to match. To decide whether a matching exists for the original sets P and T , we solve the sub-problem defined by the bounding box of both P and T . Notice that all the sub-problems correspond to quadrilaterals defined by a pair of possible matching edges (or by the edges of the bounding box). Moreover, since in every sub-problem A the y -coordinates of the corners of the bounding box are at least as large as that of every point in A , then the union of the regions of the sub-problems of A will contain all the points in A . Thus no point will be left out.

The dynamic programming table has $kn + 2$ rows and $kn + 2$ columns, each of which corresponds to a possible matching edge or one of the left and right edges of the bounding box; the cells correspond to sub-problems (a pair of non-adjacent edges defines a quadrilateral), and we fill them with true or false depending on whether or not a matching exists for the considered sub-problem. Filling a cell of the table corresponds to solving at most k pairs of sub-problems, which implies at most $2k$ lookups in the table for each of the $O(k^2n^2)$ cells. Therefore, the total time and space required to solve the problem is $O(k^3n^2)$. \square

The additional restriction of having points on a line greatly simplifies the problem, because the problem is NP-hard in the general case, but is polynomial for points on a line.

3 Matching Points with Line Segments: General Case

In this section we show that deciding the existence of a non-crossing matching between a set of points and a set of line segments is NP-complete, even if the segments are all horizontal or all have equal length. The proof uses appropriate gadgets to show that this problem reduces from the problem of matching points to triples (Theorem 2). It is omitted due to space limitations.

Theorem 4. *Given an ordered set P of points and an ordered set T of line segments, it is NP-complete to decide whether (P, T) has a non-crossing matching. The problem remains NP-complete even if all line segments in T are horizontal or all have equal length.*

4 Matching Points to an Enclosing Convex Polygon

In this special case of matching points with line segments, we assume the segments are the edges of a convex polygon and the points to be matched are inside the polygon, in general position.

We first describe some geometric properties of the input of this problem. We then describe an algorithm that runs in $O(n \log^2 n)$ time, and finds a non-crossing matching (if one exists) between a given set of point-segment pairs where the line segments form a convex polygon enclosing the points. This algorithm allows a minimum-length matching to be extracted easily.

Let $D^o = \{\Delta_1^o, \Delta_2^o, \dots, \Delta_n^o\}$ be a set of triangles where each Δ_i^o is the triangle with apex p_i and base t_i . Any valid matching edge e_i must lie completely inside Δ_i^o . Depending on the positions of other triangles in D^o , some candidate positions for e_i can be identified as invalid because they would always cross other matching edges. By identifying such cases, triangle Δ_i^o can be reduced to a smaller triangle Δ_i . At any time, the *reduced triangle* Δ_i has apex p_i but its opposite base is a subsegment of t_i . Initially, $\Delta_i = \Delta_i^o$.

There are four ways in which two triangles Δ_i and Δ_j interact. The second case leads to a *reduction rule*. We describe the four cases below (see Figure 4):

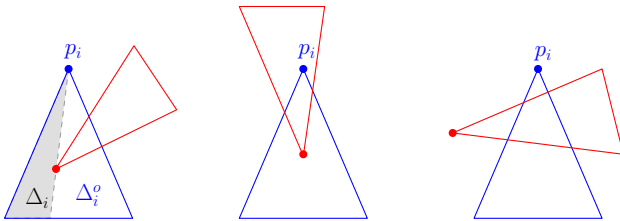


Fig. 4. Left: Δ_i^o is reduced to Δ_i (case 2). Middle: one of the two combinatorially distinct solutions (case 3). Right: no solution exists (case 4).

1. Δ_i, Δ_j are disjoint. In this case there will never be a direct interaction between the two.
2. p_j is in Δ_i , but p_i is not in Δ_j . In this case Δ_i should be reduced so that the two triangles become tangent (so that p_j is no longer in Δ_i).
3. p_i is in Δ_i and p_j is in Δ_j . We call Δ_i and Δ_j *inverted triangles*, and cannot immediately make a reduction.
4. Both edges incident to each of p_i and p_j pairwise intersect. Then no non-crossing matching exists.

Note that in case (2) there is no choice but to reduce. The matching edge e_j that is finally chosen will block any candidate e_i that is outside the newly reduced Δ_i . In case (3) there are two combinatorially valid placements for e_i, e_j , with respect to the positions of p_i, p_j . There is no reason to choose arbitrarily before verifying that neither triangle will be reduced further.

If we exhaustively apply our reduction rule to the triangles based on the cases described above, we would end up with a set having certain properties. Due to lack of space, we omit a detailed discussion of these properties.

Let two (three) mutually inverted triangles be called an *inverted pair (triple)*. Let a *unit* be a (possibly reduced) triangle, an inverted pair, or an inverted triple.

Theorem 5. *Given an ordered set P of points inside a convex polygon having an ordered set T of line segments as edges, deciding whether (P, T) admits a non-crossing matching can be done in $O(n \log^2 n)$ time.*

Proof. We provide an algorithm where we employ a divide-and-conquer technique. Suppose that we have solved the problem separately on two consecutive convex chains (we can transform a chain into a polygon by adding 3 fake edges and points; thus, solving the problem on a chain is equivalent to solving the polygonal version).

We claim that we can merge the two solutions in $O(n \log n)$ time. Each solution is a set of disjoint triangles and inverted pairs or triples. Refer to Figure 5.

Let A and B be two solved sub-problems of size k . We construct a standard point-location data structure on each in $O(k)$ time [23] by first triangulating A and B using Chazelle’s linear-time triangulation algorithm [9]. Now, for every

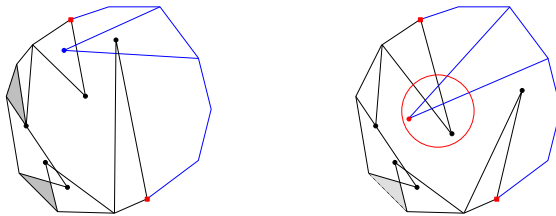


Fig. 5. Merging two solved sub-problems. In the left diagram, the grey regions in the left (black) sub-problem cannot contain points from the right (blue) sub-problem if there is a valid solution. In the right diagram, we see the type of event that we must check for after some initial reductions.

point p_i in B , we locate p_i in A to determine if it is inside a unit in A . Note that p_i can be in at most one unit. If it is, we determine if Δ_i reduces this unit by case (2). Likewise, for every point p_j in A , we locate p_j in B to determine if it is inside a unit in B and apply the appropriate reductions. Note that if at some moment Δ_i (belonging to B) gets reduced, this will not affect its corresponding unit in A ; the same holds for all Δ_j in A that get reduced.

Of course, it is possible that Δ_i will be inverted with a triangle in A . In this case we simply determine if there are reductions and, if applicable, we merge the two units. Therefore a constant number of reductions are applied per point, which means we spend $O(\log k)$ time per point for the point-location step.

The only unresolved issue is to detect if case (4) will occur between triangles of A and B (see Figure 5-right). Given that all triangles have been reduced and merged into units, essentially we are verifying that no segments intersect. For this we can use the Bentley-Ottmann line segment intersection algorithm and stop as soon as a bad intersection is found [6]. For k segments, such queries take $O(\log k + h \log k)$ time, where h is the number of intersections reported. As we stop as soon as we report one intersection, $h = 1$ and hence the total time is $O(\log k)$ per point. Thus our merge procedure takes $O(k \log k)$ time. By a simple recurrence analysis, we determine that the entire algorithm takes $O(n \log^2 n)$ time. \square

The algorithm described in the proof of Theorem 5 either decides that no solution exists, or otherwise produces a final set of reduced triangles that represents all valid solutions to the problem. In the latter case, every resulting unit is disjoint and thus independent of all others. So in each triangle we can easily pick the shortest joining segment, and in each inverted pair/triple, we try out the two possible choices and take the best matching. Therefore, after the algorithm finds a solution, the min-max and min-sum optimization problems can be solved in linear time.

5 Matching Points with Segments on a Line

As another special case of matching points to line segments, we now consider the case when the input line segments belong to one single line L . Throughout this section we will assume, without loss of generality, that L is horizontal. As no matching edge will cross over L , our problem is split into two disjoint sub-problems, and we focus on points above L . We consider two cases, depending on whether the segments are disjoint or not.

5.1 Matching Points with Disjoint Segments on a Line

Theorem 6. *Given an ordered set P of points above a horizontal line L and an ordered set T of disjoint line segments belonging to L , deciding whether (P, T) admits a non-crossing matching can be done in linear time. In the affirmative, the matching that minimizes the sum of the lengths of the edges can be found within the same time bound.*

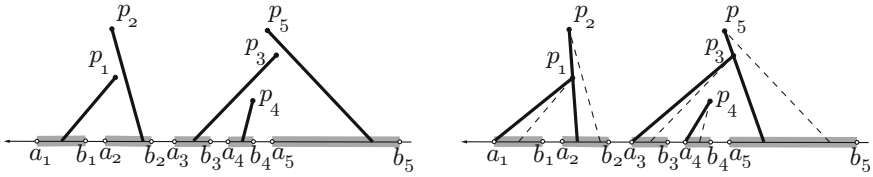


Fig. 6. Leftmost non-crossing matching (right) obtained from an initial non-crossing matching (left)

Proof. We denote by $[a_i, b_i]$ the interval corresponding to segment t_i , for $i = 1, \dots, n$. Since the intervals are given in sorted order, we have $a_1 \leq b_1 < a_2 \leq b_2 < \dots < a_n \leq b_n$.

If (P, T) admits some non-crossing matching $\{p_1m_1, p_2m_2, \dots, p_nm_n\}$, where $a_i \leq m_i \leq b_i$ for all $i = 1, 2, \dots, n$, we can always *slide* the point m_i inside t_i to a position m_i^L as far to the left as possible (see Figure 6). This gives the unique *leftmost non-crossing matching* for (P, T) , $\{p_1m_1^L, p_2m_2^L, \dots, p_nm_n^L\}$. Notice that either $m_i^L = a_i$, or p_i and m_i^L are collinear with some p_j with $j < i$.

Next we describe an algorithm for finding the leftmost non-crossing matching, if it exists. The algorithm considers points in a sequential greedy fashion, in the left-to-right order of the corresponding segments.

For p_1 , the leftmost matching is simply given by the segment p_1a_1 . We then consider the rays from the endpoints of this segment in the direction of the negative semiaxis of abscissae; their points at infinity can be symbolically described as $q_0 = (-\infty, 0)$ and $q_1 = (-\infty, y(p_1))$.

The *forbidden region* is the (unbounded) region enclosed by an alternating sequence of horizontal line segments and subsegments of matched edges (See Figure 7). This region is updated at every step of the algorithm. Initially, it is described clockwise by its vertices, namely $q_1p_1a_1q_0$. Observe that if p_2 is inside the forbidden region, then a non-crossing matching (P, T) would be impossible. If p_2 is outside the forbidden region, a matching is possible if and only if there is some point m_2 in the interval a_2b_2 such that the segment p_2m_2 does not cross the forbidden region. In the affirmative, we slide m_2 to its leftmost possible position, and shoot a ray from p_2 in the direction of the negative semiaxis of abscissae, which may go to infinity, or stop by hitting the segment p_1a_1 . The forbidden

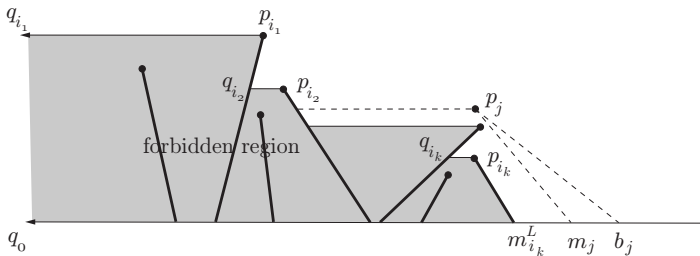


Fig. 7. Forbidden region and incremental step

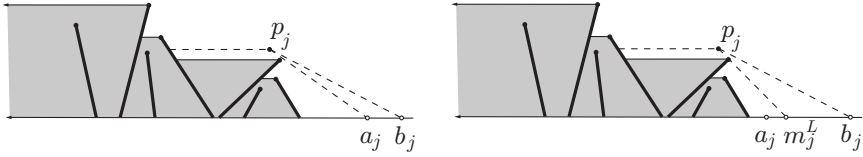


Fig. 8. Moving the new edge to the leftmost position

region is updated in each case, and is always defined by alternating horizontal edges with portions of segments from the matching.

Assume that, in a generic step, we have obtained the leftmost matching $\{p_1m_1^L, p_2m_2^L, \dots, p_{j-1}m_{j-1}^L\}$ and we are processing p_j . Let $q_{i_1}p_{i_1}q_{i_2}p_{i_2}\dots q_{i_k}p_{i_k}m_{i_k}^Lq_0$ be the current forbidden region (refer to Figure 7). Observe that if there is some $m_j \in [a_j, b_j]$ such that the segment p_jm_j can be added to the edges found so far, getting a non-crossing matching, the segment p_jb_j is also valid. We show next how to check the validity of p_jb_j .

We first check the y coordinates of the points $m_{i_k}, p_{i_k}, p_{i_{k-1}}, \dots$, which form an increasing sequence, until we find that $y(p_{i_t}) \geq y(p_j) \geq y(p_{i_{t+1}})$ (the case in which $y(p_j)$ is a maximum is completely analogous). Then, we check whether the segment p_jb_j crosses the segments $m_{i_k}p_{i_k}, q_{i_{k-1}}p_{i_{k-1}}, \dots, q_{i_{t-1}}p_{i_{t-1}}$. In the affirmative, the algorithm is over, as no crossing-free matching is possible. Otherwise, the segment p_jb_j is valid. We slide the point matched with p_j as much to the left as possible (Figure 8), which can be done by finding the angularly closest point among $p_{i_{t+1}}, p_{i_{t+2}}, \dots, p_{i_k}, a_j$.

If we shoot a ray from p_j in the direction of the negative semiaxis of abscissae, we hit the boundary of the forbidden region in a point q_j , possibly at infinity, and the forbidden region is updated to be $q_{i_1}p_{i_1}q_{i_2}p_{i_2}\dots p_{i_t}q_jp_jm_j^Lq_0$.

The cost of the step for p_j is proportional to the size of the forbidden polygonal region that disappears, and that will never be processed again. Therefore, the amortized cost of one step is constant and the global cost of the algorithm is $O(n)$. At the end we obtain the leftmost matching $\{p_1m_1^L, p_2m_2^L, \dots, p_nm_n^L\}$, unless no matching is possible.

If (P, T) admits a non-crossing matching, with a symmetrical algorithm we can obtain the rightmost matching $\{p_1m_1^R, p_2m_2^R, \dots, p_nm_n^R\}$. Then any points m_i in the intervals $[m_i^L, m_i^R]$ provide a non-crossing matching $\{p_1m_1, p_2m_2, \dots, p_nm_n\}$. In particular, in each interval $[m_i^L, m_i^R]$ we can pick the matching point m_i which is closest to p_i , and hence obtain the matching that minimizes the sum of the lengths of the edges in additional $O(n)$ time. \square

5.2 Matching Points with Arbitrary Segments on a Line

In this section, we show that when the given segments are confined to a line and possibly intersect, we can determine the existence of a non-crossing matching in polynomial time. The proof first discretizes the problem, and then uses the same approach as in the proof of Theorem 3 for k -tuples with $k = O(n^2)$. Details of the proof are omitted from this version of the paper.

Theorem 7. *Given an ordered set P of points above a horizontal line L and an ordered set T of line segments belonging to L , deciding whether (P, T) admits a non-crossing matching can be done in $O(n^8)$ time.*

6 Matching Points with Lines

In the case where points are matched with lines, it is easy to see that a non-crossing matching always exists: choose an arbitrary direction, not parallel to any line, and project each point on its corresponding line in that direction.

Here we consider the optimization problem of minimizing the maximum length over all matching edges. The proof, omitted here due to space limitations, uses a reduction from the problem of deciding the existence of a non-crossing matching between a set of points and a set of segments.

Theorem 8. *Given an ordered set P of points and an ordered set T of lines, finding a min-max non-crossing matching of (P, T) is NP-complete.*

References

1. Agarwal, P., Aronov, B., Sharir, M., Suri, S.: Selecting distances in the plane. *Algorithmica* 9(5), 495–514 (1993)
2. Aichholzer, O., Bereg, S., Dumitrescu, A., García, A., Huemer, C., Hurtado, F., Kano, M., Márquez, A., Rappaport, D., Smorodinsky, S., Souvaine, D., Urrutia, J., Wood, D.R.: Compatible geometric matchings. *Computational Geometry: Theory and Applications* 42, 617–626 (2009)
3. Aichholzer, O., Cabello, S., Fabila-Monroy, R., Flores-Penalzoza, D., Hackl, T., Huemer, C., Hurtado, F., Wood, D.R.: Edge-Removal and Non-Crossing Configurations in Geometric Graphs. In: *Proceedings of 24th European Conference on Computational Geometry*, pp. 119–122 (2008)
4. Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: *Handbook of computational geometry*, pp. 121–154 (1999)
5. Arkin, E., Kedem, K., Mitchell, J., Sprinzak, J., Werman, M.: Matching points into noise regions: combinatorial bounds and algorithms. In: *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pp. 42–51 (1991)
6. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers* 28(9), 643–647 (1979)
7. Cabello, S., Giannopoulos, P., Knauer, C., Rote, G.: Matching point sets with respect to the Earth Mover’s Distance. *Computational Geometry: Theory and Applications* 39(2), 118–133 (2008)
8. Cardoze, D., Schulman, L.: Pattern matching for spatial point sets. In: *Proceedings. 39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1998, pp. 156–165 (1998)
9. Chazelle, B.: Triangulating a simple polygon in linear time. *Discrete and Computational Geometry* 6(1), 485–542 (1991)
10. Chew, L., Dor, D., Efrat, A., Kedem, K.: Geometric pattern matching in d -dimensional space. *Discrete and Computational Geometry* 21(2), 257–274 (1999)

11. Chew, L., Goodrich, M., Huttenlocher, D., Kedem, K., Kleinberg, J., Kravets, D.: Geometric pattern matching under Euclidean motion. *Computational Geometry: Theory and Applications* 7(1-2), 113–124 (1997)
12. Chew, L., Kedem, K.: Improvements on geometric pattern matching problems. In: Nurmi, O., Ukkonen, E. (eds.) *SWAT 1992*. LNCS, vol. 621, pp. 318–325. Springer, Heidelberg (1992)
13. Cohen, S.: Finding color and shape patterns in images. PhD thesis, Stanford University, Department of Computer Science (1999)
14. Colannino, J., Damian, M., Hurtado, F., Iacono, J., Meijer, H., Ramaswami, S., Toussaint, G.: An $O(n \log n)$ -time algorithm for the restriction scaffold assignment problem. *Journal of Computational Biology* 13(4), 979–989 (2006)
15. Efrat, A., Itai, A., Katz, M.: Geometry helps in bottleneck matching and related problems. *Algorithmica* 31(1), 1–28 (2001)
16. Formella, A.: Approximate point set match for partial protein structure alignment. In: *Proceedings of Bioinformatics: Knowledge Discovery in Biology (BKDB 2005)*, Faculdade Ciências Lisboa da Universidade de Lisboa, pp. 53–57 (2005)
17. Giannopoulos, P., Veltkamp, R.: A pseudo-metric for weighted point sets. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) *ECCV 2002*. LNCS, vol. 2352, pp. 715–730. Springer, Heidelberg (2002)
18. Grauman, K., Darrell, T.: Fast contour matching using approximate earth mover’s distance. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 220–227 (2004)
19. Heffernan, P.: Generalized approximate algorithms for point set congruence. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) *WADS 1993*. LNCS, vol. 709, pp. 373–373. Springer, Heidelberg (1993)
20. Heffernan, P., Schirra, S.: Approximate decision algorithms for point set congruence. In: *Proceedings of the eighth annual Symposium on Computational geometry*, pp. 93–101 (1992)
21. Huttenlocher, D., Kedem, K.: Efficiently computing the Hausdorff distance for point sets under translation. In: *Proceedings of the Sixth ACM Symposium on Computational Geometry*, pp. 340–349 (1990)
22. Kaneko, A., Kano, M.: Discrete geometry on red and blue points in the plane—a survey. *Discrete & Computational Geometry* 25, 551–570 (2003)
23. Kirkpatrick, D.: Optimal search in planar subdivisions. *SIAM Journal on Computing* 12(1), 28–35 (1983)
24. Lovász, L., Plummer, M.D.: *Matching theory*. Elsevier Science Ltd., Amsterdam (1986)
25. Rappaport, D.: Tight bounds for visibility matching of f -equal width objects. In: Akiyama, J., Kano, M. (eds.) *JCDCG 2002*. LNCS, vol. 2866, pp. 246–250. Springer, Heidelberg (2002)
26. Typke, R., Giannopoulos, P., Veltkamp, R., Wiering, F., Van Oostrum, R.: Using transportation distances for measuring melodic similarity. In: *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pp. 107–114 (2003)
27. Vaidya, P.: Geometry helps in matching. In: *STOC 1988: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 422–425. ACM, New York (1988)