Faster Bottleneck Non-crossing Matchings of Points in Convex Position

Marko Savić* Miloš Stojaković*†

Abstract

Given an even number of points in a plane, we are interested in matching all the points by straight line segments so that the segments do not cross. Bottleneck matching is a matching that minimizes the length of the longest segment. For points in convex position, we present a quadratic-time algorithm for finding a bottleneck non-crossing matching, improving upon the best previously known algorithm of cubic time complexity.

1 Introduction

Let P be the set of n points in the plane, where n is an even number. Let M be a perfect matching of points in P, using $n/2$ straight line segments to match the points, that is, each point in P is an endpoint of exactly one line segment. We forbid line segments to cross. Denote the length of a longest line segment in M with $bn(M)$, which we also call the *value* of M. We aim to find a matching that minimizes $bn(M)$. Any such matching is called *bottleneck matching* of P.

1.1 Related work

There is plentiful research on various geometric problems involving pairings without crossings. Some of considered problems examine matchings of various planar objects, see [7, 6, 13], while more basic problems involve matching pairs of points by straight line segments, see [4, 3, 5]. There is always a non-crossing matching of points with non-crossing segments, and moreover it is straightforward to prove that a matching minimizing the total sum of lengths of its segments has to be non-crossing.

In [10], Chang, Tang and Lee gave an $O(n^2)$-time algorithm for computing a bottleneck matching of a point set, but allowing crossings. This result was extended by Efrat and Katz in [12] to higher-dimensional Euclidean spaces.

Abu-Affash, Carmi, Katz and Trablesi showed in [2] that problem of computing non-crossing bottleneck matching of a point set is NP-complete and does not allow a PTAS. They gave a $2\sqrt{10}$ factor approximation algorithm, and also showed that the case where

*University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics. Partly supported by Ministry of Education and Science, Republic of Serbia. {marko.savic, milos.stojakovic}@dmi.uns.ac.rs

†Partly supported by Provincial Secretariat for Science, Province of Vojvodina.

all points are in convex position can be solved optimally in $O(n^3)$ time. In [1], Abu-Affash, Biniaz, Carmi, Maheshwari and Smid presented an algorithm for computing a non-crossing bottleneck plane matching of size at least $n/5$ in $O(n \log^2 n)$ time. They then extended it to provide an $O(n \log n)$-time approximation algorithm which computes a plane matching of size at least $2n/5$ whose edges have length at most $\sqrt{2}+\sqrt{3}$ times the length of a longest edge in a non-crossing bottleneck matching.

Bichromatic (sometimes also called bipartite) versions of the bottleneck matching problem, where only points of different colors are allowed to be matched, have also been studied. Efrat, Itai and Katz showed in [11] that a bottleneck matching between two point sets, with possible crossings, can be found in $O(n^{3/2} \log n)$ time. Bichromatic non-crossing bottleneck problem was proved to be NP-complete by Carlson, Armbruster, Bellam and Saladi in [9].

Biniaz, Maheshwari and Smid in [8] study special cases of non-crossing bichromatic bottleneck matchings. They show that the case where all points are in convex position can be solved in $O(n^3)$ time with an algorithm similar to the one for monochromatic case presented in [2]. They also consider the case where the points of one color lie on a line and all points of the other color are on the same side of that line, providing an $O(n^4)$ algorithm to solve it. The same results for these special cases are independently obtained in [9]. In [8] an even more restricted problem, a case where all points lie on a circle, is solved by constructing an $O(n \log n)$-time algorithm.

1.2 Monochromatic bottleneck non-crossing matchings for convex point sets and our results

In what follows we consider the case where all points of P are in convex position, i.e. they are the vertices of a convex polygon $\mathcal{P}$, and they are monochromatic, i.e. any two points from P can be matched. As we are going to deal with matchings without crossings, from now on, the word matching is used to refer only to pairings that are crossing-free.

Let us label the points $v_0, v_1, \ldots, v_{n-1}$ in positive (counterclockwise) direction. To simplify the notation, we will often use only the indices when referring to the vertices. We write $\{i, \ldots, j\}$ to represent the sequence $i, i+1, i+2, \ldots, j-1, j$, where all operations are calculated modulo n; note that i is not necessarily less than j, and $\{i, \ldots, j\}$ is not the same as $\{j, \ldots, i\}$. We say that (i, j) is a *feasible* pair if there exists a matching containing (i, j), which in this case simply means that $\{i, \ldots, j\}$ is of even size.

The problem of finding a bottleneck matching of points in convex position can be solved in polynomial time using dynamic programming algorithm, as presented in [2]. Similar algorithm for bichromatic case is presented in [8] and [9]. The algorithm is fairly straightforward, and we are going to describe it briefly.

The subproblems we consider are the tasks of optimally matching only the points in $\{i, \ldots, j\}$, where $i, j \in \{0, \ldots, n-1\}$ and $j - i$ is odd. Each matching M on $\{i, \ldots, j\}$ matches i with some $k \in \{i+1, \ldots, j\}$, where (i, k) is feasible. Segment (i, k) divides M in two parts, a matching on $\{i+1, \ldots, k-1\}$ and a matching on $\{k+1, \ldots, j\}$. If we solve those two parts optimally, we can combine them into an optimal matching of $\{i, \ldots, j\}$ that contains (i, k). We go through all the possibilities for k and take the best matching obtained in this way, yielding an optimal matching of points in $\{i, \ldots, j\}$. If we denote the value of this optimal matching by $b_{i,j}$, we get the following recursive formula,

$$b_{i,j} = \min_{k=i+1,i+3,\ldots,j} \begin{cases} |v_i v_j| & \text{if } j-i=1 \\ max\{|v_i v_k|, b_{k+1,j}\} & \text{if } k-i=1 \\ max\{|v_i v_k|, b_{i+1,k-1}\} & \text{if } k=j \\ max\{|v_i v_k|, b_{i+1,k-1}, b_{k+1,j}\} & \text{otherwise.} \end{cases}$$

This formula is then used to to fill in the dynamic programming table. There are $O(n^2)$ entries, and to calculate each we need $O(n)$ time. Therefore, the described algorithm finds a bottleneck matching for monochromatic points in convex position in $O(n^3)$ time.

In this paper, we present a faster algorithm for finding a bottleneck matching for monochromatic points in convex position, with only $O(n^2)$ time complexity. En route, we prove a series of results that give insights in the properties and structure of bottleneck matchings.

2 Structure of bottleneck matching

Our aim is to show the existence of a bottleneck matching with a certain structure that we can utilize to construct an efficient algorithm. We do so by proving a sequence of lemmas, with each lemma imposing an increasingly stronger condition on the structure.

Let us split all point pairs into the two categories. Pairs consisting of two neighboring vertices of $\mathcal{P}$ are called *edges*, and all other pairs are called *diagonals*. Each matching is, thus, comprised of edges and diagonals.

The *turning angle* of $\{i, \ldots, j\}$, denoted by $\tau(i, j)$, is the angle by which the vector $\overrightarrow{v_i v_{i+1}}$ should be rotated in positive direction to align with the vector $\overrightarrow{v_{j-1} v_j}$, see Figure 1.

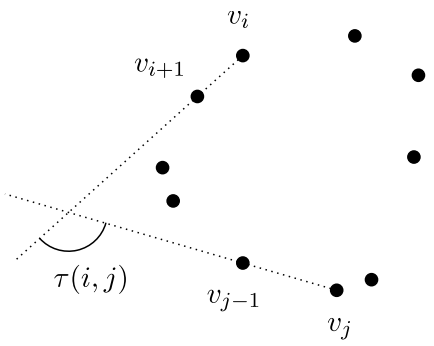

Figure 1: Turning angle.

Lemma 1. *There is a bottleneck matching M of P such that all diagonals $(i, j) \in M$ have $\tau(i, j) > \pi/2$.*

Proof. Suppose there is no such matching. Let M' be a bottleneck matching with the least number of diagonals. By assumption, there is a diagonal $(i, j) \in M'$ such that $\tau(i, j) \leq \pi/2$, see Figure 2(a). If we replace all pairs from M' lying in $\{i, \ldots, j\}$ with edges $(i, i+1), (i+2, i+3), \ldots, (j-1, j)$, we obtain a new matching M^*, see Figure 2(b). The diameter of $\{i, \ldots, j\}$, i.e. the longest distance between any pair of points from $\{i, \ldots, j\}$, is achieved by the pair (i, j), so $bm(M^*) \leq bm(M')$. Since M' is a bottleneck matching, $bm(M^*) = bm(M')$, meaning that M^* is a bottleneck matching as well. Diagonal (i, j)

belongs to M', but does not belong to M^*, so the new matching, M^*, has at least one diagonal less than M', which contradicts the assumption. □

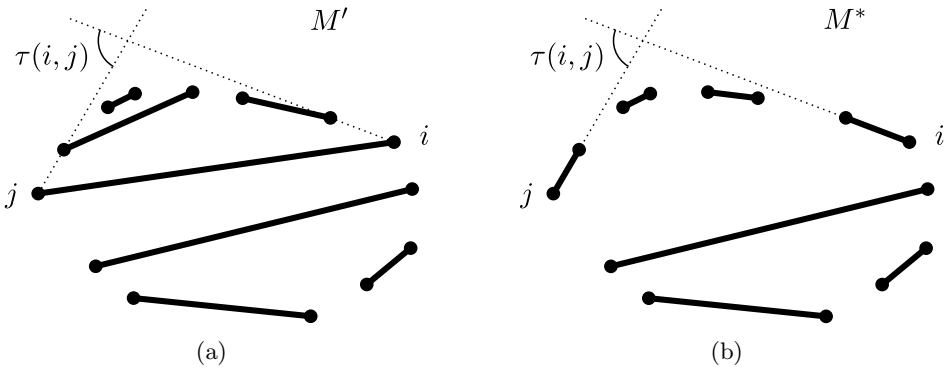

Figure 2: Matchings before (M') and after (M^*) the transformation.

Let us consider the division of the polygon $\mathcal{P}$ into regions obtained by cutting it along diagonals (but not edges) of the given matching M. Each region in this division is bounded by some diagonals of M and by some edges from the polygon's boundary. If there are exactly k diagonals bounding a region, we say the region is *k-bounded*. Any maximal sequence of diagonals connected by 2-bounded regions is called a *cascade*, see Figure 3.

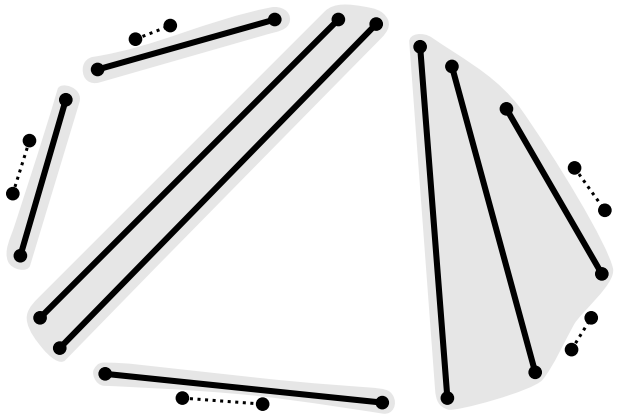

Figure 3: Diagonals inside each shaded area make a single cascade. There are three cascades with only one diagonal, one cascade with two diagonals, and one cascade with three diagonals.

Lemma 2. *There is a bottleneck matching having at most three cascades.*

Proof. Let M be a matching provided by Lemma 1, with turning angles of all diagonals greater than $\pi/2$. There cannot be a region bounded by 4 or more diagonals of M, since if it existed, the total turning angle would be greater than 2π. Hence, M only has regions with at most 3 bounding diagonals. Suppose there are two or more 3-bounded regions. We look at arbitrary two of them. There are two diagonals bounding the first region and two diagonals bounding the second region such that these four diagonals are in cyclical formation, meaning that each diagonal among them has other three on the same side. Applying the same argument once again we see that this situation is impossible because it yields turning angle greater than 2π. We conclude that there can be at most one 3-bounded region. □

The case of a bottleneck matching having exactly three cascades is possible, as shown in Figure 4.

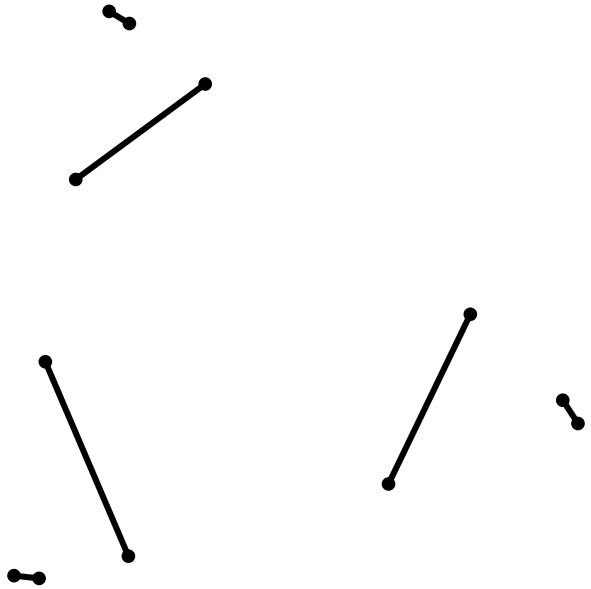

Figure 4: Configuration of points for which the only bottleneck matching has exactly three cascades.

Obviously, it is not possible for a matching to have exactly two cascades. So, from Lemma 2 we know that there is a bottleneck matching which either has at most one cascade and no 3-bounded regions, or it has a single 3-bounded region and exactly three cascades. In the following section we define a set of simpler problems that will be used to find an optimal solution in both of these cases.

3 Matchings with at most one cascade

When talking about matchings with minimal value under certain constraints, we will refer to these matchings as *optimal*.

For $j - i$ odd, let MATCHING(i, j) be the problem of finding an optimal matching $M_{i,j}$ of points $\{i, \ldots, j\}$, so that $M_{i,j}$ has at most one cascade, and the segment (i, j) belongs to a region bounded by at most one diagonal from $M_{i,j}$ different from (i, j).

If $j - i = 1$, then the solution to MATCHING(i, j) is exactly the edge (i, j). If $j - i > 2$, we consider the following cases. If there is a solution to MATCHING(i, j) that contains the pair (i, j), then $M_{i,j}$ can be constructed by taking (i, j) together with $M_{i+1,j-1}$. If not, then at least one of the edges $(i, i + 1)$ and $(j - 1, j)$ must be a part of $M_{i,j}$ (as otherwise points i and j would be endpoints of two different diagonals from $M_{i,j}$, neither of which is (i, j)), which is not allowed (by the requirement that the region containing (i, j) has at most one other bounding diagonal). If $(i, i+1) \in M_{i,j}$, then $M_{i,j}$ can be constructed from $M_{i+2,j}$ and the edge $(i, i+1)$. Similarly, if $(j-1, j) \in M_{i,j}$, then we can get $M_{i,j}$ as $M_{i,j-2}$ plus the edge $(j - 1, j)$.

Since these problems have optimal substructure, we can apply dynamic programming to solve them. If $bn(M_{i,j})$ is saved into $S[i, j]$, the following recurrent formula can be used

to calculate the solution to MATCHING(i, j) for all feasible pairs (i, j),

$$S[i,j] = \min \begin{cases} \max\{S[i+1, j-1], |v_i v_j|\} & \text{(1a)} \\ \max\{S[i+2, j], |v_i v_{i+1}|\} & \text{(1b)} \\ \max\{S[i, j-2], |v_{j-1} v_j|\}. & \text{(1c)} \end{cases}$$

Initially, we set $S[i, i] = 0$, for all i, and then we fill values in S in order of increasing $j - i$, so that all subproblems are already solved when needed.

Beside the value of a solution to MATCHING(i, j), it is going to be useful to determine if pair (i, j) is necessary for constructing $M_{i,j}$, i.e. we want to know if all the solutions to MATCHING(i, j) contain (i, j). If that is true then we call such a pair *necessary*. This can be easily incorporated into the calculation of $S[i, j]$. Namely, if case (1a) is the only one achieving minimum among cases (1a), (1b) and (1c), we set $necessary(i, j)$ to $\top$, otherwise we set it to $\bot$.

We have $O(n^2)$ subproblems, each of which takes $O(1)$ time to be calculated. Hence, all calculations together require $O(n^2)$ time and the same amount of space. To find the optimum for matchings with at most one cascade, we just find the minimum of all $S[i+1, i]$, and take any $M_{i+1,i}$ that achieves it. This step takes only linear time.

Note that we calculated only the values of solutions to all subproblems. If an actual matching is needed, it can be easily reconstructed in linear time from the data in S.

4 Finding bottleneck matching

As we concluded earlier, there is a bottleneck matching of P having either at most one cascade, or exactly three cascades. An optimal matching with at most one cascade can be found easily from calculated solutions to subproblems, as shown in the previous section. Next, we focus on finding an optimal matching among all matchings with exactly three cascades, denoted by 3-*cascade matchings* in the following text.

Any three distinct points i, j and k, where (i, j), $(j + 1, k)$ and $(k + 1, i - 1)$ are feasible pairs, can be used to construct a 3-cascade matching by simply taking a union of $M_{i,j}$, $M_{j+1,k}$ and $M_{k+1,i-1}$. To find the best one we could run through all possible triplets (i, j, k) and see which one minimizes $\max\{S[i, j], S[j + 1, k], S[k + 1, i - 1]\}$. However, that requires $O(n^3)$ time, and thus is not suitable, since our goal is to design a faster algorithm. Our approach is to show that instead of looking at all (i, j) pairs, it is enough to select (i, j) from a set of linear size, which would reduce the search space to quadratic number of possibilities, so the search would take only $O(n^2)$ time.

In a 3-cascade matching, let us call the three diagonals bounding the single 3-bounded region the *inner* diagonals.

Lemma 3. *If there is no bottleneck matching with at most one cascade, then there is a bottleneck* 3*-cascade matching whose every inner diagonal is necessary.*

Proof. Take any 3-cascade bottleneck matching M. If it has an inner diagonal (i, j) that is not necessary, then (by definition) there is a solution to MATCHING(i, j) that does not contain the pair (i, j) and has at most one cascade. We use that solution to replace all pairs from M that are inside $\{i, \ldots, j\}$, and thus obtain a new 3-cascade matching that does not contain the pair (i, j). Since M was optimal and there was at most one

cascade inside $\{i, \ldots, j\}$, replaced pairs were also a solution to MATCHING(i, j), so the new matching must have the same value as the original matching. And since there is no bottleneck matching with at most one cascade, the new matching must be a bottleneck 3-cascade matching as well. We repeat this process until all inner diagonals are necessary. The process has to terminate because the 3-bounded region is getting larger with each replacement. □

We say that (i, j) is a *candidate* diagonal, if it is a necessary diagonal and $\tau(i, j) \leq 2\pi/3$.

Lemma 4. *If there is no bottleneck matching with at most one cascade, then there is a 3-cascade bottleneck matching M, such that at least one inner diagonal of M is a candidate diagonal.*

Proof. Lemma 3 provides us with a 3-cascade matching M whose every inner diagonal is necessary. At least one of the 3 inner diagonals of M has turning angle at most $2\pi/3$, hence it is a candidate diagonal. Otherwise, the total turning angle would be greater than 2π, which is impossible. □

Let us now look at a candidate diagonal (i, j), and examine the position of points $\{i + 1, \ldots, j - 1\}$ relative to it. We construct the circular arc h on the right side of the directed line $v_i v_j$, from which the line segment $v_i v_j$ subtends an angle of $\pi/3$, see Figure 5. We denote the midpoint of h with A. Points v_i, A and v_j form an equilateral triangle, hence we are able to construct the arc a^- between A and v_i with the center in v_j, and the arc a^+ between A and v_j with the center in v_i. These arcs define three areas: Π^-, bounded by h and a^-, Π^+, bounded by h and a^+, and Π^0, bounded by a^-, a^+ and the line segment $v_i v_j$, all depicted in Figure 5.

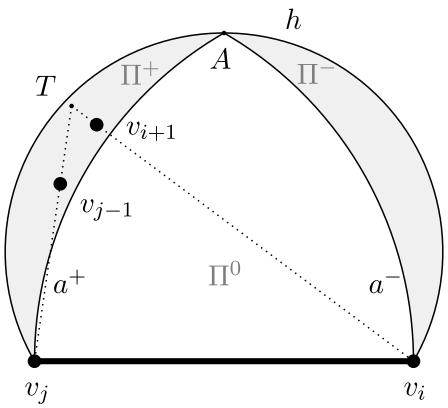

Figure 5: Points $v_{i+1}, \ldots, v_{j-1}$ all lie inside either Π^- or Π^+.

Lemma 5. *If (i, j) is a candidate diagonal, then points $v_{i+1}, \ldots, v_{j-1}$ either all belong to Π^- or all belong to Π^+.*

Proof. Let T be the point of intersection of lines $v_i v_{i+1}$ and $v_j v_{j-1}$, see Figure 5. Since $\tau(i, j) \leq 2\pi/3$, the point T lies in the area bounded by the line segment $v_i v_j$ and the arc h. Because of convexity, all points in $\{i, \ldots, j\}$ must lie inside the triangle $\triangle v_i T v_j$, so there cannot be two points from $\{i + 1, \ldots, j - 1\}$ such that one is on the right of the directed line $v_i A$ and the other is on the left of the directed line $v_j A$. This as well means

that either Π^- or Π^+ is empty. Without loss of generality, let us assume that there are no points from $\{i+1,\ldots,j-1\}$ in Π^-.

It remains to be proved that none of the points in $\{i+1,\ldots,j-1\}$ lies in Π^0. Suppose the opposite, that there is such a point in Π_0. Let k be the first index in the sequence $\{i,\ldots,j\}$ such that $v_k \in \Pi^+$, see Figure 6. Since (i,j) is a feasible pair, $\{i,\ldots,j\}$ is of even size, implying that the parity of the number of points in Π^+ is the same as the parity of the number of points in Π^0. (If there are points on a^+, we assign them to either region.)

If the number of points in Π^+, as well as in Π^0, is odd (not counting points v_i and v_j), see Figure 6(a), we make a matching using pairs $(i,i+1),(i+2,i+3),\ldots,(j-1,j)$. In the case the number is even, see Figure 6(b), we make a matching using pairs $(i,i+1),(i+2,i+3),\ldots,(k-3,k-2)$, pair $(k-1,j)$, and pairs $(k,k+1),(k+2,k+3),\ldots,(j-2,j-1)$.

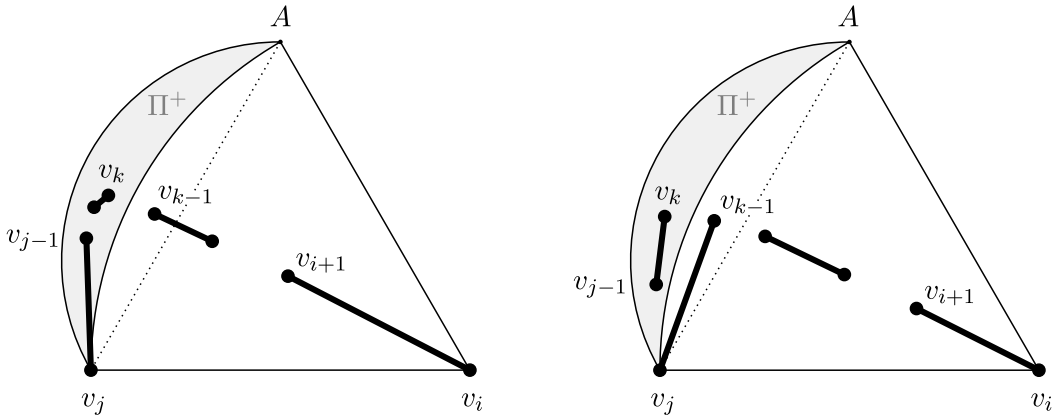

Figure 6: Matching points in $\{i,\ldots,j\}$, depending on the parity of their number.

For each matched pair in any of these two cases, points of the pair either both belong to Π^0, or they both belong to the area bounded by a^+ and the line segment Av_j. Both of these areas have diameter $|v_iv_j|$, so all matched pairs have distance not larger than $|v_iv_j|$. So, in each of the cases we constructed a matching $M_{i,j}$ of all points in $\{i,\ldots,j\}$ which does not contain (i,j), with $bn(M_{i,j}) \leq |v_iv_j|$. The matching also satisfies the condition for subproblem Matching(i,j), i.e. it has at most one cascade, and the pair (i,j) belongs to a region bounded by at most one diagonal from $M_{i,j}$ different from (i,j) (which can only be the diagonal $(v_{k-1}v_j)$ in the second case). Consequently, (i,j) cannot be a necessary diagonal, and, thereby, it can not be a candidate diagonal, leading to a contradiction with the assumption that there is a point from $\{i+1,\ldots,j-1\}$ in Π^0. □

With $\Pi^-(i,j)$ and $\Pi^+(i,j)$ we respectively denote areas Π^- and Π^+ corresponding to a candidate diagonal (i,j).

Two possibilities for a candidate diagonal (i,j) provided by Lemma 5 bring forth a concept of *polarity*. If points $\{i+1,\ldots,j-1\}$ lie in $P^-(i,j)$ we say that candidate diagonal (i,j) has *negative polarity* and has i as its *pole*. Otherwise, if these points lie in $P^+(i,j)$, we say that (i,j) has *positive polarity* and pole in j.

Lemma 6. *No two candidate diagonals of the same polarity can have the same point as a pole.*

Proof. Let us suppose the contrary, that is, that there are two candidate diagonals of the

same polarity with the same point as a pole. Assume, without loss of generality, that (i, k) and (j, k) are two such candidate diagonals, both with positive polarity, each having its pole in k. Without loss of generality, we also assume that $j \in \{i + 1, \ldots, k - 1\}$, see Figure 7.

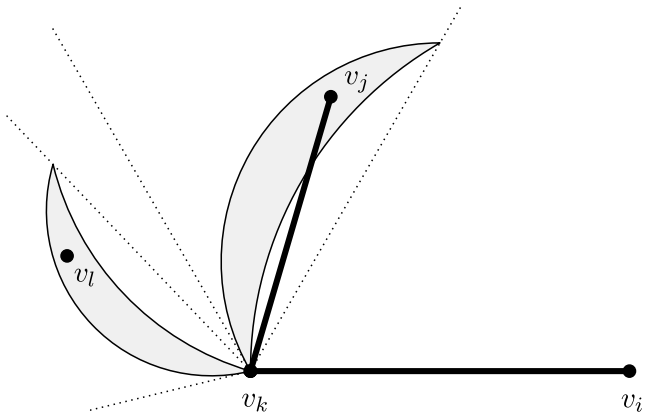

Figure 7: Two candidate diagonals of equal polarity having the same pole.

Area $\Pi^+(i, k)$ lies inside the angle with vertex v_k and sides at angles of $\pi/3$ and $2\pi/3$ with line $v_k v_i$. Similarly, $\Pi^+(j, k)$ lies inside the angle with vertex v_k and sides at angles of $\pi/3$ and $2\pi/3$ with line $v_k v_j$.

Since (j, k) is a diagonal, there is $l \in \{j + 1, \ldots, k - 1\}$. Points v_j and v_l lie in $\Pi^+(i, k)$ and $\Pi^+(j, k)$, respectively, meaning that $\pi/3 \leq \angle v_i v_k v_j, \angle v_j v_k v_l \leq 2\pi/3$, implying $2\pi/3 \leq \angle v_i v_k v_j + \angle v_j v_k v_l = \angle v_i v_k v_l \leq 4\pi/3$. This means that v_l does not belong to $\Pi^+(i, j)$. However, that is not possible, since $l \in \{i+1, \ldots, j-1\}$ as well, so we have a contradiction. □

As a simple corollary of Lemma 6, we get that there is at most linear number of candidate diagonals.

Lemma 7. *There are $O(n)$ candidate diagonals.*

Proof. Among all candidate diagonals of the same polarity no two can have a pole in the same point of P. Therefore, there are at most n candidate diagonals of the same polarity, and, consequently, at most $2n$ candidate diagonals in total. □

Finally, we combine our findings from Lemma 4 and Lemma 7, as described in the beginning of Section 4, to construct Algorithm 1.

Theorem 8. *Algorithm 1 finds the value of bottleneck matching in $O(n^2)$ time.*

Proof. The first step, calculating $S[i, j]$ and $necessary(i, j)$ for all (i, j) pairs, is done in $O(n^2)$ time, as described in Section 3. The second step finds the minimal value of all matchings with at most one cascade in $O(n)$ time.

The rest of the algorithm finds the minimal value of all 3-cascade matchings. Lemma 4 tells us that there is a bottleneck matching among 3-cascade matchings with one inner diagonal being a candidate diagonal, so the algorithm searches through all such matchings. We first fix the candidate diagonal (i, j) and then enter the inner for-loop, where we search

Algorithm 1 Bottleneck Matching

Calculate $S[i,j]$ and $necessary(i,j)$ for all feasible (i,j) pairs, as described in Section 3.
$best \leftarrow \min\{S[i+1,i] : i \in \{0,\ldots,n-1\}\}$
for all feasible (i,j) **do**
  if $necessary(i,j)$ and $\tau(i,j) \leq 2\pi/3$ **then**
    for $k \in \{j+1,\ldots,i-1\}$ such that $(j+1,k)$ is feasible **do**
      $best \leftarrow \min\{best, \max\{S[i,j], S[j+1,k], S[k+1,i-1]\}\}$
    end for
  end if
end for

for an optimal 3-cascade matching having (i,j) as an inner diagonal. Although the outer for-loop is executed $O(n^2)$ times, Lemma 7 guarantees that the if-block is entered only $O(n)$ times. The inner for-loop splits $\{j+1,\ldots,i-1\}$ in two parts, $\{j+1,\ldots,k\}$ and $\{k+1,\ldots,i-1\}$, which together with $\{i,\ldots,j\}$ make three parts, each to be matched with at most one cascade. We already know the values of optimal solutions for these three subproblems, so we combine them and check if we get a better overall value. At the end, the minimum value of all examined matchings is contained in *best*, and that has to be the value of a bottleneck matching, since we surely examined at least one bottleneck matching. □

Algorithm 1 gives only the value of a bottleneck matching, however, it is easy to reconstruct an actual bottleneck matching by reconstructing matchings for subproblems that led to the minimum value. This reconstruction can be done in linear time.

References

[1] A Karim Abu-Affash, Ahmad Biniaz, Paz Carmi, Anil Maheshwari, and Michiel Smid. Approximating the bottleneck plane perfect matching of a point set. *Computational Geometry*, 48(9):718 – 731, 2015.

[2] A Karim Abu-Affash, Paz Carmi, Matthew J Katz, and Yohai Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457, 2014.

[3] Oswin Aichholzer, Sergey Bereg, Adrian Dumitrescu, Alfredo García, Clemens Huemer, Ferran Hurtado, Mikio Kano, Alberto Márquez, David Rappaport, Shakhar Smorodinsky, Diane Souvaine, Jorge Urrutia, and David R Wood. Compatible geometric matchings. *Computational Geometry*, 42(6):617–626, 2009.

[4] Oswin Aichholzer, Sergio Cabello, Ruy Fabila-Monroy, David Flores-Penaloza, Thomas Hackl, Clemens Huemer, Ferran Hurtado, and David R Wood. Edge-removal and non-crossing configurations in geometric graphs. *Discrete Mathematics and Theoretical Computer Science*, 12(1):75–86, 2010.

[5] Noga Alon, Sridhar Rajagopalan, and Subhash Suri. Long non-crossing configurations in the plane. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 257–263. ACM, 1993.

[6] Greg Aloupis, Esther M Arkin, David Bremner, Erik D Demaine, Sándor P Fekete, Bahram Kouhestani, and Joseph SB Mitchell. Matching regions in the plane using non-crossing segments. EGC, 2015.

[7] Greg Aloupis, Jean Cardinal, Sébastien Collette, Erik D Demaine, Martin L Demaine, Muriel Dulieu, Ruy Fabila-Monroy, Vi Hart, Ferran Hurtado, Stefan Langerman, Maria Saumell, Carlos Seara, and Perouz Taslakian. Non-crossing matchings of points with geometric objects. *Computational geometry*, 46(1):78–92, 2013.

[8] Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Bottleneck bichromatic plane matching of points. Canadian Conference on Computational Geometry, 2014.

[9] John Gunnar Carlsson, Benjamin Armbruster, Haritha Bellam, and Rahul Saladi. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applications*, to appear.

[10] Maw-Shang Chang, Chuan Yi Tang, and Richard C. T. Lee. Solving the euclidean bottleneck matching problem by k-relative neighborhood graphs. *Algorithmica*, 8(1-6):177–194, 1992.

[11] Alon Efrat, Alon Itai, and Matthew J Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[12] Alon Efrat and Matthew J Katz. Computing euclidean bottleneck matchings in higher dimensions. *Information processing letters*, 75(4):169–174, 2000.

[13] Jan Kratochvíl and Torsten Ueckerdt. Non-crossing connectors in the plane. In *Theory and Applications of Models of Computation*, volume 7876 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2013.