Bottleneck Bichromatic Non-crossing Matchings using Orbits[*]

Marko Savić[1] and Miloš Stojakovic[2]

1,2 Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia.
{marko.savic, milos.stojakovic}@dmi.uns.ac.rs

Abstract

Given a set of n red and n blue points in the plane, we are interested in matching red points with blue points by straight line segments so that the segments do not cross. Bottleneck matching is such a matching that minimizes the length of the longest segment. We develop tools which enable us to solve the problem of finding bottleneck matchings of points in convex position in $O(n^2)$ time. We use the same approach to design an $O(n)$-time algorithm for the case where all points lie on a circle. Previously best known results were $O(n^3)$ for points in convex position, and $O(n \log n)$ for points on a circle.

1 Introduction

Let R and B be sets of n red and n blue points in the plane, respectively, with $P = R \cup B$. Let M be a perfect matching between points from R and B, using n straight line segments to match the points, that is, each point is an endpoint of exactly one line segment, and each line segment has one red and one blue endpoint. We forbid line segments to cross. The length of a longest line segment in M is called the *value* of M. Our goal is to find a matching under given constraints with the minimum value. Any such matching is called a *bottleneck matching* of P.

1.1 Related work

Monochromatic case The monochromatic variant of the problem is the case when points are not assigned colors, and any two points are allowed to be matched. The problem of computing bottleneck monochromatic non-crossing matching of a point set is shown to be NP-complete by Abu-Affash, Carmi, Katz and Trablesi in [2]. They also proved that it does not allow a PTAS, gave a $2\sqrt{10}$ factor approximation algorithm, and showed that the case where all points are in convex position can be solved exactly in $O(n^3)$ time. We improved this result in [6] by constructing $O(n^2)$-time algorithm. In [1], Abu-Affash, Biniaz, Carmi, Maheshwari and Smid presented an algorithm for computing a bottleneck monochromatic non-crossing matching of size at least $n/5$ in $O(n \log^2 n)$ time. They extended the same approach 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 the longest edge in a non-crossing bottleneck matching.

Bichromatic case The problem of finding a bottleneck bichromatic non-crossing matching (BBNCM) was proved to be NP-complete by Carlson, Armbruster, Bellam and Saladi in [4].

[*] Partly supported by Ministry of Education and Science, Republic of Serbia, and Provincial Secretariat for Science, Province of Vojvodina.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

But for the version where crossings are allowed, Efrat, Itai and Katz showed in [5] that a bottleneck matching between two point sets can be found in $O(n^{3/2}\log n)$ time.

Biniaz, Maheshwari and Smid in [3] studied special cases of BBNCMs. They showed that the case where all points are in convex position can be solved in $O(n^3)$ time, utilizing an algorithm similar to the one for monochromatic case presented in [2]. They also considered 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 [4]. In [3], an even more restricted problem is studied, a case where all points lie on a circle, for which an $O(n \log n)$-time algorithm is provided.

1.2 Our results

Here, we develop tools which enable us to solve the problem of finding a BBNCM of points in convex position in $O(n^2)$ time. Also, using the same toolset we design an optimal $O(n)$ algorithm for the case when the points lie on a circle.

Some important structural properties of BBNCMs of points in convex position that we aim to exploit are captured well by the concept of (what we refer to as) *orbit*. Informally speaking, orbits form a partition of the point set that turns out to have the following property – two differently colored points can be connected by a segment in some non-crossing matching if and only if they belong to to the same orbit.

As it turns out, there is a number of additional properties of orbits that we can put to good use, and once we combine them with ideas used to efficiently solve the monochromatic case in [6], we are able to get a considerable improvement of the algorithm running time, both in the convex case and in the case where all points lie on a circle.

For detailed exposition of our results and all the proofs, please refer to [7].

2 Orbits

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. Here we only deal with matchings without crossings, so from now on, the word matching is used to refer only to pairings that are crossing-free.

Let us label the points $v_0, v_1, \ldots, v_{2n-1}$ 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 $i, i+1, i+2, \ldots, j-1, j$. All operations are calculated modulo $2n$; note that i is not necessarily less than j, and that $\{i, \ldots, j\}$ 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). It can be shown that every set of n blue and n red points in a convex position can be perfectly matched, implying that a pair (i, j) is feasible iff i and j are of different colors and each of $\{i+1, \ldots, j-1\}$ and $\{j+1, \ldots, i-1\}$ contains the same number of red and blue points.

Let $o(i)$ be the first point j starting from i in positive direction such that (i, j) is feasible. The function o has an inverse, denoted by o^{-1}, and it is easy to see that $o^{-1}(j)$ is the first point i starting from j in the negative (clockwise) direction such that (i, j) is feasible. We define $o^0(i) := i$.

▶ **Definition 2.1.** An *orbit* of i, denoted by $\mathcal{O}(i)$, is defined by $\mathcal{O}(i) := \{o^k(i) : k \in \mathbb{Z}\}$, see Figure 1. By $\mathcal{O}(P)$ we denote the set of all orbits of a convex point set P, that is $\mathcal{O}(P) := \{\mathcal{O}(i) : i \in P\}$.

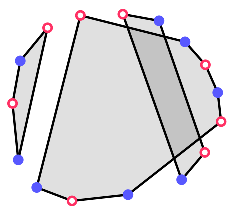

Figure 1 Orbits.

It can be seen that each point belongs to exactly one orbit. Any two neighboring points on an orbit have different colors, so each orbit has an equal number of red and blue points.

Next, we state a number of properties that can be shown to hold for orbits.

▶ **Property 2.2.** *Points i and j of different colors form a feasible pair iff $\mathcal{O}(i) = \mathcal{O}(j)$.*

If i and j are neighboring vertices of a convex polygon defined by the points of an orbit such that i precedes j in the positive direction, then $j = o(i)$. Informally, by repeatedly applying function o we visit all points of an orbit in a single turn around the polygon.

All feasible point pairs can be split into the two categories depending on their mutual position in their orbit. Pairs consisting of two neighboring vertices of the orbit are called *edges*, and all other pairs are called *diagonals*. More precisely, for $j \in \mathcal{O}(i)$, (i, j) is an edge if and only if $i = o(j)$ or $j = o(i)$, otherwise, it is a diagonal.

▶ **Lemma 2.3.** *Orbits can be computed in $O(n)$ time.*

We say that an edge $(i, o(i))$ is a *red-blue* edge if $i \in R$, and *blue-red* edge if $i \in B$. We consider edges directed from i to $o(i)$, so points right of an edge $(i, o(i))$ are points $\{i, \ldots, o(i)\} \setminus \{i, o(i)\}$.

▶ **Property 2.4** (Orbit synchronicity). *Let $\mathcal{A}, \mathcal{B} \in \mathcal{O}(P)$. There are no points of $\mathcal{B}$ on the right side of red-blue edges of $\mathcal{A}$ if and only if there are no points of $\mathcal{A}$ on the right of blue-red edges of $\mathcal{B}$.*

▶ **Definition 2.5.** Let $\mathcal{A}, \mathcal{B} \in \mathcal{O}(P)$. We say that $\mathcal{A} \leq \mathcal{B}$ iff there are no points of $\mathcal{B}$ right of red-blue edges of $\mathcal{A}$ and no points of $\mathcal{A}$ right of blue-red edges of $\mathcal{B}$

It can be proven that the relation $\leq$ on orbits is transitive, which together with orbit synchronicity gives us the following important property of orbits.

▶ **Property 2.6.** *The relation $\leq$ on $\mathcal{O}(P)$ is a total order.*

2.1 Orbit graph

The orbit graph for P is a directed acyclic graph in which each vertex corresponds to an orbit in $\mathcal{O}(P)$, and there is a directed edge from $\mathcal{A}$ to $\mathcal{B}$ iff $\mathcal{A} \leq \mathcal{B}$ and orbits $\mathcal{A}$ and $\mathcal{B}$ are different and intersect each other (meaning that there is a line segment between a pair in $\mathcal{A}$ and a line segment between a pair in $\mathcal{B}$ so that those line segments intersect).

▶ **Property 2.7.** *Each weakly connected component of the orbit graph has a Hamiltonian path.*

These Hamiltonian paths are possible to calculate without constructing the full orbit graph first, as the following lemma states.

▶ **Lemma 2.8.** *All Hamiltonion paths of connected components of the orbit graph can be computed in $O(n)$ total time.*

3 Finding bottleneck matchings

3.1 Cascades

Now, what remains is to utilize the theory that we developed for orbits and the orbit graph, combining it with some parts of the approaches used in [6] to tackle the monochromatic case.

Let us consider the division of the polygon defined by points in P into regions obtained by cutting it with diagonals (but not edges) of the given matching M. Each region is bounded by some diagonals of M and by the polygon's boundary. We call a region k-*bounded* if there are exactly k diagonals bounding it. Any maximal sequence of diagonals connected by 2-bounded regions is called a *cascade*, see Figure 2. We can prove the following lemma.

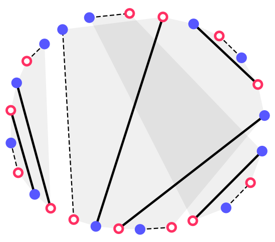

Figure 2 Matching consisting of edges (dashed lines) and diagonals (solid lines). There are three cascades in this example: one consist of the three diagonals in the left part, one consist of the two diagonals in the lower right, and one consist of the single diagonal in the upper right.

▶ **Lemma 3.1.** *There is a bottleneck matching having at most three cascades.*

It is not possible for a matching to have exactly two cascades, so we know that there is a bottleneck matching either with at most one cascade, or with exactly three cascades. We define a set of subproblems that is used to find an optimal solution in both of these cases.

3.2 Subproblems

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

Let (i, j) be such that $\{i, \dots, j\}$ contains the same number of red and blue points. We define $\textsc{Matching}(i, j)$ to be the problem of finding an optimal matching $M_{i,j}$ of points $\{i, \dots, j\}$, so that $M_{i,j}$ has at most one cascade, and pair (i, j) belongs to a region bounded by at most one diagonal from $M_{i,j}$ different from (i, j).

All these subproblems can be solved in $O(n^2)$ total time using dynamic programming. Beside the value $S_{i,j}$ of matching $M_{i,j}$, we determine if pair (i, j) is necessary for constructing $M_{i,j}$, i.e. do all solutions to $\textsc{Matching}(i, j)$ contain (i, j). If that is true then such a pair is called *necessary*. This can be easily calculated together with the solution to subproblems.

An optimal matching of the whole set P having at most one cascade can be found in linear time from calculated solutions to subproblems. We run through all subproblems of the form $\textsc{Matching}[i + 1, i]$ for all feasible pairs $(i, i + 1)$, and take the minimum.

Next, we focus on finding an optimal matching among all matchings with exactly three cascades (3-*cascade matchings*). 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 taking a union of $M_{i,j}$, $M_{j+1,k}$ and $M_{k+1,i-1}$. We can 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.

3.3 Candidate pairs and polarity

In 3-cascade matching, we call the three diagonals at the inner ends of the three cascades the *inner diagonals*. We take the largest region by area, such that it is bounded, but not crossed by matched pairs, and such that each two of the three cascades are separated by that region. We call this region the *inner region*. Matched pairs defining the boundary of the inner region are called the *inner pairs*.

▶ **Lemma 3.2.** *If there is no bottleneck matching with at most one cascade, then there is a bottleneck* 3*-cascade matching whose every inner pair is necessary.*

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 vector $\overrightarrow{v_{j-1} v_j}$, see Figure 3.

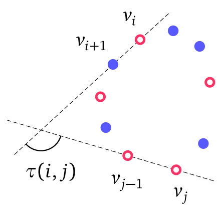

Figure 3 Turning angle.

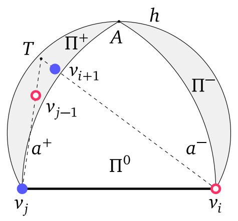

Figure 4 $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$ all lie inside either Π^- or Π^+.

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

▶ **Lemma 3.3.** *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 pair of M is a candidate pair.*

Let us now look at a candidate pair (i, j), and examine the position of points $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$. 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 4. Let A be the midpoint of h. Points v_i, A and v_j form an equilateral triangle, so we can 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 $v_i v_j$. With $\Pi^-(i, j)$ and $\Pi^+(i, j)$ we respectively denote areas Π^- and Π^+ corresponding to the candidate pair (i, j).

▶ **Lemma 3.4.** *If (i, j) is a candidate pair, then points $\{v_{i+1}, \ldots, v_{j-1}\} \cap \mathcal{O}(i)$ either all belong to Π^- or all belong to Π^+.*

Two possibilities for a candidate pair (i, j) provided by Lemma 3.4 bring forth a concept of *polarity*. If points $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$ lie in $\Pi^-(i, j)$ we say that candidate pair (i, j) has *negative polarity* and has i as its *pole*. Otherwise, if these points lie in $\Pi^+(i, j)$, we say that (i, j) has *positive polarity* and pole in j. The following lemma gives us the crucial observation about polarity, which enables us to limit the search space of the algorithm.

▶ **Lemma 3.5.** *No two candidate pairs of the same polarity have the same point as a pole. Hence, there are $O(n)$ candidate pairs.*

Finally, we use our findings from Lemma 3.3 and Lemma 3.5, to construct an algorithm which finds a BBNCM in $O(n^2)$ time. We first solve all subproblems and find candidate pairs. Then, we minimize $\max\{S[i,j], S[j+1,k], S[k+1,i-1]\}$ by running through all candidate pairs (i,j) and for each such pair through all $k \in \{j+1,\ldots,i-1\}$.

4 Points on a circle

Now, let us consider the special case where all points lie on a circle. The geometry of a circle provides us with the following lemma (also stated in [3]), which together with orbit properties enables us to construct an $O(n)$ time algorithm for this problem.

▶ **Lemma 4.1.** *There is a bottleneck matching in which each point i is connected either to $o(i)$ or $o^{-1}(i)$.*

This means there is a bottleneck matching M_E which can be constructed by taking alternating edges from each orbit, that is from each orbit we take either all red-blue or all blue-red edges. To find a bottleneck matching we want to search through only such matchings, and to reduce the number of possibilities, we use the properties of the orbit graph.

Consider the Hamiltonian path $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_m$ for some connected component of the orbit graph, as provided by Property 2.7. Since there is a directed edge from $\mathcal{L}_k$ to $\mathcal{L}_{k+1}$, those two orbits intersect each other, and by Property 2.4 we know that only edges from $\mathcal{L}_k$ that intersect $\mathcal{L}_{k+1}$ are blue-red edges, and only edges from $\mathcal{L}_{k+1}$ that intersect $\mathcal{L}_k$ are red-blue edges. Hence, M_E cannot have blue-red edges from $\mathcal{L}_k$ and red-blue edges from $\mathcal{L}_{k+1}$. This further implies that there is $l \in 0, 1, \ldots, m$ such that $\mathcal{L}_1, \ldots, \mathcal{L}_l$ all contribute to M_E with red-blue edges and $\mathcal{L}_{l+1}, \ldots, \mathcal{L}_m$ all contribute to M_E with blue-red edges.

For each l, we can compute the longest red-blue edge in $\mathcal{L}_1, \ldots, \mathcal{L}_l$ and the longest blue-red edge in $\mathcal{L}_{l+1}, \ldots, \mathcal{L}_m$, and computing all of this can be done in $O(n)$ total time. After we obtain these values, we can quickly get the value of a matching for each possible l, and take the one with the minimum value.

By Lemmas 2.3 and 2.8, each step in this process has time complexity not greater than $O(n)$, so we get an algorithm for points on a circle which runs 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 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Bottleneck bichromatic plane matching of points. Canadian Conference on Computational Geometry, 2014.

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

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

6 Marko Savić and Miloš Stojaković. Faster bottleneck non-crossing matchings of points in convex position. *Computational Geometry*, 65:27–34, 2017.

7 Marko Savić and Miloš Stojaković. Bottleneck bichromatic non-crossing matchings using orbits. *ArXiv e-prints*, 2018. `arXiv:1802.06301`.