Computational Geometry 65 (2017) 27–34

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Faster bottleneck non-crossing matchings of points in convex position

Marko Savić *,[1], Miloš Stojaković [1,2]

University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Serbia

ARTICLE INFO

Article history:
Received 16 February 2016
Received in revised form 11 April 2017
Accepted 26 April 2017
Available online 4 May 2017

Keywords:
Bottleneck
Matchings
Non-crossing
Convex position

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 a 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 the 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 the 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 all points are in convex position can be solved exactly in $O(n^3)$ time. In [1], Abu-Affash, Biniaz, Carmi,

* Corresponding author.
E-mail addresses: marko.savic@dmi.uns.ac.rs (M. Savić), milos.stojakovic@dmi.uns.ac.rs (M. Stojaković).

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

http://dx.doi.org/10.1016/j.comgeo.2017.05.002
0925-7721/

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 Carlsson, Armbruster, Bellam and Rahul 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,

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

This formula is then used 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.

On several occasions it will be useful to discern between the two possible orientations of a diagonal. Although in most of the paper we do not worry about the order of i and j in the pair (i,j), we will add the qualifier "oriented" whenever the distinction between (i,j) and (j,i) is important.

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