University of Novi Sad
Faculty of Sciences
Departman of Mathematics and Informatics

Marko Savić

Efficient algorithms for discrete geometry problems

— Ph.D. thesis —

Efikasni algoritmi za probleme iz diskretne geometrije

— Doktorska disertacija —

Novi Sad, 2018.

Contents

Introduction **1**

Thesis summary 3

I Bottleneck matchings **7**

Background and related work 9

1 Monochromatic bottleneck matchings **13**

1.1 Introduction 14

1.1.1 Problem statement 14

1.1.2 Our results 14

1.1.3 A simple solution 15

1.2 Structure of bottleneck matching 15

1.3 Finding bottleneck matchings 18

1.3.1 Matchings with at most one cascade 18

1.3.2 Matchings with three cascades 19

2 Bichromatic bottleneck matchings **25**

2.1 Introduction 26

2.1.1 Problem statement 26

2.1.2 Our results 26

2.2 Preliminaries 27

2.3 Orbits and their properties 28

2.3.1 Orbit graphs 38

2.4 Finding bottleneck matchings 40

2.4.1 Matchings with at most one cascade 42
2.4.2 Matchings with three cascades 45
2.5 Points on a circle 54

II Network dilations and feed-links 57

Background and related work 59

3 Optimal feed-link placement 63

3.1 Introduction 64
3.1.1 Problem statement 64
3.1.2 Our results 64
3.2 Preliminaries 65
3.3 Another view of the problem 66
3.4 Sliding lever algorithm 69
3.4.1 Lever 69
3.4.2 States 70
3.4.3 Jumping and retargeting 72
3.4.4 Events 76
3.4.5 Sequence of realized states 82
3.5 Merging the two dilations 85
3.6 Conclusion and further work 89

Prošireni izvod (in Serbian) 93

Bibliography 111

Short biography 117

Abstract

The first class of problem we study deals with geometric matchings. Given a set of points in the plane, we study perfect matchings of those 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 are interested in finding a bottleneck matching of points in convex position.

In the monochromatic case, where any two points are allowed to be matched, we give an $O(n^2)$-time algorithm for finding a bottleneck matching, improving upon previously best known algorithm of $O(n^3)$ time complexity.

We also study a bichromatic version of this problem, where each point is colored either red or blue, and only points of different color can be matched. We develop a range of tools, for dealing with bichromatic non-crossing matchings of points in convex position. Combining that set of tools with a geometric analysis enable us to solve the problem of finding a bottleneck matching in $O(n^2)$ time. We also design an $O(n)$-time algorithm for the case where the given 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.

The second class of problems we study deals with dilation of geometric networks. Given a polygon representing a network, and a point p in the same plane, we aim to extend the network by inserting a line segment, called a feed-link, which connects p to the boundary of the polygon. Once a feed link is fixed, the geometric dilation of some point q on the boundary is the ratio between the length of the shortest path from p to q through the extended network, and their Euclidean distance. The utility of a feed-link is inversely proportional to the maximal dilation over all boundary points.

We give a linear time algorithm for computing the feed-link with the minimum overall dilation, thus improving upon the previously known algorithm of complexity that is roughly $O(n \log n)$.

Izvod (in Serbian)

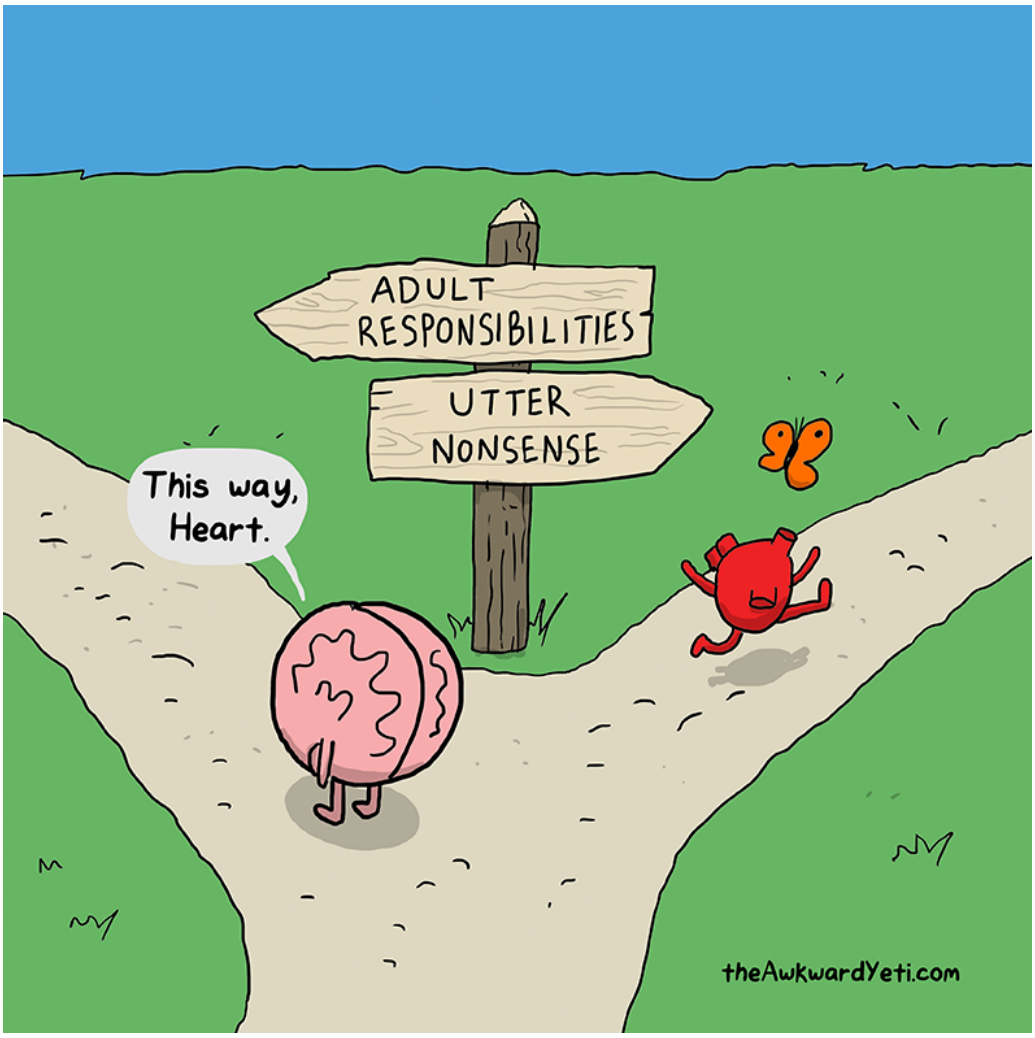
Prva klasa problema koju proučavamo tiče se geometrijskih mečinga. Za dat skup tačaka u ravni, posmatramo savršene mečinge tih tačaka spajajući ih dužima koje se ne smeju seću. Bottleneck mečing je takav mečing koji minimizuje dužinu najduže duži. Naš cilj je da nađemo bottleneck mečing tačaka u konveksnom položaju.

Za monohromatski slučaj, u kom je dozvoljeno upariti svaki par tačaka, dajemo algoritam vremenske složenosti $O(n^2)$ za nalaženje bottleneck mečinga. Ovo je bolje od prethodno najbolji poznatog algoritam, čija je složenost $O(n^3)$.

Takođe proučavamo bihromatsku verziju ovog problema, u kojoj je svaka tačka obojena ili u crveno ili u plavo, i dozvoljeno je upariti samo tačke različite boje. Razvijamo niz alata za rad sa bihromatskim nepresecajućim mečinzima tačaka u konveksnom položaju. Kombinovanje ovih alata sa geometrijskom analizom omogućava nam da rešimo problem nalaženja bottleneck mečinga u $O(n^2)$ vremenu. Takođe, konstruišemo algoritam vremenske složenosti $O(n)$ za slučaj kada sve date tačke leže na krugu. Prethodno najbolji poznati algoritmi su imali složenosti $O(n^3)$ za tačke u konveksnom položaju i $O(n \log n)$ za tačke na krugu.

Druga klasa problema koju proučavamo tiče se dilacije u geometrijskim mrežama. Za datu mrežu u obliku poligona, i tačku p u istoj ravni, želimo proširiti mrežu dodavanjem duži zvane feed-link koja povezuje p sa obodom poligona. Kada je feed-link fiksiran, definišemo geometrijsku dilaciju neke tačke q na obodu kao odnos između dužine najkraćeg puta od p do q kroz proširenu mrežu i njihovog Euklidskog rastojanja. Korisnost feed-linka je obrnuto proporcionalna najvećoj dilaciji od svih tačaka na obodu poligona.

Konstruišemo algoritam linearne vremenske složenosti koji nalazi feed-link sa najmanom sveukupnom dilacijom. Ovim postižemo bolji rezultat od prethodno najboljeg poznatog algoritma složenosti približno $O(n \log n)$.


ADULT RESPONSIBILITIES
UTTER NONSENSE
This way, Heart.
theAwkwardYeti.com

Zahvalnice (in Serbian)

Bitna etapa mog života konačno je privedena kraju. Razne stvari i dešavanja su taj moj put ka zvanju doktora nauka učinili vrednijim od samog postignuća. Ali od najveće vrednosti uvek će mi biti ljudi koji su bili deo svega toga. Neki od njih su pomogli direktno, a neki “samo” tako što su bili tu da sa njima delim i sreću i tugu.

Kako svega ovog ne bi bilo da nije mog mentora, a slobodan sam reći i druga, Miloša, njemu dugujem ogromnu zahvalnost. Imao sam tu privilegiju da u toku našeg poznanstva, starog već više od jedne decenije, od njega naučim nebrojeno mnogo korisnih stvari, od kojih neke čak imaju veze i sa naukom. Cenim tvoju toleranciju i odlično izlaženje na kraj sa mojim specifičnostima, i zahvalan sam ti za sve vreme koje si mi posvetio, bilo kroz saradnju ili kroz druženje.

Energiju potrošenu napornim radom nadoknađivali su mi moji drugari, Sova, Tigar, Nikola, Rajkoni, Nemanja, Trka i Let (po redosledu pojavljivanja). Bez obzira na to što oni i nisu bogzna koliko uticali na moj put ka ostvarivanju akademskih ciljeva, uvek su bili uz mene, kako na ovom tako i na svakom drugom mom putu. U stizanju na cilj leži samo deo zadovoljstva, a prava sreća je u putovanju i onome što je uz put. Delići života provedeni sa vama su mi najdragoceniji u njemu.

Iskreno i neizmerno sam zahvalan i svojoj Danici, koja već dugo nalazi razumevanje za moje mane i nestašluke. Mnoge trenutke koji bi bez tebe bili sasvim obični načinila si lepim, a u teškim trenucima bila si mi najveća podrška.

Od svih, ipak, najviše su za mene učinili moji roditelji, i njima dugujem najveću zahvalnost zbog bezuslovne ljubavi koju su mi pružili. Jedino zbog njihovog nesebičnog odricanja sam uvek imao mogućnost da sam sebi biram put. Ako je neko zaslužan za moja postignuća, to su oni, i zato njima posvećujem ovaj svoj mali životni uspeh.

Introduction

The time complexity of a computational problem is the smallest time complexity among all possible algorithms that can solve that problem. Time complexity is one of key instruments for understanding the inherent difficulty of an individual problem. Excluding the problems for which a linear time solution is known, the exact value of time complexity is known only for a very narrow variety of computational problems, almost none of which can be described as “natural”. Similarly, non-trivial lower bounds on problem time complexity very rarely can be determined exactly. If we know some useful lower bounds, they almost surely come with a restriction to some specific computational models. The well-known lower bound of $O(n \log n)$ for comparison sorts is one example. On the other hand, all that is needed to give an upper bound on time complexity is to construct an algorithm that solves the problem in question. With respect to this, constructing better algorithms yields better upper bounds for problem complexity. As such, the design of efficient algorithms is a discipline of great theoretical importance, apart from the obvious practical importance of being able to solve a problem quickly.

The field of discrete geometry studies geometric structures and properties which are inherently combinatorial in their nature. Computational geometry is a narrower field, researching algorithmic problems on discrete geometric structures. A good introduction into these fields can be found in [38], [26] and [27]. The studied structures are embedded in metric spaces – most often it is either a Euclidean space of two or more dimensions, or L1 metric or its generalization, fixed orientation metric. Geometric structures that are used can be as simple as a set of points, or more complex such as geometric graphs – graphs whose vertices and edges can be various geometric objects adhering to problem specific constraints.

The goal of this thesis is to determine new upper bounds for time complexity of some computational geometry problems by designing efficient algorithms that solve those problems.

One of the computational geometry problem clases which will be studied here is a class of geometric matching problems. These problems deal with partitioning different geometric objects in pairs under specified constraints, in order to optimize the certain measure of that pairing. In the most fundamental setting, the objects in question are points in the Euclidean plane, which can be matched using straight line segments, and the goal is to optimize some function on the lengths of segments. The most commonly used constraint is that no two segments can intersect, that is, the matching should be *crossing-free*. For example, we can ask to match the points from the given set of points in the plane by straight line segments, so that the total length of all segments is minimized.

Matching problems studied in this thesis are concerned with the so-called *bottleneck* measure, which represents the length of the longest line segment in the matching. The corresponding optimization problem is to find a matching with the bottleneck measure as small as possible. Any such matching is called *bottleneck matching*. It is known that the problem of finding bottleneck non-crossing matching of the point set in the plane is NP-complete problem, hence the current research on that topic is directed toward finding approximation algorithms for the general case and efficient exact algorithms for solving some special cases of the problem.

Additional variations on the geometric matching problems that will be of interest here are bichromatic (or bipartite) matchings. In such matchings, the objects to be matched are partitioned into two sets, and there is a requirement that no two matched objects can belong to the same set. Like the monochromatic version, the problem of finding bichromatic bottleneck non-crossing matching is NP-complete, so our focus will be on finding efficient solutions to certain special cases.

Another class of computational geometry problems which be studied in this thesis is a class of problems related to dilation and spanning factors of geometric networks. A *geometric network*, or just network, is a geometric graph embedded in a metric space M, whose vertices are points and edges are straight line segments between them. For two given points p and q lying on the edges or coinciding with the vertices of the network G, we define their network distance to be the length of the shortest curve connecting p and q which is fully contained in G. *Dilation* between p and q is the ratio between their network distance and their distance in the metric M. Dilation of the whole network G is the maximum dilation over all pair of points in that network, points on edges included. If we calculate the maximum only over pairs of points corresponding to vertices of the network, then this maximum value is called the *spanning factor* of the network G. Intuitively, dilation and spanning factor are measures of how good an approximation is G for complete graph over the metric space, and complete graph over the set of vertices of G, respectively.

The central problem concerning dilation and spanning factor of networks is their computation in different metrics for general networks and for special network types, such as paths, trees and polygons. Another important direction of research on this topic is about network modifications under given constraints in order to minimize or maximize dilation or the spanning factor. These modifications include adding or removing additional vertices or edges to the network. In this thesis, we will study a problem of extending a polygonal network with an aditional vertex.

Thesis summary

The following is a short overview of the presented results, with the references to their location in the thesis.

These results are joint work with Miloš Stojaković, and can be found in articles published in journals, conference proceedings and preprints [43, 44, 40, 41, 42].

Part I

The first part of a thesis deals with bottleneck non-crossing matchings of points in convex position. We design several algorithms for related problems, both in monochromatic and bichromatic case.

Chapter 1

In this chapter our goal is to design an efficient algorithm for the monochromatic version of the problem.

In Section 1.2 we show that there is always a bottleneck matching having a certain structure that we will be able to exploit. In Section 1.3.1 we define restricted subproblems and show how to efficiently solve all of them. Then, in Section 1.3.2 we use the observed structure to split the problem into subproblems. Doing so naively does not give us the wanted time complexity, so we find a way to reduce the search space, thus finally achieving $O(n^2)$-time complexity.

The results presented in this chapter are published in the journal of Computational Geometry: Theory and Applications [43], and the conference proceedings of the 32nd European Workshop on Computational Geometry (EuroCG'16) [41].

Chapter 2

In this chapter we consider bichromatic version of the problem. To be able to relate bichromatic and monochromatic versions, we develop a theory of orbits.

In Section 2.3 we define structures called orbits and derive numerous properties that hold for them. We observe the relationships between orbits, which enables us to define the orbit graph and to show some of its properties, also noting that the developed theory can be of independent interest when studying bichromatic non-crossing matchings.

In Section 2.4 we apply this theory of orbits to the ideas used for the monochromatic version of the problem. The result is an $O(n^2)$-time algorithm for finding a bichromatic bottleneck non-crossing matching of points in convex position.

In Section 2.5 we consider the variation of the previous problem where all the points lie on a circle. Geometry of a circle provides us with certain insights about bottleneck matchings, and we employ the properties of orbits and orbit graph once again, but in a different manner, to construct an $O(n)$-time algorithm that solves this problem.

The results presented in this chapter are published in the conference proceedings of the 34th European Workshop on Computational Geometry (EuroCG'18) [42], and can be found in the preprint [44].

Part II

The second part of the thesis deals with dilations of geometric networks. For a given polygonal network P and a point p not on P, we want to extend the network by connecting p to a point on P using a single straight line segment called feed-link. The goal is to do this in such a way to minimize the maximum dilation from p to any other point on P.

Chapter 3

In Section 3.2 we split the problem into two symmetrical components, the left and the right dilation. Our further analysis will be performed only on one of them.

In Section 3.3 we give a different view on the problem, using the plot of the distance function from p to the points on P. We introduce a key object, the *lever*, which is a line segment defined on the plot, having the slope inversely proportional to the left dilation.

In Section 3.4 we use a sweep algorithm simulating the lever movement on the plot, which is equivalent to the movement of a connection point around P. We do this once for the left dilation and once for the right dilation, and then in Section 3.5 we show how to merge our findings from those two sweeps in order to obtain the solution that achieves the minimum dilation.

The results presented in this chapter are published in the journal of Computational Geometry: Theory and Applications [40].

Part I

Bottleneck matchings

Background and related work

Geometric matchings are widely researched. In the most general setting, various planar objects are matched, see [11, 12, 37], or various planar objects can be used as edges in matchings, see [1, 29, 17, 15, 19]. Several papers, see [8, 9, 10], deal with matching points by straight line segments.

Most commonly, crossings in the geometric matchings are not allowed. 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.

Variations on these problem are plentiful. We are mostly interested in non-crossing matchings of points by straight line segments in such a way that the length of the longest segment is minimized. Such matchings are called *bottleneck matchings*.

The monochromatic variants of matching problems have no restrictions on which points can be matched, as long as other possible constraints are satisfied (such as no crossings constraint). In the bichromatic (sometimes also called bipartite) versions of matching problems, points are partitioned into two sets (each colored with its own color), and only points of different colors are allowed to be matched.

Monochromatic case. In [23], 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 [33] to higher-dimensional Euclidean spaces.

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 [5]. 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 [43] by constructing $O(n^2)$-time algorithm.

In [4], 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.

The so called *containment problems* are closely related to the problems of explicitly finding these matchings. In containment problems we want to find a geometric graphs that has a matching with certain properties as a subgraph. For example, in [29] Dillencourt proved that a perfect matching is contained in the Delaunay triangulation of the point set. In [35], Kaiser, Saumell and Cleemput proved that 10-Gabriel graph (10-GG) contains a bottleneck Hamiltonian cycle of P, which implies that there is a perfect matching in 10-GG. On the other hand, they showed that there might be no bottleneck Hamiltonian cycle in 5-GG.

Even when we tighten the constraints on allowed matchings, it is still true that some interesting geometric graphs have them as subgraphs. Chang, Tang and Lee in [23] showed that bottleneck mathing of a point set is contained in 16-relative neighborhood graph (16-RNG), and thus in 16-GG. Biniaz, Maheshwari, Smid improved this in [17] and [18] by showing that 9-GG contains perfect bottleneck matching, but it is possible that there is no such matching in 8-GG.

Bichromatic case. The problem of finding a bottleneck bichromatic non-crossing matching was proved to be NP-complete by Carlson, Armbruster, Bellam and Saladi in [22]. But for the version where crossings are allowed, Efrat, Itai and Katz showed in [32] that a bottleneck matching between two point sets can be found in $O(n^{3/2}\log n)$ time.

Biniaz, Maheshwari and Smid in [20] studied special cases of bottleneck bichromatic non-crossing matchings. 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 [5]. 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 [22]. An even more restricted problem is studied in [20], a case where all points lie on a circle, for which an $O(n\log n)$-time algorithm is given.

A variant of the bichromatic case is the so-called bicolored (or multicolored, when there are arbitrary many colors) case, where only the points of the same color are allowed to be matched. Abu-Affash, Bhore and Carmi in [2] examined bicolored matchings that minimize the number of crossings between edges matching different color sets. They presented an algorithm to compute a bottleneck matching of points in convex position among all matchings that have no crossings of this kind.

Apart from matchings, there is an interest in finding a bottleneck instances of other structures, such as Steiner trees. In [16], Biniaz, Maheshwari and Smid constructed an $O(n \log n)$ algorithm for computing the bottleneck full Steiner tree in monochromatic setting, and later Abu-Affash, Bhore, Carmi and Chakraborty in [3], presented an $O(n \log n)$-time algorithm for computing the bottleneck full Steiner tree in bichromatic setting.

Chapter 1

Monochromatic bottleneck matchings

1.1 Introduction

1.1.1 Problem statement

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.

value of matching

bottleneck matching

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

feasible pair

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 [5]. Similar algorithm for bichromatic case is presented in [20] and [22]. The algorithm is fairly straightforward, and we are going to describe it briefly in the following subsection.

In this thesis, 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. This result is published in [43].

1.1.3 A simple solution

To familiarize the reader with the problem, we present a simple but less efficient solution to it. This solution is given in [5].

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.

1.2 Structure of bottleneck matching

Our goal is to show the existence of a bottleneck matching that has a certain structure, which we then utilize to develop an efficient algorithm for finding such a matching. 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 $\mathscr{P}$ are called *edges*, and all other pairs are called *diagonals*. Each matching is, thus, comprised of edges and diagonals.

edges
diagonals

On several occasions it will be useful to discern between the two possible orientations of a diagonal. Although in most cases 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.

turning angle

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.

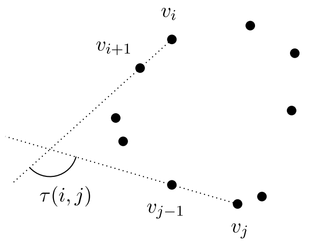
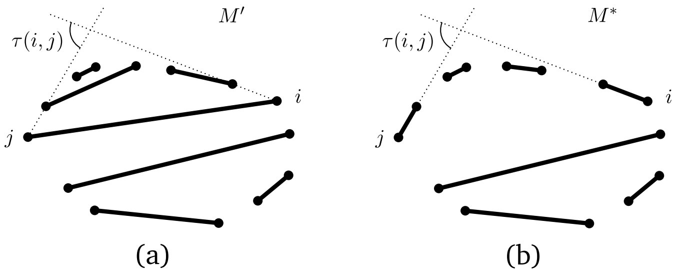


Figure 1. Turning angle.

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

Figure 2. (a) M', matching before the transformation. (b) M^*, matching after the transformation.

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. $\square$

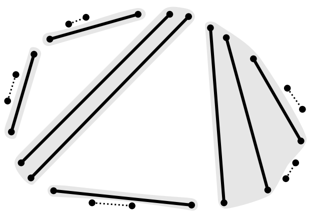


Figure 3. Diagonals inside each shaded area make a single cascade. There are three cascades with only one diagonal, one cascade with two diagonals, and one cascade with three diagonals.

k-bounded region

cascade

Let us consider the division of the interior of the polygon $\mathscr{P}$ into regions obtained by cutting it along diagonals of the given matching M. Each region created by this division is bounded by some diagonals of M and by the boundary of $\mathscr{P}$. 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 for an example.

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

Proof. Let M be a matching provided by Lemma 1, with all diagonals having their turning angles 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.

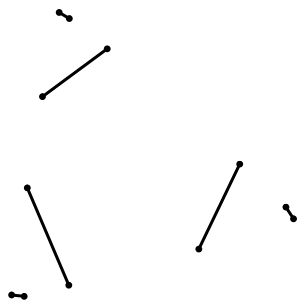


Figure 4. Configuration of points for which the only bottleneck matching has exactly three cascades.

It is not possible for a matching to have exactly two cascades. If there were exactly two cascades, there would be a region defined by diagonals from both of these cascades. If that region is bounded by exactly one diagonal from each cascade, it would then be 2-bounded and, by definition of cascade, we would have a single cascade. Otherwise, if that region is bounded by more than one diagonal from one of the two cascades, it would then be at least 3-bounded and, by definition of cascade, no two of its diagonals would belong to the same cascade, and hence we would have more than 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.

1.3 Finding bottleneck matchings

1.3.1 Matchings with at most one cascade

optimal matching

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),

$$S[i,j]=\min\begin{cases}\max\{S[i+1,j-1],|v_iv_j|\} & \text{(1.1a)}\\ \max\{S[i+2,j],|v_iv_{i+1}|\} & \text{(1.1b)}\\ \max\{S[i,j-2],|v_{j-1}v_j|\}. & \text{(1.1c)}\end{cases}$$

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.

necessary pair

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 an oriented pair *necessary*. This can be easily incorporated into the calculation of $S[i,j]$. Namely, if case (1.1a) is the only one achieving minimum among cases (1.1a), (1.1b) and (1.1c), we set $necessary(i,j)$ to $\top$, otherwise we set it to $\bot$. Note that $necessary(i,j)$ does not imply $necessary(j,i)$.

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.

1.3.2 Matchings with three cascades

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.

3-*cascade matchings*

Any three distinct points i, j and k with $j \in \{i+1, \ldots, k-1\}$, 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 $M_{k+1,i-1}$. These three feasible pairs are not necessarily diagonals in the combined matching, since they might not be necessary pairs in their respective 1-cascade matchings. To find the optimal matching 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, by *inner diagonals* we denote three oriented diagonals bounding the single 3-bounded region from their left side. More precisely, an oriented diagonal (i, j) is an inner diagonal if it defines the boundary of the 3-bounded region and that region lies on the opposite side of the diagonal from points in $\{i+1, \ldots, j-1\}$. So, if an oriented pair (i, j) is an inner diagonal, then the oriented pair (j, i) is not.

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. □

candidate diagonal

We say that oriented diagonal (i, j) is a *candidate diagonal*, if $necessary(i, j)$ 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. □

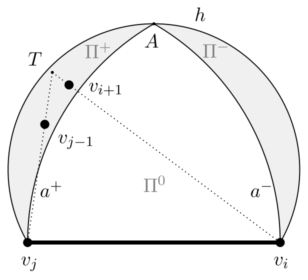


Figure 5. Points $v_{i+1}, \ldots, v_{j-1}$ all lie inside either Π^- or Π^+.

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.

Lemma 5. *If (i, j) is a candidate diagonal, then points $v_{i+1}, \dots, v_{j-1}$ 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 $\{i, \dots, j\}$ must lie inside the triangle $\triangle v_i T v_j$, so there cannot be two points from $\{i + 1, \dots, j-1\}$ 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 $\{i + 1, \dots, j - 1\}$ in Π^-.

It remains to be proved that none of the points in $\{i+1, \dots, j-1\}$ 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 $\{i, \dots, j\}$ such that $v_k \in \Pi^+$. Since (i, j) is a feasible pair, $\{i, \dots, j\}$ 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, we make a matching using pairs $(i, i + 1), (i + 2, i + 3), \dots, (j - 1, j)$. In the case the number is even, see Figure 7, we make a matching using pairs $(i, i + 1), (i + 2, i + 3), \dots, (k - 3, k - 2)$, pair $(k - 1, j)$, and pairs $(k, k + 1), (k + 2, k + 3), \dots, (j - 2, j - 1)$.

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 $|v_i v_j|$, so all matched pairs have distance not larger than $|v_i v_j|$. So, in each of the cases we constructed a matching $M_{i,j}$ of all points in $\{i, \dots, j\}$ which does not contain (i, j), with $bn(M_{i,j}) \leq |v_i v_j|$. 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

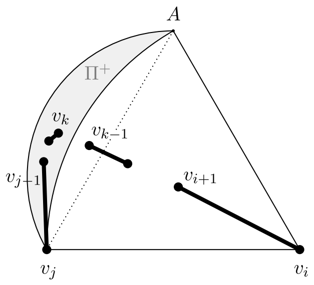


Figure 6. Matching points in $\{i, \dots, j\}$, the even case.

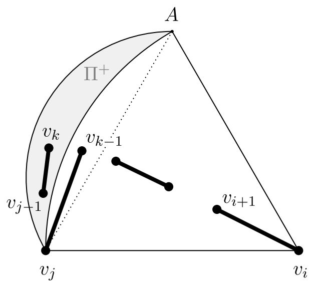


Figure 7. Matching points in $\{i, \dots, j\}$, the odd case.

with the assumption that there is a point from $\{i+1,\dots,j-1\}$ 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).

polarity
negative polarity
pole
positive polarity

Two possibilities for a candidate diagonal (i,j) provided by Lemma 5 bring forth a concept of *polarity*. If points $\{i+1,\dots,j-1\}$ lie in $\Pi^-(i,j)$ we say that candidate diagonal (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.

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

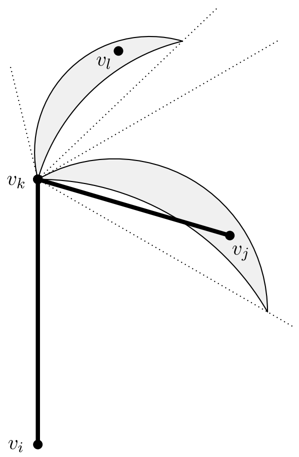


Figure 8. Two candidate diagonals of equal polarity having the same pole.

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 $j \in \{i+1,\dots,k-1\}$, see Figure 8.

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 $l \in \{j+1,\dots,k-1\}$. Points v_j and v_l lie in $\Pi^+(i,k)$ and $\Pi^+(j,k)$, respectively, meaning that $\pi/3 \le \angle v_i v_k v_j, \angle v_j v_k v_l \le 2\pi/3$, implying $2\pi/3 \le \angle v_i v_k v_j + \angle v_j v_k v_l = \angle v_i v_k v_l \le 4\pi/3$. This means that v_l does not belong to $\Pi^+(i,j)$. However, that is not possible, because $l \in \{i+1,\dots,j-1\}$ 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 1.3.2, to construct Algorithm 1.

Algorithm 1 Bottleneck Matching

```
Calculate $S[i,j]$ and $necessary(i,j)$ for all feasible $(i,j)$ pairs,
as described in Section 1.3.1.
$best \leftarrow \min\{S[i+1,i] : i \in \{0,\dots,n-1\}\}$
**for** all feasible $(i,j)$ **do**
    **if** $necessary(i,j)$ and $\tau(i,j) \leq 2\pi/3$ **then**
        **for** $k \in \{j+1,\dots,i-1\}$ such that $(j+1,k)$ is feasible **do**
            $best \leftarrow \min\{best, \max\{S[i,j], S[j+1,k], S[k+1,i-1]\}\}$
```

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 1.3.1. 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 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 $\{j+1,\dots,i-1\}$ in two parts, $\{j+1,\dots,k\}$ and $\{k+1,\dots,i-1\}$, which together with $\{i,\dots,j\}$ 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.

Chapter 2

Bichromatic bottleneck matchings

2.1 Introduction

2.1.1 Problem statement

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. Denote the length of a longest line segment in M with $\text{bn}(M)$, which we also call the *value* of M. We aim to find a matching under given constraints that minimizes $\text{bn}(M)$. Any such matching is called a *bottleneck matching* of P.

value, bn

bottleneck matching

2.1.2 Our results

We develop tools which enable us to solve the problem of finding a bottleneck bichromatic non-crossing matching of points in convex position in $O(n^2)$ time, improving upon previously best-known algorithm of $O(n^3)$-time complexity, given in [20] and [22]. Also, combining the same toolset with a geometric analysis we design an optimal $O(n)$ algorithm for the same problem in case when the points lie on a circle, where previously best-known algorithm has $O(n \log n)$-time complexity. The manuscript containing our results can be found in [44].

In order to efficiently deal with bichromatic non-crossing matchings on convex point sets, we introduce a structure that we refer to as *orbits*, which turn out to capture well some of the structural properties of such matchings. Namely, the points naturally partition into sets, i.e. orbits, in such a way that two differently colored points can be connected by a segment in a non-crossing perfect matching if and only if they belong to the same orbit.

There is a number of additional properties of orbits that we can put to good use, and once we combine them with the ideas used to efficiently solve the monochromatic case in [43], we are able to construct efficient algorithms in the bichromatic version of the problem, both for the convex case and for the case where all points lie on a circle. We note that the theory behind orbits may be of independent interest when tackling related problems.

The exposition of these results is organized as follows. In Section 2.3 we formally define orbits and derive numerous properties that hold for them. We note the existance of a structured relationship between orbits. This leads us to the definition of orbit graphs for which we also show certain properties. In Section 2.4 we make use of the theory developed around orbits to construct an efficient algorithm for finding a bottleneck matching of points in convex position. In Section 2.5 we again use properties of orbits and orbit graph to optimally solve the problem of finding a bottleneck matching for points on a circle.

2.2 Preliminaries

As we deal with perfect matchings without crossings, from now on, when we talk about matchings, it is understood that we refer to matchings that are both perfect and crossing-free.

Also, we assume that the given points in P are in convex position, i.e. they are the vertices of a convex polygon $\mathcal{P}$. Let us label the points of P by $v_0, v_1, \ldots, v_{2n-1}$ in positive (counterclockwise) direction. To simplify the notation, we will often use only indices when referring to points. 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 $\{j, \ldots, i\}$.

balanced, blue-heavy, red-heavy

Definition 9. A set of points is *balanced* if it contains the same number of red and blue points. If the set has more red points than blue, we say that it is *red-heavy*, and if there are more blue points than red, we call it *blue-heavy*.

As we already mentioned, we assume that P consists of n red and n blue points, i.e. it is balanced.

The following lemma gives us a simple but important tool that ensures the existence of a matching on a point set.

Lemma 10. *Every balanced set of points can be matched.*

Proof. We denote the set of points by Q, and let $v \in Q$. W.l.o.g., assume v is red. We scan all other points by angle around v, starting from one neighbor of v on the convex hull and ending in the other. We keep track of the difference between the number of blue and red points encountered so far. At the beginning, this difference is 0, and at the end it is 1, since there is one more blue point in $Q \setminus \{v\}$. As the difference changes by one at each point, it must go from 0 to 1 at some blue point

u. We match v with u, and we split the point set into two balanced parts, one on each side of the line uv, continuing this process recursively for both parts, until we match all the points. □

feasible pair

Definition 11. We say that (i, j) is a *feasible pair* if there exists a matching containing (i, j).

We will make good use of the following characterization of feasible pairs.

Lemma 12. *A pair (i, j) is feasible if and only if i and j have different colors and $\{i, \dots, j\}$ is balanced.*

Proof. If (i, j) is feasible, then i and j have different colors. Also, there is a matching that contains the pair (i, j), and at the same time the set $\{i+1, \dots, j-1\}$, containing all points on one side of the line ij, is matched. Then $\{i+1, \dots, j-1\}$ must be balanced, so $\{i, \dots, j\}$ is balanced as well.

On the other hand, if i and j are of different colors and $\{i, \dots, j\}$ is balanced, then both $\{i+1, \dots, j-1\}$ and $\{j+1, \dots, i-1\}$ are also balanced. Thus we can match i with j, and Lemma 10 ensures that each of the sets $\{i+1, \dots, j-1\}$ and $\{j+1, \dots, i-1\}$ can be matched. Clearly, the obtained matching remains crossing free. □

The statement of Lemma 12 is quite simple, and we will apply it on many occasions. To avoid its numerous mentions that could make some of our proofs unnecessarily cumbersome, from now on we will use it without explicitly stating it.

2.3 Orbits and their properties

functions o and o^{-1}

Definition 13. By $o(i)$ we denote the first point starting from i in the positive direction such that $(i, o(i))$ is feasible. By $o^{-1}(i)$ we denote the first point starting from i in the negative direction such that $(o^{-1}(i), i)$ is feasible.

As we assume that the given point set is balanced, Lemma 10 guarantees that both o and o^{-1} are well-defined. Let us also point out that the chosen notation is appropriate, as we will later show, as a part of Property 16, that o^{-1} is the inverse function of o.

Property 14. *If a set $\{i, \dots, j\}$ is such that the number of points in $\{i, \dots, j\}$ of the same color as i is not larger than the number of points of the other color, then $o(i) \in \{i+1, \dots, j\}$.*

If a set $\{i, \dots, j\}$ is such that the number of points in $\{i, \dots, j\}$ of the same color as j is not larger than the number of points of the other color, then $o^{-1}(j) \in \{i, \dots, j-1\}$.

Proof. W.l.o.g. assume that i is red. We observe the difference between the number of red points and the number blue points in $\{i, \dots, k\}$, as k goes from i to j. In the beginning, when $k = i$, this difference is 1, and at the end, when $k = j$ the difference is at most 0. In each step this difference changes by 1, so the first time this difference is 0, the point k must be blue. This is the first time the set $\{i, \dots, k\}$ is balanced, so $o(i) = k \in \{i+1, \dots, j\}$.

The second part of the property is proven in an analogous manner. □

A straightforward consequence of Property 14 follows.

Property 15. *If $\{i, \dots, j\}$ is balanced, then $o(i) \in \{i+1, \dots, j\}$ and $o^{-1}(j) \in \{i, \dots, j-1\}$.*

Proof. Since $\{i, \dots, j\}$ is balanced, conditions from Property 14 hold both for i and j, and hence, $o(i) \in \{i+1, \dots, j\}$ and $o^{-1}(j) \in \{i, \dots, j-1\}$. □

The next property establishes the connection of o and o^{-1} which has already been informally suggested by the notation.

Property 16. *Function o is a bijection, and o^{-1} is its inverse function.*

Proof. To show that the function o is bijective and o^{-1} is its inverse, it is enough to prove that, for all i, we have $o(o^{-1}(i)) = i$ and $o^{-1}(o(i)) = i$.

Let $j = o(i)$ and $k = o^{-1}(j)$. Suppose that $i \neq k$. By definition of o, the set $\{i, \dots, j\}$ is balanced, so by Property 15 we have that $k \in \{i, \dots, j-1\}$. On the other hand, by definition of o^{-1}, the set $\{k, \dots, j\}$ is also balanced, so $\{i, \dots, k-1\}$ must be balanced as well. But this means, again by Property 15, that $o(i) \in \{i+1, \dots, k-1\}$, which is a contradiction. Hence, $o^{-1}(o(i)) = i$. The claim that $o(o^{-1}(i)) = i$ is proven analogously. □

Now we are ready to define orbits.

orbit

Definition 17. An *orbit* of i, denoted by $\mathcal{O}(i)$, is defined by $\mathcal{O}(i) := \{o^k(i) : k \in \mathbb{Z}\}$. By $\mathcal{O}(P)$ we denote the set of all orbits of a convex point set P, that is $\mathcal{O}(P) := \{\mathcal{O}(i) : i \in P\}$.

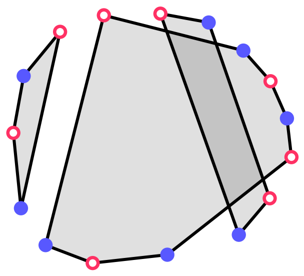


Figure 9. Orbits – an example.

An example of a balanced 2-colored convex point set along with its set of orbits can be found in Figure 9. Note that from the definition of orbits it is clear that for each $j \in \mathcal{O}(i)$ we have $\mathcal{O}(j) = \mathcal{O}(i)$, and therefore the set of all orbits, $\mathcal{O}(P)$, is a *partition* of the set of all points.

The number of orbits can be anything from 1, when colors alternate, as in Figure 10(a), to $n/2$, when points in each color group are consecutive, as in Figure 10(b).

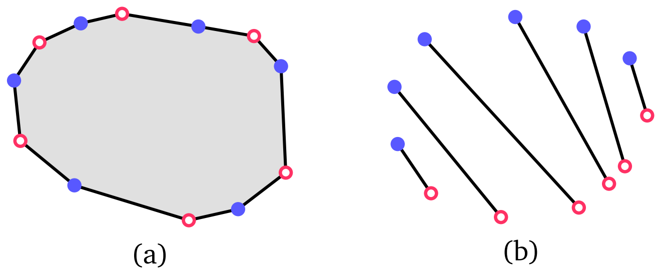


Figure 10.
(a) One orbit of size $2n$.
(b) n orbits of size 2.

Next, we prove a number of properties of orbits.

The first property provides a simple characterization of a feasible pair via orbits, which is essential for our further application of orbits.

Property 18. *Points i and j form a feasible pair if and only if they have different colors and $\mathcal{O}(i) = \mathcal{O}(j)$.*

Proof. First, suppose that i and j have different colors and belong to the same orbit. Then $j = o^s(i)$, where s is odd (as i and j have different colors). For each $r \in \{0, \dots, s-1\}$, the pair $(o^r(i), o^{r+1}(i))$ is feasible so $\{o^r(i), \dots, o^{r+1}(i)\}$ is balanced. This, together with the fact that the sequence $o^0(i), o^1(i), \dots, o^s(i)$ alternates between red and blue points, implies that $\{i, \dots, j\}$ is balanced as well, that is, the pair (i, j) is feasible.

Next, let (i, j) be a feasible pair, where, say, i is red and j is blue. Suppose for a contradiction that i and j belong to different orbits. Let r be such that $j \in \{o^r(i)+1, \dots, o^{r+1}(i)-1\}$, see Figure 11. W.l.o.g. suppose that $o^r(i)$ is blue (the other case is symmetrical with respect to the direction around P). Both $(i, o^r(i))$ and (i, j) are feasible pairs, from which it follows that $\{o^r(i)+1, \dots, j\}$ is balanced. The points $o^r(i)$ and j are of the same color, so $\{o^r(i), \dots, j-1\}$ is also balanced. However, Property 15 implies that $o^{r+1}(i) = o(o^r(i)) \in \{o^r(i)+1, \dots, j-1\}$, which is a contradiction with the choice of r. $\square$

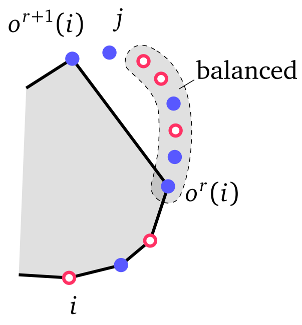


Figure 11. Illustrating the proof of Property 18

The following property discusses the way a feasible pair divides an orbit, whether it belongs to it or not.

Property 19. *A feasible pair divides points of* any *orbit into two balanced parts.*

Proof. Let (i, j) be a feasible pair and let $\mathscr{A}$ be an orbit. By Property 18 points can be matched only within their orbit, so if $\{i, \dots, j\} \cap \mathscr{A}$ is not balanced, then it is not possible to complete a matching containing (i, j) which is a contradiction with (i, j) being feasible. $\square$

Informally speaking, the following property ensures that by repeatedly applying function o, we follow the points of an orbit as they appear on $\mathscr{P}$, thus visiting *all* the points of the orbit in a *single* turn around the polygon.

Property 20. *No point of an orbit $\mathscr{O}(i)$ lies between i and $o(i)$, that is, $\{i, \dots, o(i)\} \cap \mathscr{O}(i) = \{i, o(i)\}$.*

Proof. Suppose there is a point $j \in \mathscr{O}(i)$ such that $j \in \{i, \dots, o(i)\} \setminus \{i, o(i)\}$. The colors of i and $o(i)$ are different, so the color of j is either different from i or from $o(i)$.

If i and j have different colors, knowing that they belong to the same orbit, by Property 18 the pair (i, j) is feasible, which is a contradiction with $o(i)$ being the first point from i in the positive direction such that $(i, o(i))$ is feasible.

The other case, when $o(i)$ and j have different colors, is treated analogously. $\square$

The following two properties are simple consequences of the previous statement.

Property 21. *Any two neighboring points in an orbit have different colors.*

Proof. From Property 20 we have that if i and j are neighboring points on an orbit, then either $j = o(i)$ or $i = o(j)$. By the definition of the function o, this means that i and j have different colors. □

Property 22. *Every orbit is balanced.*

Proof. This follows directly from Property 21. □

Next, we discuss a structural property of two different orbits.

Property 23. *Let i and j be points from two different orbits such that there are no other points from their orbits between them, that is, $\{i, \dots, j\} \cap \mathcal{O}(i) = i$ and $\{i, \dots, j\} \cap \mathcal{O}(j) = j$. Then, i and j have the same color.*

Proof. Suppose for a contradiction that i and j have different colors, say, i is blue and j is red. Since they are not from the same orbit, by Property 18 the pair (i, j) is not feasible. Thus, $\{i, \dots, j\}$ is not balanced, so it is either red-heavy or blue-heavy.

If it is red-heavy, then by Property 14 we have $o(i) \in \{i + 1, \dots, j\}$, which is a contradiction with $\{i, \dots, j\} \cap \mathcal{O}(i) = i$.

If $\{i, \dots, j\}$ is blue-heavy, then, again by Property 14, we have that $o^{-1}(j) \in \{i, \dots, j - 1\}$, which is a contradiction with $\{i, \dots, j\} \cap \mathcal{O}(j) = j$. □

Moving on to the algorithmic part of the story, we show that we can efficiently compute all the orbits, or more precisely – all the values of the function o.

Lemma 24. *The function $o(i)$, for all i, can be computed in $O(n)$ time.*

Proof. The goal is to find $o(i)$ for each $i \in \{0,\dots,2n-1\}$. We start by showing how to find i_0 such that for every $j \in \{0,\dots,2n-1\}$, we have that $\{i_0,\dots,j\}$ is either balanced or red-heavy.

We define z_i to be the number of red points minus the number of blue points in $\{0,\dots,i-1\}$. All these values can be calculated in $O(n)$ time, since $z_i = z_{i-1} \pm 1$, where we take the plus sign if the point $i-1$ is red, and the minus sign if it is blue. If for i_0 we take i for which z_i is minimum, breaking ties arbitrarily, it is straightforward to check that the above condition is satisfied.

Now, to calculate the function o in all the red points, we run the following algorithm.

```
Find $i_0$ as described.
Create new empty stack $\mathscr{S}$.
**for** $i \in \{i_0,\dots,i_0-1\}$ **do**
    **if** $i \in R$ **then**
        $\mathscr{S}.Push(i)$
    **else**
        $j \leftarrow \mathscr{S}.Pop()$
        $o(j) \leftarrow i$
```

The way i_0 is chosen guarantees that for every $j \in \{0,\dots,2n-1\}$, the number of blue points in the set $\{i_0,\dots,j\}$ is at most the number of red points in the same set, i.e. the set is either balanced or red-heavy. This ensures that the stack will never be empty when Pop operation is called. When $o(j)$ is assigned, the point j is the last on the stack because each red point that came after j is popped when its blue pair is encountered, meaning that $\{j,\dots,i\}$ is balanced. Moreover, this is the first time such a situation happens, so the assignment $o(j) = i$ is correct.

By running this algorithm we calculated the function o in all red points. To calculate it in blue points as well, we run an analogous algorithm where the color roles are swapped.

All the parts of this process run in $O(n)$ time, so the function o

and, thereby, all orbits, are calculated in $O(n)$ time as well. □

We define two categories of feasible pairs according to the relative position within their orbit.

edge, diagonal

Definition 25. We call a feasible pair (i, j) an *edge* if and only if $i = o(j)$ or $j = o(i)$; otherwise, it is called a *diagonal*.

In other words, pairs consisting of two neighboring vertices of an orbit are called edges, and all other feasible pairs are called diagonals. Note that edges are not necessarily neighboring vertices in P.

Property 26. *If (i, j) is an edge such that $|\{i, \dots, j\}| > 2$, then the pair $(i+1, j-1)$ is feasible.*

Proof. The pair (i, j) is feasible, so $\{i, \dots, j\}$ is balanced. Points i and $i+1$ have the same color, otherwise $(i, i+1)$ would be an edge, and similarly, points j and $j-1$ have the same color. Hence, points $i+1$ and $j-1$ have different colors, which, together with the fact that $\{i+1, \dots, j-1\}$ is balanced as well, implies that the pair $(i+1, j-1)$ is feasible. □

Property 27. *If $\{i, \dots, j\}$ is balanced, then points in $\{i, \dots, j\}$ can be matched using edges only.*

Proof. We prove this by induction on the size of $\{i, \dots, j\}$. The statement obviously hold for the base case, where $j = i+1$, since $(i, i+1)$ itself must be an edge.

Let us assume that the statement is true for all balanced sequences of points of size less than r, and let $|\{i, \dots, j\}| = r$. Property 15 implies that $o(i) \in \{i, \dots, j\}$. We construct a matching on $\{i, \dots, j\}$ by taking the edge $(i, o(i))$ together with edge-only matchings on $\{i+1, \dots, o(i)-1\}$ and $\{o(i)+1, \dots, j\}$, which are provided by the induction hypothesis. □

When speaking about edges, we consider them to be ordered pairs of points, so that the edge $(i, o(i))$ is considered to be directed from i to $o(i)$. We say that points $\{i, \dots, o(i)\} \setminus \{i, o(i)\}$ lie on the right side of the edge $(i, o(i))$, and points $\{o(i), \dots, i\} \setminus \{i, o(i)\}$ lie on its left side. Directionality of edges and coloring of points together bring forth the two possible types of edges, as the following definition states.

Definition 28. We say that $(i, o(i))$ is a *red-blue* edge if $i \in R$, and *blue-red* edge if $i \in B$.

red-blue edge, blue-red edge

Note that sometimes an orbit comprises only two points, in case when $o(o(i)) = i$; we think of it as if it has two edges, $(i, o(i))$ and $(o(i), i)$, one being red-blue and the other being blue-red.

Property 29. *Two edges of the same type (both red-blue, or both blue-red) from different orbits do not cross.*

Proof. Let $(i, o(i))$ and $(j, o(j))$ be two edges of the same type, and $\mathcal{O}(i) \neq \mathcal{O}(j)$. Suppose, for a contradiction, that these edges cross, then we either have $j \in \{i, \ldots, o(i)\}$ or $i \in \{j, \ldots, o(j)\}$.

W.l.o.g. we can assume that $j \in \{i, \ldots, o(i)\}$. Then, there are no points from $\mathcal{O}(i) \cup \mathcal{O}(j)$ in $\{j, \ldots, o(i)\} \setminus \{j, o(i)\}$, and Property 23 implies that points $o(i)$ and j have the same color. However, this is a contradiction with the assumption that $(i, o(i))$ and $(j, o(j))$ are of the same type. $\square$

Property 30. *For every two orbits $\mathscr{A}, \mathscr{B} \in \mathcal{O}(P)$, $\mathscr{A} \neq \mathscr{B}$, either all points of $\mathscr{B}$ are on the right side of red-blue edges of $\mathscr{A}$, or all points of $\mathscr{B}$ are on the right side of blue-red edges of $\mathscr{A}$.*

Proof. Suppose for a contradiction that there are two points from $\mathscr{B}$, one on the right side of a red-blue edge of $\mathscr{A}$, and the other on the right side of a blue-red edge of $\mathscr{A}$, see Figure 12. Let i and j be two such points with no other points from $\mathscr{B}$ in $\{i, \ldots, j\}$ (we can always find such a pair, since each point of $\mathscr{B}$ is either behind a red-blue edge, or behind a blue-red edge of $\mathscr{A}$). Then, (i, j) is an edge of $\mathscr{B}$ which crosses both a red-blue edge and a blue-red edge of $\mathscr{A}$, which is a contradiction with Property 29. $\square$

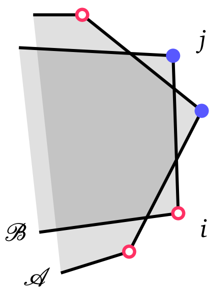


Figure 12. Illustrating the proof of Property 30

The following property tells us about how the orbits are mutually synchronized.

Property 31. *Let $\mathscr{A}, \mathscr{B} \in \mathscr{O}(P)$. There are no points of $\mathscr{B}$ on the right side of red-blue edges of $\mathscr{A}$ if and only if there are no points of $\mathscr{A}$ on the right side of blue-red edges of $\mathscr{B}$.*

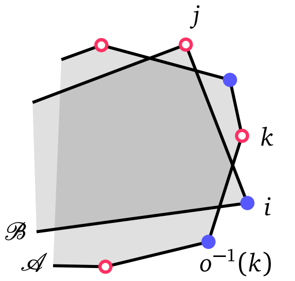


Figure 13. Illustrating the proof of Property 31

Proof. If $\mathscr{A} = \mathscr{B}$ this is trivially true.

Let there be no points of $\mathscr{B}$ on the right side of red-blue edges of $\mathscr{A}$. Suppose for a contradiction that there is a blue-red edge (i, j) of $\mathscr{B}$ such that there are points of $\mathscr{A}$ on its right side, see Figure 13. Let k be the first point from $\mathscr{A}$ in $\{i, \dots, j\}$. It must be red, otherwise point i of $\mathscr{B}$ would be on the right side of the red-blue edge $(o^{-1}(k), k)$ of $\mathscr{A}$. But now, $i \in \mathscr{B}$ is blue and $k \in \mathscr{A}$ is red, and no points of $\mathscr{A} \cup \mathscr{B}$ are in $\{i, \dots, k\}$ other than i and k, which is a contradiction with Property 23.

The other direction is proven analogously. □

relation $\preceq$

Definition 32. The relation $\preceq$ on the set of all orbits, $\mathscr{O}(P)$, is defined by setting $\mathscr{A} \preceq \mathscr{B}$ if and only if there are no points of $\mathscr{B}$ on the right sides of red-blue edges of $\mathscr{A}$ (which, by Property 31, is equivalent to no points of $\mathscr{A}$ being on the right sides of blue-red edges of $\mathscr{B}$).

Property 33. *The relation $\preceq$ on $\mathscr{O}(P)$ is a total order.*

Proof. For each $\mathscr{A}, \mathscr{B} \in \mathscr{O}(P)$, the following holds.

Totality. $\mathscr{A} \preceq \mathscr{B}$ or $\mathscr{B} \preceq \mathscr{A}$.

If $\mathscr{A} = \mathscr{B}$ this is trivially true. Suppose $\mathscr{A} \preceq \mathscr{B}$ does not hold. Because of Property 30, no points of $\mathscr{B}$ are on the right side of blue-red edges of $\mathscr{A}$, so $\mathscr{B} \preceq \mathscr{A}$, by the definition of the relation $\preceq$.

Antisymmetry. If $\mathscr{A} \preceq \mathscr{B}$ and $\mathscr{B} \preceq \mathscr{A}$, then $\mathscr{A} = \mathscr{B}$.

From $\mathscr{A} \preceq \mathscr{B}$ we know that no points of $\mathscr{A}$ are on the right side of blue-red edges of $\mathscr{B}$. But, since $\mathscr{B} \preceq \mathscr{A}$, there are no points of $\mathscr{A}$ on the right side of red-blue edges of $\mathscr{B}$, either. This is only possible if $\mathscr{A} = \mathscr{B}$.

Transitivity. If $\mathscr{A} \preceq \mathscr{B}$ and $\mathscr{B} \preceq \mathscr{C}$, then $\mathscr{A} \preceq \mathscr{C}$.

If $\mathscr{A} \preceq \mathscr{B}$ then all red-blue edges of $\mathscr{A}$ must lie on the right side of red-blue edges of $\mathscr{B}$, because no red-blue edges of $\mathscr{A}$ can cross a red-blue edge of $\mathscr{B}$ (Property 29) and there are no points of $\mathscr{A}$ on the right side of blue-red edges of $\mathscr{B}$. However, since $\mathscr{B} \preceq \mathscr{C}$, there are no points of $\mathscr{C}$ right of red-blue edges of $\mathscr{B}$, so no point of $\mathscr{C}$ can be on the right side of some red-blue edge of $\mathscr{A}$. Hence, $\mathscr{A} \preceq \mathscr{C}$. □

Property 34. *Let $\mathscr{A}$ and $\mathscr{B}$, $\mathscr{A} \preceq \mathscr{B}$, be two consecutive orbits in the total order of orbits, that is, there is no $\mathscr{L}$ different from $\mathscr{A}$ and $\mathscr{B}$, such that $\mathscr{A} \preceq \mathscr{L} \preceq \mathscr{B}$. If i and j are two points, one from $\mathscr{A}$ and the other from $\mathscr{B}$ such that there are no points from $\mathscr{A}$ or $\mathscr{B}$ in $\{i,\dots,j\}$ other than i and j, then i and j are two consecutive points on $\mathscr{P}$.*

The inverse also holds, for any two consecutive points i and $i+1$ in P which belong to different orbits, orbits $\mathscr{O}(i)$ and $\mathscr{O}(i+1)$ are two consecutive orbits in the total order of orbits.

Note that Property 21 and Property 23 ensure that two consecutive points in P belong to different orbits if and only if they have the same color.

Proof. (of Property 34) Assume that $i \in \mathscr{A}$ and $j \in \mathscr{B}$ are two points such that $\mathscr{A} \cap \{i,\dots,j\} = \{i\}$ and $\mathscr{B} \cap \{i,\dots,j\} = \{j\}$, see Figure 14. (The case when $i \in \mathscr{B}$ and $j \in \mathscr{A}$ is proven analogously.)

Points i and j must have the same color, by Property 23. Since j is on the right side of the edge $(i, o(i))$ and $\mathscr{A} \preceq \mathscr{B}$, that edge must be blue-red, so both i and j are blue.

Suppose that there is an orbit $\mathscr{L}$ with points in $\{i+1,\dots,j-1\}$. But then, those points are on the right side of the blue-red edge $(i, o(i))$ and on the right side of the red-blue edge $(o^{-1}(j), j)$, that is, $\mathscr{A} \preceq \mathscr{L}$ and $\mathscr{L} \preceq \mathscr{B}$, a contradiction.

To show the inverse statement, assume that points i and $i+1$ belong to different orbits. W.l.o.g., assume $\mathscr{O}(i) \preceq \mathscr{O}(i+1)$. If there is an orbit $\mathscr{L}$ different from both $\mathscr{O}(i)$ and $\mathscr{O}(i+1)$, such that $\mathscr{O}(i) \preceq \mathscr{L} \preceq \mathscr{O}(i+1)$, then i would lie on the right side of red-blue edges of $\mathscr{L}$, and no points of $\mathscr{O}(i+1)$ would lie on the right side of red-blue edges of $\mathscr{L}$. But, this is not possible since position of points i and $i+1$ must be the same relative to any edge containing neither i nor $i+1$. □

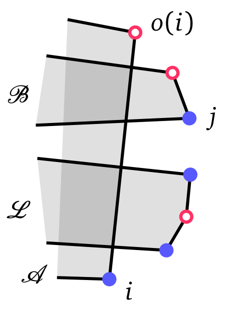


Figure 14. Illustrating the proof of Property 34

2.3.1 Orbit graphs

Orbit graph

Definition 35. *Orbit graph* $\mathscr{G}(P)$ is a directed graph whose vertex set is the set of all orbits, $\mathscr{O}(P)$, and there is an arc from an orbit $\mathscr{A}$ to an orbit $\mathscr{B}$ if and only if $\mathscr{A}$ and $\mathscr{B}$ cross and $\mathscr{A} \preceq \mathscr{B}$.

Property 36. *Let $\mathscr{A}, \mathscr{B}, \mathscr{C} \in \mathscr{O}(P)$. If both $(\mathscr{A}, \mathscr{B})$ and $(\mathscr{A}, \mathscr{C})$ are arcs of $\mathscr{G}(P)$, or both $(\mathscr{B}, \mathscr{A})$ and $(\mathscr{C}, \mathscr{A})$ are arcs of $\mathscr{G}(P)$, then either $(\mathscr{B}, \mathscr{C})$ or $(\mathscr{C}, \mathscr{B})$ is an arc of $\mathscr{G}(P)$ as well.*

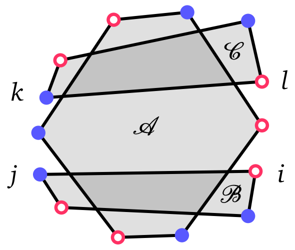


Figure 15. Illustrating the proof of Property 36

Proof. Assume that in $\mathscr{G}(P)$ there is an arc between $\mathscr{A}$ and $\mathscr{B}$, an arc between $\mathscr{A}$ and $\mathscr{C}$, but no arc between $\mathscr{B}$ and $\mathscr{C}$. By definition, $\mathscr{A}$ crosses both $\mathscr{B}$ and $\mathscr{C}$, and $\mathscr{B}$ and $\mathscr{C}$ do not cross, as illustrated in Figure 15. Then, there is an edge (i, j) of $\mathscr{B}$ such that the whole $\mathscr{C}$ lies on its right side, and there is an edge (k, l) of $\mathscr{C}$ such that the whole $\mathscr{B}$ lies on its right side.

From Property 23 we know that points i and l must be of the same color. Therefore, edges (i, j) and (k, l) are of different types. Orbit $\mathscr{A}$ crosses both $\mathscr{B}$ and $\mathscr{C}$, so it must cross both (i, j) and (k, l). If $\mathscr{A} \preceq \mathscr{B}$ then (i, j) must be red-blue, since there are points of $\mathscr{A}$ on the right side of (i, j), and thus (k, l) must be blue-red. But there are also points of $\mathscr{A}$ on the right side of (k, l), so $\mathscr{C} \preceq \mathscr{A}$. Analogously, If $\mathscr{B} \preceq \mathscr{A}$, then $\mathscr{A} \preceq \mathscr{C}$.

Hence, if both $\mathscr{A} \preceq \mathscr{B}$ and $\mathscr{A} \preceq \mathscr{C}$ or both $\mathscr{B} \preceq \mathscr{A}$ and $\mathscr{C} \preceq \mathscr{A}$, then $\mathscr{B}$ and $\mathscr{C}$ must cross. □

Property 37. *Each weakly connected component of $\mathscr{G}(P)$ contains a unique Hamiltonian path.*

Proof. Assume there is a weakly connected component of $\mathscr{G}(P)$ without a Hamiltonian path. Let $L = \mathscr{L}_0, \mathscr{L}_1, \ldots, \mathscr{L}_m$ be the longest path in that component.

Firstly, let us suppose that there is an orbit $\mathscr{A} \notin L$ and an arc from $\mathscr{A}$ to $\mathscr{L}_i$, for some i. Let i_0 be the smallest such index. It must be that $i_0 > 0$, otherwise the path $\mathscr{A}, \mathscr{L}_0, \mathscr{L}_1, \ldots, \mathscr{L}_m$ would be longer than L. From Property 36 it follows that there is an arc between $\mathscr{A}$ and $\mathscr{L}_{i_0-1}$, but it cannot be from $\mathscr{L}_{i_0-1}$ to $\mathscr{A}$ because of the way we chose i_0. Therefore, there is an arc $(\mathscr{L}_{i_0-1}, \mathscr{A})$, and the path $\mathscr{L}_0, \dots \mathscr{L}_{i_0-1}, \mathscr{A}, \mathscr{L}_{i_0}, \dots \mathscr{L}_m$ is longer than L, which is a contradiction.

If there is no such i, meaning that there is no arc going from an orbit not in L to an orbit in L, then, since the component is weakly connected, there must be an arc going from an orbit in L to an orbit not in L. We can now apply the exact same reasoning to the graph obtained by reversing all arcs of that component and choosing the same longest path, only reversed, to again arrive to a contradiction.

Finally, since the graph $\mathscr{G}(P)$ is a subgraph of a total order graph, there is at most one Hamiltonian path in a weakly connected component. □

Lemma 38. *The total order of orbits, and the Hamiltonian paths for all weakly connected components of the orbit graph can be found in $O(n)$ time in total.*

Proof. Our goal here is to compute $succ(\mathscr{A})$ and $succG(\mathscr{A})$ for each orbit $\mathscr{A}$, defined as the successor of $\mathscr{A}$ in the total order of orbits, and the successor of $\mathscr{A}$ in the corresponding Hamiltonian path, respectively. (Undefined values of these functions mean that there is no successor in the respective sequence.) Having these two functions calculated, it is then easy to reconstruct the total order and the Hamiltonian paths. We start by computing the orbits in $O(n)$ time, as described in Lemma 24.

From Property 34 it is obvious that for every two consecutive orbits in the total order, there are at least two consecutive points on P, one from each of those orbits. We scan through all consecutive pairs of points on P. Let i and $i+1$ be two consecutive points. If they have different color, then they belong to the same orbit and we do nothing in this case. If their color is the same, they belong to different orbits, and from Property 34 we know that those two orbits are consecutive in the total order. If the color of the points is blue then there is a point $i+1$ from $\mathscr{O}(i+1)$ on the right side of blue-red edge $(i, o(i))$ from $\mathscr{O}(i)$, so we conclude that $\mathscr{O}(i) \leq \mathscr{O}(i+1)$, and we set $succ(i) = i+1$. In the other case, when the points are red, we set $succ(i+1) = i$. It is only left to check whether these two orbits cross. If they cross anywhere, then edges $(i, o(i))$ and $(o^{-1}(i+1), i+1)$ must cross each other (otherwise, the whole $\mathcal{O}(i+1)$ would lie on the right side of $(i, o(i))$), so it is enough to check only for this pair of edges whether they cross. If they do cross, we do the same with the function $succG$, we either set $succG(i) = i+1$ if the points are blue, or $succG(i+1) = i$ if they are red. If they do not cross, we do not do anything.

Constructing the corresponding sequences of orbits is done by first finding the orbits which are not successor of any other orbit and then by just following the corresponding successor function.

The whole process takes $O(n)$ time in total. □

2.4 Finding bottleneck matchings

In order to solve the problem of finding a bottleneck bichromatic matching of points in convex position, we will make use of the theory that we developed for orbits and the orbit graph, combining it with the approach we used in [43] to tackle the monochromatic case.

For the special configuration where colors alternate, i.e. two points are colored the same if and only if the parity of their indices is the same, we note that every pair (i, j) where i and j are of different parity is feasible. This is also the case with the monochromatic version of the same problem, so since the set of pairs that is allowed to be matched is the same in both cases, the bichromatic problem is in a way a generalization of the monochromatic problem – to solve the monochromatic problem it is enough to color the points in an alternating fashion, and then apply the algorithm which solves the bichromatic problem.

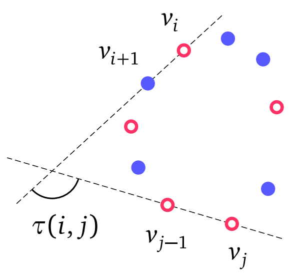


Figure 16. Turning angle.

oriented pairs and diagonals

We already said that edges are considered to be oriented. As far as arbitrary pairs are concerned, in most cases we do not need to worry about the order of i and j in the pair (i, j). Nevertheless, for the situations where this distinction between (i, j) and (j, i) is important, we will add qualifier *oriented* and speak about oriented pairs or oriented diagonals.

turning angle, τ

Definition 39. The *turning angle* of $\{i, \dots, j\}$, 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 16.

Lemma 40. *There is a bottleneck matching M of P such that all diagonals $(i,j) \in M$ have $\tau(i,j) > \pi/2$.*

Proof.

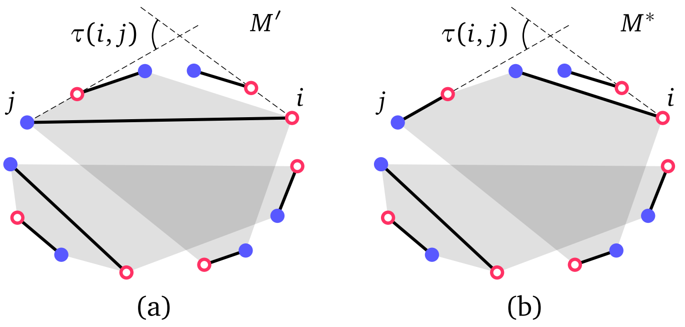


Figure 17.
(a) Matching before the transformation.
(b) Matching after the transformation.

Let us suppose that there is no such matching. Let M' be a bottleneck matching with the least number of diagonals. By the assumption, there is a diagonal $(i,j) \in M'$ such that $\tau(i,j) \le \pi/2$, see Figure 17(a). By Property 27 we can replace all pairs from M' lying in $\{i,\dots,j\}$, including the diagonal (i,j), with the matching containing only edges, and by doing so we obtain a new matching M^*, see Figure 17(b).

The longest distance between any pair of points from $\{i,\dots,j\}$ is achieved by the pair (i,j), hence $bm(M^*) \le bm(M')$. Since M' is a bottleneck matching, M^* is a bottleneck matching as well, and M^* has at least one diagonal less than M', a contradiction. □

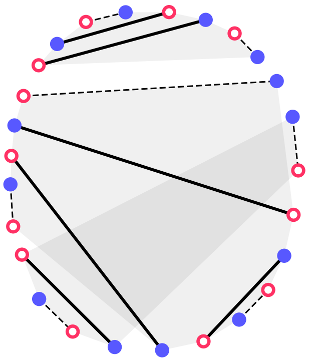


Figure 18. Matching consisting of edges (dashed lines) and diagonals (solid lines). Orbits are denoted by gray shading. There are three cascades in this example: one consist of the three diagonals in the upper part, one consist of the two diagonals in the lower left, and one consist of the single diagonal in the lower right.

Next, we consider the division of the interior of the polygon $\mathscr{P}$ into regions obtained by cutting it along all diagonals (but not edges) from the given matching M. Each region created by this division is bounded by some diagonals of M and by the boundary of the polygon $\mathscr{P}$.

cascade, k-bounded region

Definition 41. Regions bounded by exactly k diagonals are called *k-bounded* regions. Any maximal sequence of diagonals connected by 2-bounded regions is called a *cascade* (see Figure 18 for an example).

Lemma 42. *There is a bottleneck matching having at most three cascades.*

Proof. Let M be a matching provided by Lemma 40, with turning angles of all diagonals greater than $\pi/2$. There cannot be a region bounded by four 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 three 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. From this we conclude that there can be at most one 3-bounded region. $\square$

It is not possible for a matching to have exactly two cascades. If there were exactly two cascades, there would be a region defined by diagonals from both cascades. If that region were bounded by exactly one diagonal from each cascade, it would then be 2-bounded and, by definition of cascade, those two diagonals would belong to the same cascade. Otherwise, if that region were bounded by more than one diagonal from one of the two cascades, it would then be at least 3-bounded and, by definition of cascade, no two of its diagonals would belong to the same cascade, and hence we would have more than two cascades.

So, from Lemma 42 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 more elementary problems that will be used to find an optimal solution in both of these cases.

2.4.1 Matchings with at most one cascade

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

MATCHING^0, M^0

Definition 43. For i and j such that $\{i, \ldots, j\}$ is balanced, let MATCHING$^0(i, j)$ be the problem of finding an optimal matching $M^0_{i,j}$ of points in $\{i, \ldots, j\}$ using edges only.

MATCHING^1, M^1

Definition 44. For i and j such that $\{i, \ldots, j\}$ is balanced, let MATCHING$^1(i, j)$ be the problem of finding an optimal matching $M^1_{i,j}$ of points in $\{i, \ldots, j\}$, so that $M^1_{i,j}$ has at most one cascade, and the segment (i, j) belongs to a region bounded by at most one diagonal from $M^1_{i,j}$ different from (i,j).

When $\{i,\ldots,j\}$ is balanced, Property 27 ensures that solutions for $\textsc{Matching}^0(i,j)$ and $\textsc{Matching}^1(i,j)$ exist, so $M^0_{i,j}$ and $M^1_{i,j}$ are well defined.

Let i and j be such so that $\{i,\ldots,j\}$ is balanced. First, let us analyze how $\textsc{Matching}^0(i,j)$ can be reduced to smaller subproblems. The point i can be matched either with $o(i)$ or with $o^{-1}(i)$. The first option is always possible because Property 15 states that $o(i) \in \{i,\ldots,j\}$, but the second one is possible only if $o^{-1}(i) \in \{i,\ldots,j\}$ (it is also possible that $o(i) = o^{-1}(i)$, but no special analysis is needed for that). In the first case, $M^0(i,j)$ is constructed as the union of $(i,o(i))$, and optimal edge-only matchings for point sets $\{i+1,\ldots,o(i)-1\}$, if $|\{i,\ldots,o(i)\}| > 2$, and $\{o(i)+1,\ldots,j\}$, if $o(i) \neq j$, since both sets are balanced. The second case is similar, $M^0(i,j)$ is constructed as the union of $(o^{-1}(i),i)$, and optimal edge-only matchings for point sets $\{i+1,\ldots,o^{-1}(i)-1\}$, if $|\{i,\ldots,o^{-1}(i)\}| > 2$, and $\{o^{-1}(i)+1,\ldots,j\}$, if $o^{-1}(i) \neq j$, since both of these sets are balanced.

Next, we show how to reduce $\textsc{Matching}^1(i,j)$ to smaller subproblems. If i and j have different colors, then (i,j) is a feasible pair, and it is possible that $M^1_{i,j}$ includes this pair. In that case, $M^1_{i,j}$ is obtained by taking (i,j) together with $M^1(i+1,j-1)$, if $\{i,\ldots,j\} > 2$, since $\{i+1,\ldots,j-1\}$ is balanced. Now, assume that i is not matched to j (no matter whether (i,j) is feasible or not). Let k and l be the points in $\{i,\ldots,j\}$ which are matched to i and j in the matching $M^1_{i,j}$, respectively. By the requirement, (i,k) and (l,j) cannot both be diagonals, otherwise (i,j) would belong to the region bounded by more than one diagonal from $M^1_{i,j}$. If (i,k) is an edge, then, depending on the position of the diagonals that belong to the single cascade of $M^1_{i,j}$, the matching is constructed by taking (i,k) together either with $M^0_{i+1,k-1}$, if $|\{i,\ldots,k\}| > 2$, and $M^1_{k+1,j}$, if $k \neq j$, or with $M^1_{i+1,k-1}$, if $|\{i,\ldots,k\}| > 2$, and $M^0_{k+1,j}$, if $k \neq j$. Similarly, if (l,j) is an edge, then $M^1_{i,j}$ is constructed by taking (l,j) together either with $M^0_{l+1,j-1}$, if $|\{l,\ldots,j\}| > 2$, and $M^1_{i,l-1}$, if $i \neq l$, or with $M^1_{l+1,j-1}$, if $|\{l,\ldots,j\}| > 2$, and $M^0_{i,l-1}$, if $i \neq l$. All the mentioned matchings exist because their respective underlying point sets are balanced.

As these problems have optimal substructure, we can apply dynamic programming to solve them. If $\mathrm{bn}(M^0_{i,j})$ and $\mathrm{bn}(M^1_{i,j})$ are saved into $S^0(i,j)$ and $S^1(i,j)$, respectively, the following recurrent formulas can be used to calculate the solutions to $\textsc{Matching}^0(i,j)$ and

MATCHING$^1(i,j)$ for all pairs (i,j) such that $\{i,\dots,j\}$ is balanced.

$$S^0(i,j) = \min \begin{cases} \max \begin{cases} & |v_i v_{o(i)}| \\ \text{if } i+1 \neq o(1): & S^0(i+1, o(i)-1) \\ \text{if } o(i) \neq j: & S^0(o(i)+1, j) \end{cases} \\ \text{if } (o^{-1}(i) \in \{i,\dots,j\}): \\ \qquad \max \begin{cases} & |v_i v_{o^{-1}(i)}| \\ \text{if } i+1 \neq o^{-1}(i): & S^0(i+1, o^{-1}(i)-1) \\ \text{if } o^{-1}(i) \neq j: & S^0(o^{-1}(i)+1, j) \end{cases} \end{cases}$$

$$S^1(i,j) = \min \begin{cases} \max \begin{cases} & |v_i v_{o(i)}| \\ \text{if } i+1 \neq o(i): & S^0(i+1, o(i)-1) \\ \text{if } o(i) \neq j: & S^1(o(i)+1, j) \end{cases} \\ \max \begin{cases} & |v_i v_{o(i)}| \\ \text{if } i+1 \neq o(i): & S^1(i+1, o(i)-1) \\ \text{if } o(i) \neq j: & S^0(o(i)+1, j) \end{cases} \\ \max \begin{cases} & |v_{o^{-1}(j)} v_j| \\ \text{if } o^{-1}(j)+1 \neq j: & S^0(o^{-1}(j)+1, j-1) \\ \text{if } o^{-1}(j) \neq i: & S^1(i, o^{-1}(j)-1) \end{cases} \\ \max \begin{cases} & |v_{o^{-1}(j)} v_j| \\ \text{if } o^{-1}(j)+1 \neq j: & S^1(o^{-1}(j)+1, j-1) \\ \text{if } o^{-1}(j) \neq i: & S^0(i, o^{-1}(j)-1) \end{cases} \\ \text{if } (i,j) \text{ is feasible:} \\ \qquad \max \begin{cases} & |v_i v_j| \\ \text{if } i+1 \neq j: & S^1(i+1, j-1) \end{cases} \end{cases}$$

We fill values of S^0 and S^1 in order of increasing $j-i$, so that all subproblems are already solved when needed.

Beside the value of a solution MATCHING$^1(i,j)$, it is going to be useful to determine if pair (i,j) is necessary for constructing $M^1_{i,j}$.

necessary pair

Definition 45. We call an oriented pair (i,j) *necessary* if it is contained in every solution to MATCHING$^1(i,j)$.

Obviously, a pair can be necessary only if it is feasible. Computing whether (i,j) is a necessary pair can be easily incorporated into the calculation of $S^1(i,j)$. Namely the pair (i,j) is necessary, if (i,j) is an edge, or the equation for $S^1(i,j)$ achieves the minimum only in the last case (when (i,j) is feasible). If this is true, we set $necessary(i,j)$ to $\top$, otherwise we set it to $\bot$. Note that $necessary(i,j)$ does not imply $necessary(j,i)$.

We have $O(n^2)$ subproblems in total, 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.

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

We note that every matching with at most one cascade has a feasible pair $(k, k+1)$ such that the segment $(k, k+1)$ belongs to a region bounded by at most one diagonal from that matching. Indeed, if there are no diagonals in the matching, any pair $(k, k+1)$ where k and $k+1$ have different colors satisfies the condition. If there is a cascade, we take one of the two endmost diagonals of the cascade, let it be (i, j), so that there are no other diagonals from M in $\{i, \ldots, j\}$. Since $\{i, \ldots, j\}$ is balanced, there are two neighboring points $k, k+1 \in \{i, \ldots, j\}$ with different colors, and the pair $(k, k+1)$ is the one we are looking for.

Now, an optimal matching with at most one cascade can be found easily from calculated solutions to subproblems by finding the minimum of all $S^1(k+1, k)$ for all feasible pairs $(k, k+1)$ and reconstructing $M^1_{k+1,k}$ for k that achieved the minimum. The last (reconstruction) step takes only linear time.

2.4.2 Matchings with three cascades

As we already concluded, 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. We now 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 with $j \in \{i+1, \ldots, k-1\}$, 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 $M^1_{i,j}$, $M^1_{j+1,k}$ and $M^1_{k+1,i-1}$. (Note that these three feasible pairs do not necessarily belong to the combined matching, since they might not be necessary pairs in their respective 1-cascade matchings.)

To find the optimal matching we could run through all possible triplets (i, j, k) such that (i, j), $(j+1, k)$ and $(k+1, i-1)$ are feasible pairs, and see which one minimizes $\max\{S^1[i, j], S^1[j+1, k], S^1[k+1, i-1]\}$. However, this 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.

Candidate pairs and polarity

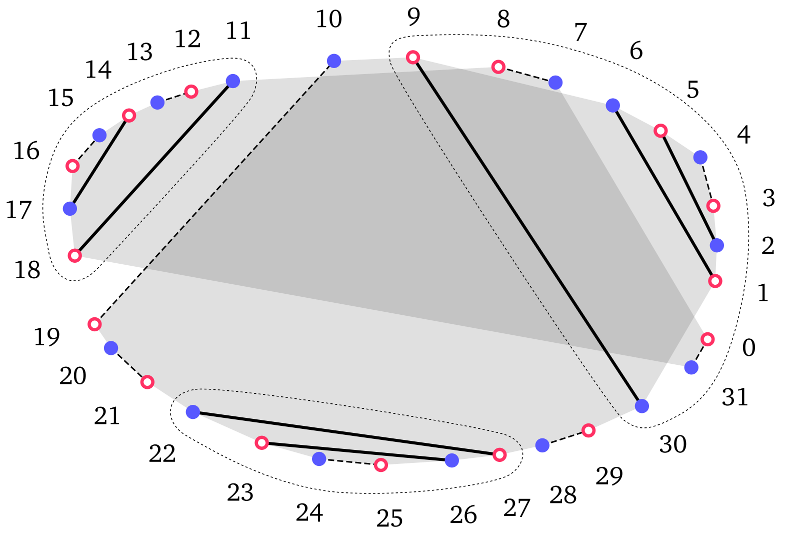
inner diagonals, inner region, inner pairs

Definition 46. 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, and we call this region the *inner region*. Matched pairs defining the boundary of the inner region are called the *inner pairs*.

For an example, see Figure 19.

Figure 19.

edges: dashed lines,
diagonals: solid lines,
cascades:
$\{(11, 18), (14, 17)\}$
$\{(22, 27), (23, 26)\}$
$\{(30, 9), (1, 6), (2, 5)\}$
inner diagonals:
$(11, 18)$
$(22, 27)$
$(30, 9)$,
inner pairs:
$(10, 19)$
$(20, 21)$
$(22, 27)$
$(28, 29)$
$(30, 9)$.



Since the inner region separates the cascades, there must be at least 3 inner pairs.

Lemma 47. *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.*

Proof. Take any 3-cascade bottleneck matching M. If it has an inner pair (i,j) that is not necessary, then (by definition) there is a solution to MATCHING$^1(i,j)$ 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 $\{i,\ldots,j\}$, pairs that were replaced are also a solution to MATCHING$^1(i,j)$, 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 pairs are necessary. The process has to terminate because the inner region is getting larger with each replacement. □

Definition 48. An oriented pair (i,j) is a *candidate pair*, if it is a necessary pair and $\tau(i,j) \leq 2\pi/3$. If a candidate pair is a diagonal, it is called a *candidate diagonal*.

candidate pair, candidate diagonal

Lemma 49. *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.*

Proof. Lemma 47 provides us with a 3-cascade matching M whose every inner pair is necessary. There are at least three inner pairs of M, so at least one of them has turning angle at most $2\pi/3$. Otherwise, the total turning angle would be greater than 2π, which is not possible. Such an inner pair is a candidate pair. □

Let us now take a look at an arbitrary candidate diagonal (i,j), and examine the position of points $\{i,\ldots,j\} \cap \mathcal{O}(i)$ relative to it. To do that, we locate points v_i and v_j and then define several geometric regions relative to their position, inspired by the geometric structure used in [43] to tackle the monochromatic version of the problem.

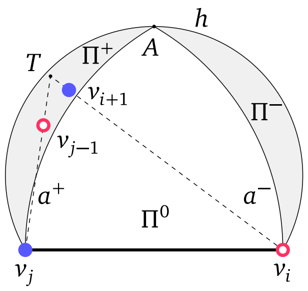


Figure 20. Geometric regions used for locating points $v_{i+1},\ldots,v_{j-1}$.

Firstly, we construct the circular arc h on the right side of the directed line v_iv_j, from which the line segment v_iv_j subtends an angle of $\pi/3$, see Figure 20. We denote the midpoint of h with A. Points v_i, A and v_j form an equilateral triangle, hence 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 $v_i v_j$, all depicted in Figure 20.

The following lemma is crucial in our analysis of bichromatic bottleneck matchings. Even though in statement it is similar to [43, Lemma 5], which was developed to tackle monochromatic bottleneck matchings, the proof we show here is much more involved, capturing the specifics of the bichromatic version of the problem and making use of the theory we developed around orbits.

Lemma 50. *For every candidate diagonal (i, j), the points from $\{i, \dots, j\} \cap \mathcal{O}(i)$ other than i and j lie either all in Π^- or all in Π^+.*

Proof. W.l.o.g. let us assume that point i is red. Since (i, j) is a diagonal, there are more than two points in $\{i, \dots, j\} \cap \mathcal{O}(i)$. Let T be the point of intersection of lines $v_i v_{i+1}$ and $v_j v_{j-1}$, see Figure 20. Since $\tau(i, j) \le 2\pi/3$, 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 $\{i, \dots, j\}$ must lie inside the triangle $\triangle v_i T v_j$, so there cannot be two points from $\{i, \dots, j\}$ such that one is on the right side of the directed line $v_i A$ and the other is on the left side of the directed line $v_j A$. This implies that either Π^- or Π^+ is empty.

W.l.o.g., let us assume that there are no points from $\{i, \dots, j\}$ on the right side of the directed line $v_i A$. By Δ^+ we denote the area bounded by a^+ and line segments $v_i v_j$ and $v_i A$, see Figure 21, so all points in $\{i, \dots, j\}$ lie in $\Pi^+ \cup \Delta^+$. It is important to note that both Π^+ and Δ^+ have the diameter $|v_i v_j|$, that is, no two points both inside Π^+ or both inside Δ^+ are at a distance of more than $|v_i v_j|$.

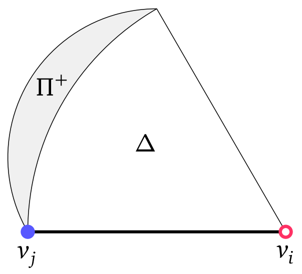


Figure 21. Regions Π^+ and Δ^+. Each of the regions has the diameter $|v_i v_j|$.

To complete the proof, we need to prove that no points of $\{i, \dots, j\} \cap \mathcal{O}(i)$ other than i and j lie in Δ^+, so for a contradiction we suppose the opposite, that there is at least one such point in Δ^+.

We denote the set of points in Π^+ (including j) with U. If there are points on a^+, we consider them to belong to U. The pair (i, j) is a feasible pair, so, by Property 19, the number of points from any orbit inside $\{i, \dots, j\}$ is even, implying that the parity of $|U \cap \mathcal{O}(i)|$ is the same as the parity of $|(\{i, \dots, j\} \setminus U) \cap \mathcal{O}(i)|$. We will analyze two cases depending on the parity of the number of points in $U \cap \mathcal{O}(i)$.

Case 1. There is an even number of points in $U \cap \mathcal{O}(i)$, and thus also in $(\{i,\dots,j\} \setminus U) \cap \mathcal{O}(i)$.

Let M be an optimal matching of points in $\{i,\dots,j\}$. The pair (i,j) is a candidate pair, and thus necessary, so it is contained in every optimal matching of points in $\{i,\dots,j\}$, including M, and hence $\mathrm{bn}(M) \geq |v_i v_j|$. To complete the proof in this case, we will construct another optimal matching M' that does not contain the pair (i,j), by joining two newly constructed matchings, M'_{out} and M'_{in}, thus arriving to a contradiction with the assumption that the pair (i,j) is a candidate pair.

We obtain the matching M'_{out} by arbitrarily matching the set $\{l,\dots,o(l)\}$, for each red-blue edge $(l,o(l))$ of $\mathcal{O}(i)$ in $\{i,\dots,j\}$, as illustrated in Figure 22 (note that in the figure only points from $\mathcal{O}(i)$ are depicted as points). More formally, M'_{out} is a union of matchings of sets $\{o^{2k}(i),\dots,o^{2k+1}(i)\}$, for each $k \in \{0,1,\dots,(s-1)/2\}$, where s is the smallest positive integer such that $o^s(i) = j$ (by Property 19 and Lemma 10, all these matchings exists). Since $|U \cap \mathcal{O}(i)|$ is even, points of each pair in M'_{out} are either both in U or both in $\{i,\dots,j\} \setminus U$, that is, they are either both in Π^+ or both in Δ^+, so the distance of each pair is at most $|v_i v_j|$, implying $\mathrm{bn}(M'_{out}) \leq |v_i v_j|$.

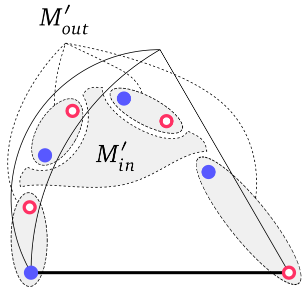


Figure 22. M'_{in} and M'_{out}; only points from $\mathcal{O}(i)$ are depicted as points.

The rest of the points in $\{i,\dots,j\}$ are all on the right side of blue-red edges of $\mathcal{O}(i)$, and by Property 30 the points they are paired up with in M are also on the right side of blue-red edges of $\mathcal{O}(i)$. Therefore, all those pairs are unobstructed by the segments in M'_{out}, and we can simply define M'_{in} to be the restriction of M to the set of those points from $\{i,\dots,j\}$ that are on the right side of blue-red edges of $\mathcal{O}(i)$.

All points in $\{i,\dots,j\}$ are covered by $M' = M'_{out} \cup M'_{in}$, and we have that $\mathrm{bn}(M') = \max\{\mathrm{bn}(M'_{in}), \mathrm{bn}(M'_{out})\} \leq \max\{\mathrm{bn}(M), |v_i v_j|\} = \mathrm{bn}(M)$. Since M is optimal, the equality holds and M' is optimal too. So we constructed an optimal matching M' on $\{i,\dots,j\}$ that does not contain the pair (i,j), and such a matching cannot exist, a contradiction.

Case 2. There is an odd number of points in $U \cap \mathcal{O}(i)$, and thus also in $(\{i,\dots,j\} \setminus U) \cap \mathcal{O}(i)$.

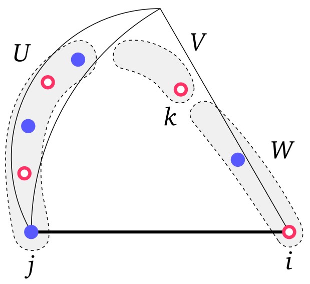


Figure 23. U, V and W; only points from $\mathcal{O}(i)$ are depicted as points.

Let k be the last point from the sequence $\{i,\ldots,j\} \cap \mathcal{O}(i)$ that lies in Δ^+, see Figure 23. Note that k must have the same color as i. We define $V := \{k,\ldots,j\} \setminus U$ and $W := \{i,\ldots,k-1\}$ (we earlier assumed that there is at least one point from $\mathcal{O}(i)$ other than i in Δ^+, so $k \neq i$).

By M we denote an optimal matching of points in $\{i,\ldots,j\}$ that minimizes the number of matched pairs between U and W. The pair (i,j) is a candidate pair, so it is a necessary pair, that is, every optimal matching of points in $\{i,\ldots,j\}$ contains (i,j), meaning that there is at least one matched pair between U and W in M. Let a be the last point in $\{i,\ldots,k-1\}$ matched to a point in U, and b be the point from U it is matched to, i.e. $(a,b) \in M$.

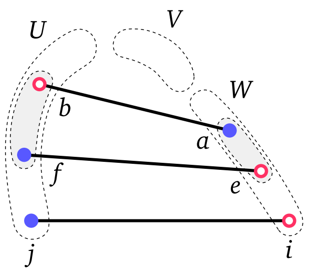


Figure 24. Even number of points in $\{i,\ldots,a\} \cap \mathcal{O}(a)$

If there is an even number of points in $\{i,\ldots,a\} \cap \mathcal{O}(a)$, then the numbers of red and blue points in that set are equal, so at least one of those points (which has a different color from a) must be matched to a point in U as well. Let that point be e and let its pair in U be f, see Figure 24.

We can now modify the matching by replacing (a,b), (e,f), and all the matched pairs between them with a matching of points in $\{e,\ldots,a\}$, and a matching of points in $\{b,\ldots,f\}$, which is possible by Property 19 and Lemma 10. Each newly matched pair has both its endpoints in the same set, either U or W, so its distance is at most $|v_i v_j|$, meaning that this newly constructed matching is optimal as well. This, however, reduces the number of matched pairs between U and W while keeping the matching optimal, which is in contradiction with the choice of M, so there must be at odd number of points in $\{i,\ldots,a\} \cap \mathcal{O}(a)$.

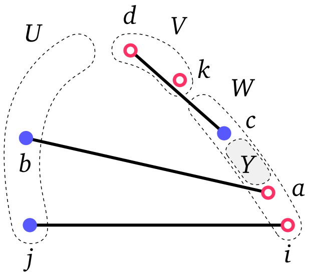


Figure 25. Odd number of points in $\{i,\ldots,a\} \cap \mathcal{O}(a)$

As the number of points from $\mathcal{O}(i)$ in $V \cup W$ is odd, and the only point in V from $\mathcal{O}(i)$ is k, there is an even number of points from $\mathcal{O}(i)$ in W. Since i and k are from the same orbit, there is an even number of points from any particular orbit in W (as can be seen by applying Property 19 to each pair of consecutive points of $\mathcal{O}(i)$ inside W). As there is an odd number of points in $\{i,\ldots,a\} \cap \mathcal{O}(a)$, there is an even number of points in $\{a,\ldots,k-1\} \cap \mathcal{O}(a)$, so at least one of them with a color different from a is matched with a point outside of W. Let c be the first such point in $\{a,\ldots,k-1\}$, see Figure 25. The way we chose a implies that c cannot be matched to a point in U, so it must be matched to some point d in V.

Let us denote the set $\{a,\dots,c\} \setminus \{a,c\}$ by Y. The choice of a guarantees that no point in Y is matched to a point in U. Points a and c belong to the same orbit, so by Property 19 there is an even number of points from any particular orbit in Y. Hence, if there is a point g_1 in Y matched to a point h_1 in V, then there must be another matched pair (g_2,h_2) from the same orbit such that $g_2 \in Y$, $h_2 \in V$, and g_1 and g_2 have different colors. We modify the matching by replacing (g_1,h_1), (g_2,h_2) and all the matched pairs between them with a matching M_g of points in $\{g_1,\dots,g_2\}$, and a matching M_h of points in $\{h_1,\dots,h_2\}$. This is again possible by Property 19 and Lemma 10. Matchings M_g and M_h are fully contained in W and V, respectively, so no matched pair of theirs is at a distance greater than $|v_iv_j|$, and the newly obtained matching is optimal as well. By iteratively applying this modification we can eliminate all matched pairs between Y and V, so that finally there is no matched pairs going out from Y, meaning no matched pair crosses either (a,c) or (b,d).

We are now free to “swap” the matched pairs between points a, b, c, and d, by replacing (a,b) and (c,d) with (a,c) and (b,d), because no other matched pair can possibly cross the newly formed pairs. We need to show that this swap does not increase the value of the matching. The pair (a,c) cannot increase the matching value because a and c are both in W, so their distance is at most $|v_iv_j|$. To show that the pair (b,d) also does not increase the value of the matching, we consider two cases based on the position of the point d.

Let Z be the midpoint of the line segment v_iA. Let us denote the region $(\Pi^+ \cup \Delta^+) \setminus \triangle v_iZv_j$ by Υ. No two points in Υ are at a distance greater than $|v_iv_j|$. The point b lies in Υ. If the point d lies in Υ as well, then $|bd| \le |v_iv_j|$. Otherwise, d lies in $\triangle v_iZv_j$, see Figure 26, and $\angle adb > \angle v_idv_j > \angle v_iZv_j = \pi/2$ (the first inequality holds because the points are in convex position). The angle $\angle adb$ is hence obtuse, and therefore $|bd| < |ab|$. But the pair (a,b) belongs to the original matching M, so the newly matched pair (b,d) also does not increase the value of the matching.

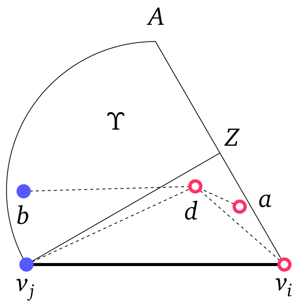


Figure 26. d lies in $\triangle v_iZv_j$

By making modifications to the matching M we constructed a new matching M' with the value not greater than the value of M. Since M is optimal, these values are actually equal, and the matching M' is also optimal. However, the pair (a,b) is contained in M, but not in M', and we did not introduce new matched pairs between U and W, so there is a strictly smaller number of matched pairs between U and W in M' than in M, which is a contradiction with the choice of M.

The analysis of both Case 1 and Case 2 ended with a contradiction, which completes the proof of the lemma. □

With $\Pi^-(i,j)$ and $\Pi^+(i,j)$ we respectively denote areas Π^- and Π^+ corresponding to an ordered pair (i,j). For candidate diagonals, the existance of the two possibilities given by Lemma 50 induces a concept of *polarity*.

polarity, pole

Definition 51. Let an oriented pair (i,j) be a candidate diagonal. If all points from $\{i,\dots,j\} \cap \mathcal{O}(i)$ other then i and j lie in $\Pi^-(i,j)$, we say that candidate diagonal (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 the pole in j.

Lemma 52. *No two candidate diagonals of the same polarity can have the same point as a pole.*

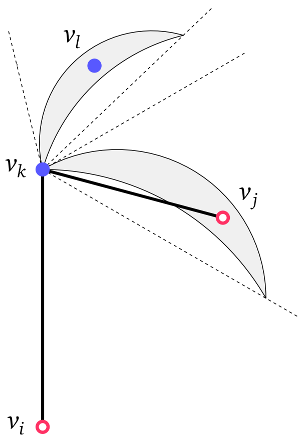


Figure 27. Two candidate diagonals of equal polarity cannot have the same pole.

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, w.l.o.g., that (i,k) and (j,k) are two such candidate diagonals, $i \neq j$, both with positive polarity, each having its pole in k. Since both (i,k) and (j,k) are feasible pairs, i, j and k belong to the same orbit. W.l.o.g., we also assume that the order of points in the positive direction is $i - j - k$, that is $j \in (\{i,\dots,k\} \cap \mathcal{O}(k)) \setminus \{i,k\}$, see Figure 27.

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 $l \in (\{j,\dots,k\} \cap \mathcal{O}(k)) \setminus \{j,k\}$. 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 lie in the region $\Pi^+(i,k)$. However, that cannot be the case, since $l \in (\{i,\dots,k\} \cap \mathcal{O}(k)) \setminus \{i,k\}$ as well, so we have a contradiction. □

As a simple corollary of Lemma 52, we get that there is at most linear number of candidate pairs.

Lemma 53. *There are $O(n)$ candidate pairs.*

Proof. Lemma 52 ensures that there are only two candidate diagonals with poles in the same point, one having positive and one having negative polarity. Therefore, there are at most n candidate diagonals of the same polarity, and, consequently, at most $2n$ candidate diagonals in total. The only other possible candidate pairs are edges, and there are exactly n edges, so there can be at most $3n$ candidate pairs. □

Finally, we combine our findings from Lemma 49 and Lemma 53, as described in the beginning of Section 2.4.2, to construct Algorithm 2.

Algorithm 2 Bottleneck Matching

```
Compute orbits.
Calculate $S^1[i,j]$ and $necessary(i,j)$, for all $i$ and $j$ such that $\{i,\dots,j\}$ is balanced, as described in Section 2.4.1.
$best \leftarrow \min\{S^1[k+1,k] : k \in \{0,\dots,2n-1\}, (k+1,k)$ is feasible$\}$
for all feasible $(i,j)$ do
    if $necessary(i,j)$ and $\tau(i,j) \le 2\pi/3$ then
        for all $k \in \{j+1,\dots,i-1\}$ such that $(j+1,k)$ is feasible do
            $best \leftarrow \min\{best, \max\{S^1(i,j), S^1(j+1,k), S^1(k+1,i-1)\}\}$
```

Theorem 54. *Algorithm 2 finds the value of bottleneck matching in $O(n^2)$ time.*

Proof. The first step, computing orbits, can be done in $O(n)$ time, as described in the proof of Lemma 24. The second step, calculating $S^1(i,j)$ and $necessary(i,j)$, for all (i,j) pairs, is done in $O(n^2)$ time, as described in Section 2.4.1. The third 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 49 tells us that there is a bottleneck matching among 3-cascade matchings such that one inner pair of that matching is a candidate pair, so the algorithm searches through all such matchings. We first fix the candidate pair (i,j) and then enter the inner for-loop, where we search for an optimal 3-cascade matching having (i, j) as an inner pair. Although the outer for-loop is executed $O(n^2)$ times, Lemma 53 guarantees that the if-block is entered only $O(n)$ times. The inner for-loop splits $\{j+1, \dots, i-1\}$ in two parts, $\{j+1, \dots, k\}$ and $\{k+1, \dots, i-1\}$, which together with $\{i, \dots, j\}$ 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 2 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.

2.5 Points on a circle

It this section we consider the case where all points lie on a circle. Obviously, the algorithm for the convex case can be applied here, but utilizing the geometry of a circle we can do better.

Employing the properties of orbits that we developed, we construct an $O(n)$ time algorithm for the problem of finding a bottleneck matching.

We will make use of the following lemma.

Lemma 55. *[20] If all the points of P lie on the circle, then there is a bottleneck matching in which each point i is connected either to $o(i)$ or $o^{-1}(i)$.*

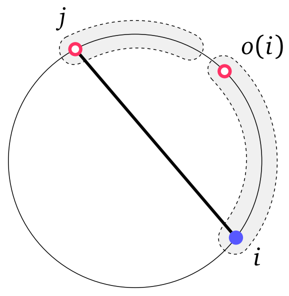


Figure 28. Illustrating the proof of Lemma 55

Proof. Let M be a bottleneck matching. If M does not contain a diagonal, then the statement of the lemma holds. Assume there is a diagonal (i, j) in M, see Figure 28. This diagonal cuts the circle into two parts. W.l.o.g., assume that the smaller part is on the right side of (i, j).

Since (i, j) is a diagonal, $o(i) \in \{i, \dots, j\}$ and sets $\{i, \dots, o(i)\}$ and $\{o(i)+1, \dots, j\}$ are both balanced. We construct a new matching by making a matching inside both of these sets, and copying matched pairs from M with endpoints in $\{j+1,\ldots,i-1\}$. The bottleneck value of this new matching is at most the bottleneck value of the original matching, since all matched pairs in $\{i,\ldots,j\}$ are at a distance smaller than the length of the diagonal (i,j).

We can repeat this process until there are no more diagonals, obtaining the matching consisting only of edges. □

This statement implies that there is a bottleneck matching M^E that can be constructed by taking alternating edges from each orbit, i.e. from each orbit we take either all red-blue or all blue-red edges. To find a bottleneck matching we can search only through such matchings, and to reduce the number of possibilities even more, we use properties of the orbit graph.

Theorem 56. *A bottleneck matching for points on a circle can be found in $O(n)$ time.*

Proof. From Property 37 we know that for an arbitrary weakly connected component of the orbit graph there is a Hamiltonian path $\mathscr{L}_0,\mathscr{L}_1,\ldots,\mathscr{L}_{m-1}$. For each $k \in \{0,\ldots,m-2\}$ there is an arc from $\mathscr{L}_k$ to $\mathscr{L}_{k+1}$, and those two orbits intersect each other. Since $\mathscr{L}_k \leq \mathscr{L}_{k+1}$, the only edges from $\mathscr{L}_k$ that intersect $\mathscr{L}_{k+1}$ are blue-red edges, and only edges from $\mathscr{L}_{k+1}$ that intersect $\mathscr{L}_k$ are red-blue edges. Hence, M^E cannot have blue-red edges from $\mathscr{L}_k$ and red-blue edges from $\mathscr{L}_{k+1}$. This further implies that there is $l \in \{0,1,\ldots,m\}$ such that $\mathscr{L}_0,\ldots,\mathscr{L}_{l-1}$ all contribute to M^E with red-blue edges and $\mathscr{L}_l,\ldots,\mathscr{L}_{m-1}$ all contribute to M^E with blue-red edges. Let M_l be the matching constructed by taking red-blue edges from $\mathscr{L}_0,\ldots,\mathscr{L}_{l-1}$, and blue-red edges from $\mathscr{L}_l,\ldots,\mathscr{L}_{m-1}$.

For each l, the value of M_l can be obtained as $\max\{RB_l, BR_l\}$, where RB_l is the length of the longest red-blue edge in $\mathscr{L}_0,\ldots,\mathscr{L}_{l-1}$, and BR_l is the length of the longest blue-red edge in $\mathscr{L}_l,\ldots,\mathscr{L}_{m-1}$. The computation of sequences RB and BR can be done in $O(n)$ total time, since RB_l is maximum of RB_{l-1} and the longest red-blue edge in $\mathscr{L}_{l-1}$, and BR_l is maximum of BR_{l+1} and the longest blue-red edge in $\mathscr{L}_l$. After we compute these sequences, we compute the value of M_l for each l, and take the one with the minimum value, which must correspond to a bottleneck matching.

We first compute orbits and Hamiltonian paths in $O(n)$ time (Lemma 24 and 38). Next, we compute the longest red-blue and blue-red edge in each orbit, which we then use to compute RB_l, BR_l, M_l, and finally M^E, as we just described. Each step in this process takes at most $O(n)$ time, so the total running time for this algorithm is $O(n)$ as well. $\square$

Part II

Network dilations and feed-links

Background and related work

As defined in the introduction of the thesis, the dilation between two points on the geometric network is the ratio between their network distance and their metric distance. Dilation of the whole network is the maximum of dilations over all pairs of points on the network, including the points on network edges. Spanning factor of the network is the maximum of dilations over pairs of points corresponding to network vertices.

Several results can be found in the literature analyzing the dilation and the spanning factor of a given network. The problem of computing the spanning factor is first introduced by Narasimhan and Smid in [39] as a dual problem to construction of t-spanners. They present efficient algorithms for computing the approximate value of spanning factor for different network types in Euclidean space.

Research that followed were predominantly on the topic of computing the dilation and spanning factors of different network types. Ebbers-Baumann at al. in [31] give an approximate algorithm for computing the dilation of planar polygonal chains. Later, Agarwal at al. in [6] give $O(n \log n)$-time exact algorithm for planar polygonal, and show how to apply the same algorithm to trees and cycles with an additional $O(\log n)$ factor. In the same article they consider three-dimensional Euclidean space, for which they give subquadratic algorithms for all considered problems. In [45], Wulff-Nilsen presents nondeterministic algorithm for computing the dilation of a planar network with the expected running time of $O(n^{3/2} \log^3 n)$.

Some methods for computing the dilation and spanning factor for rectilinear paths in L1 metric, as well as in fixed orientation metric, are presented by Wulff-Nilsen at al. in [47]

For a result on how to construct a network with a small dilation containing given points, we refer the reader to the work of Ebbers-Baumann, Grüne and Klein in [30].

Apart from the problems concerning computation of exact or approximate values, there is a plentiful research on problems of minimizing or maximizing dilation and spanning factor. These problems include extending the network in Euclidean spaces of arbitrary dimensions by addition of a new edge between two existing vertices, such that the resulting network has spanning factor as low as possible. In [34] Farshi, Giannopoulos and Gudmundsson give some approximate algorithms for this problem, and in [46] Wullf-Nilsen solves this problem with an exact algorithm of $O(n^3 \log n)$ time complexity.

Here, we are especially interested in extending a network with a new vertex. This problem is motivated by applications where we often encounter networks as models to real world structures, such as road or subway networks. It happens in geographical information systems that the locations of settlements are provided, but the data describing roads is only partial (e.g., only the location of larger roads are known, while smaller roads are missing from the database). However, in order to perform various network analysis a network needs to be connected, i.e. every settlement needs to lie on a road, which motivates us to find a way to extend the network so that the disconnected nodes become attached.

Depending on the requirements, there are many ways to connect a new node to an existing network. Probably the most straightforward one is to simply snap the location to the closest point on the network. This may be unsuitable as the node location is modified. Also, it can happen that two points geometrically close to each other are snapped to parts of network that are far away, which may be undesirable. Another approach is to link all new nodes inside a network face to the *feed-node* which is then connected to the network. Alternatively, each new node can be individually attached to the network using a *feed-link*. This approach was taken, e.g., by Dahlgren and Harrie in [24, 25], where the new location is simply connected to the nearest existing location by a feed-link.

In an attempt to reduce unnecessary detours, Aronov et al. in [14] introduced a more sophisticated way of choosing where on the existing network to attach the new feed-link, using the dilation as the measure of feed-link quality. They presented an algorithm to compute the feed-link achieving minimal dilation with a running time of $O(\lambda_7(n) \log n)$, where n is the number of vertices on the boundary of the face, and $\lambda_7(n)$ is the maximum length of a Davenport-Schinzel sequence of order 7 on n symbols, a slightly superlinear function. We consider this same setting and make an improvement by presenting an $O(n)$ time algorithm.

minimum eccentricity

Similar criterion for choosing the feed-link is recently considered by Bose at al. in [21]. They solve the problem of finding the *minimum eccentricity* feed-link, which minimizes the largest network distance from the new point to any point on the network.

The inverse problem, that is, removing elements from the existing network, is considered in [7] by Ahn, Farshi, Knauer, Smid and Wang. For a polygonal cycle in the plane they designed a nondeterministic algorithm which in $O(n \log^3 n)$ expected time finds an edge whose removal gives a network of the smallest possible dilation. They also give a deterministic algorithm of $O(n \log n)$ time complexity for finding an edge whose removal results in a network with the largest possible dilation. For the last problem in the case when the polygon is convex, they constructed a linear time algorithm.

Klein and Kutz in [36] considered the problem of obtaining a network with minimum spanning factor by removing all but a specified number of edges from a given geometric graph. They proved that this problem is NP-hard, no matter whether edge crossings are forbidden or not.

Chapter 3

Optimal feed-link placement

3.1 Introduction

3.1.1 Problem statement

For our purposes, a network P is an embedding of a connected graph into two-dimensional Euclidean space. Given two points p and q (on edges or vertices) of P, their *network distance* is defined as the length of the shortest curve contained in P connecting p and q. We define the *dilation* (sometimes also called the *detour*, or slightly less formally the *crow flight conversion coefficient*) as the ratio of the network distance between points p and q, and their Euclidean (crow flight) distance. The *geometric dilation* of the network P is the maximum detour taken over all pairs of points on the network.

network distance

dilation

Given a network P and a point p not on P, we want to extend the network by adding a single line segment, called *feed-link*, connecting p with a point on P. Note that a feed-link may have more than one point in the intersection with P, but we do not regard these points as connection points. An optimal feed-link is the one that minimizes the maximum dilation from the point p to a point on P.

feed-link

We solve the problem of finding an optimal feed-link in polygonal networks by constructing an algorithm which runs in linear time in the size of the polygon.

3.1.2 Our results

In this thesis, we give a linear time algorithm which finds the optimal feed-link – the one that minimizes the maximal dilation to the points on the boundary of the polygon. Although the initial problem statement given in [14] assumes that p lies inside the polygon and the polygon is simple, all of our calculations work out exactly the same for an arbitrary point p in the plane and arbitrary polygons, possibly self-intersecting, and hence, our results hold also in that more general setting.

Our algorithm is based on the idea that finding the maximum dilation can be reduced to finding the minimum slope of a certain line defined on the plot of the distance function from p to the boundary of P. This idea is then employed to construct a sweep algorithm over the points on the boundary. This result is published in [40].

The exposition of this result is organized as follows. In Section 3.2 we introduce the notation and give a formal definition of the problem. Then we split the problem into two symmetrical components, the left and the right dilation, so that we could perform our further analysis only on one of them. An alternative view of the problem is given in Section 3.3 by plotting the distance function from p to points on the boundary of P, and mapping feed-links to levers – line segments defined on the plot, with slope inversely proportional to the left dilation of a feed-link. In Section 3.4 we design a sweep algorithm which simulates moving of the lever, and outputs a sequence describing different states the lever passes through while moving. Finally, the sweep algorithm is run once for the left and once for the right dilation, and the two produced outputs are then combined to obtain the solution for the original problem. This merging process is described in Section 3.5.

3.2 Preliminaries

A polygon, which is not necessarily simple, is given as a list of its vertices $v_0, v_1, \ldots, v_{n-1}$. We denote the boundary of that polygon by P. We are also given a point p, not necessarily inside the polygon. A *feed-link* is a line segment pq, connecting p with some point $q \in P$.

feed-link

Definition 57. For any two points $q, r \in P$, the *dilation of r via q* is defined as

dilation of r via q

$$\delta_q(r) = \frac{|pq| + \mathrm{dist}(q,r)}{|pr|},$$

where $\mathrm{dist}(q,r)$ is the length of the shortest route between q and r over the polygon's boundary, and $|ab|$ is the Euclidean distance between points a and b, see Figure 29.

Definition 58. For a point $q \in P$, the *dilation via q* is defined as

dilation via q

$$\tilde{\delta}_q = \max_{r \in P} \delta_q(r).$$

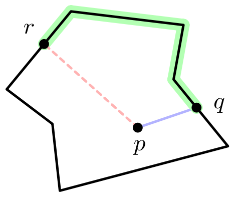


Figure 29. The concept of dilation.

The problem of finding the optimal feed-link is to find q such that $\tilde{\delta}_q$ is minimized.

Given two points $q, r \in P$, $P[q,r]$ is the portion of P obtained by going from q to r around the polygon in the positive (counterclockwise) direction, including the points q and r. Let $\mu(q,r)$ be the length of $P[q,r]$, and $\mu(P)$ the perimeter of P.

$\mu(q,r)$

$\mu(P)$

For the given $q \in P$, let q' be the point on P such that its network distance from q is exactly the half of the perimeter, that is, $\mu(q,q') = \mu(q',q) = \mu(P)/2$, see Figure 30. By $P^+[q]$ we denote $P[q,q']$, and by $P^-[q]$ we denote $P[q',q]$. Obviously, $P^+[q] \cup P^-[q] = P$ and $P^+[q] \cap P^-[q] = \{q, q'\}$.

left and right dilation

Definition 59. For given points $q \in P$ and $r \in P^+[q]$, the *left dilation of r via q* is defined as

$$\delta_q^+(r) = \frac{|pq| + \mu(q,r)}{|pr|}.$$

On the other hand, for $r \in P^-[q]$, the *right dilation of r via q* is defined as

$$\delta_q^-(r) = \frac{|pq| + \mu(r,q)}{|pr|}.$$

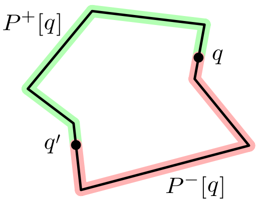


Figure 30. The left and the right portion of P observed from the point q.

When measuring $\text{dist}(q,r)$, the shortest path from q to r over P must lie entirely either in $P^+[q]$ or $P^-[q]$. This allows us to express the dilation of r via q using the left and the right dilation of r via q

$$\delta_q(r) = \begin{cases} \delta_q^+(r), & \text{if } r \in P^+[q] \\ \delta_q^-(r), & \text{if } r \in P^-[q] \end{cases}.$$

left and right dilation via q

Definition 60. Given point $q \in P$, the *left dilation via q* is defined as $\tilde{\delta}_q^+ = \max_{r \in P^+[q]} \delta_q^+(r)$, and the *right dilation via q* as $\tilde{\delta}_q^- = \max_{r \in P^-[q]} \delta_q^-(r)$.

Finally, the dilation via q can be expressed as

$$\tilde{\delta}_q = \max(\tilde{\delta}_q^+, \tilde{\delta}_q^-) = \max_{r \in P} \delta_q(r). \tag{3.1}$$

In the following two sections we will be concerned only with the left dilation, as the problem of finding the right dilation is, by definition, analogous. In Section 3.5 we will show how to combine our findings about the left and the right dilation to provide the answer to the original question. To simplify the notation, we omit the superscript + in Section 3.3 and Section 3.4, assuming that we deal with the left dilation.

3.3 Another view of the problem

Let us once again state the setting of our problem. Given a polygon boundary P and a point p, we want to compute the minimum dilation. In the previous section we saw how the dilation can be expressed in terms of the left and the right dilation. In this section, we are going to look at the distance function from p to the points on P, and observe how this function relates to the left dilation. These observations will enable us to work only with the distance function, instead of working with the polygon itself.

First, we parametrize points on P by defining $P(t)$, $t \in \mathbb{R}$, to be the point on P for which $\mu(v_0, P(t)) \equiv t \pmod{\mu(P)}$, see Figure 31.

$P(t)$

The distance of a point to the points on a straight line is known to be a hyperbolic function. The plot of the *distance function* $h(t) := |pP(t)|$ is an infinite sequence of hyperbola segments joined at their endpoints, where $(kn+r)$-th hyperbola segment corresponds to the r-th side of P, for $r \in \{0, 1, \dots, n-1\}$ and $k \in \mathbb{Z}$, see Figure 32.

$h(t)$

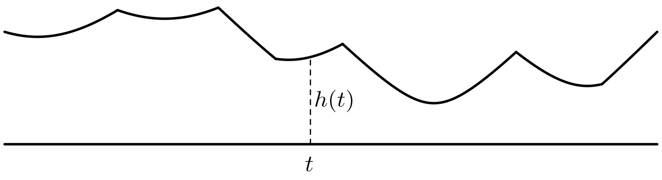


Figure 32. Plot of $h(t)$.

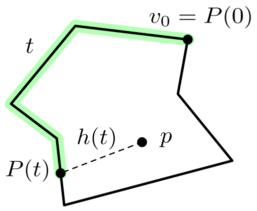


Figure 31. Parametrization of P.

For each $i = kn + r$, the hyperbola h_i is of the form $h_i(t) = \sqrt{(t - m_i)^2 + d_i^2}$, for some values m_i and d_i, so that $m_{kn+r} = m_r + k\mu(P)$, $d_{kn+r} = d_r$, and d_r is the distance between p and the line containing the r-th side of P. The left endpoint of the i-th hyperbola segment is $E_i := (e_i, h(e_i))$, where $e_i = k\mu(P) + \mu(v_r)$, and the right endpoint is at $(e_{i+1}, h(e_{i+1}))$. We will consider that each hyperbola segment contains its left endpoint, but not the right endpoint. By $H(t)$ we denote the point on the plot of h corresponding to the parameter t, so $H(t) := (t, h(t))$. The plot is, obviously, periodic, with the period of $\mu(P)$, that is, $H(t) = H(t + k\mu(P))$. We denote the i-th hyperbola segment with $\mathscr{H}_i$.

E_i, e_i

$H(t)$

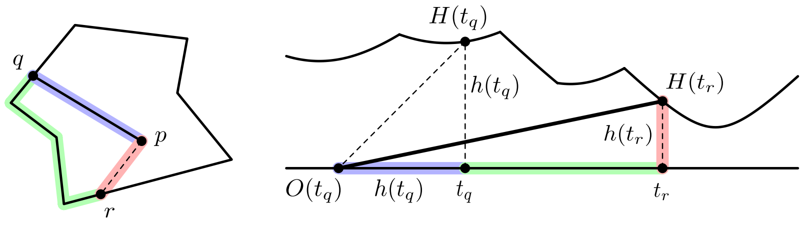


Figure 33. Relation between the dilation and the slope.

Let $o(t) = t - h(t)$, and $O(t) = (o(t), 0)$, see Figure 33. We also define $o_i(t) = t - h_i(t)$, and $O_i(t) = (o_i(t), 0)$. Given points $q \in P$ and $r \in P^+[q]$, we have their corresponding parameters t_q and t_r, such that $t_q \leq t_r \leq t_q + \mu(P)/2$. The *slope* of the line passing

$o(t)$, $O(t)$, $o_i(t)$, $O_i(t)$

slope, $s(t_q, t_r)$

through the points $O(t_q)$ and $H(t_r)$ is

$$\begin{aligned} s(t_q,t_r) &:= \operatorname{slope}\big(\ell(O(t_q),H(t_r))\big) \\ &= \frac{h(t_r)}{t_r - t_q + h(t_q)} \\ &= \frac{|pP(t_r)|}{\mu(P(t_q),P(t_r)) + |pP(t_q)|} \\ &= \frac{1}{\delta^+_{P(t_q)}(P(t_r))}, \end{aligned} \tag{3.2}$$

hence the slope between $O(t_q)$ and $H(t_r)$ is equal to the inverse of the left dilation of r via q.

$\tilde{s}(t_q)$ **Definition 61.** We define $\tilde{s}(t_q)$ to be the lowest slope from $O(t_q)$ to $H(t_r)$ among all $t_r \in [t_q, t_q + \mu(P)/2]$.

From the previous observation it follows that this slope equals the inverse of the left dilation via q,

$$\begin{aligned} \tilde{s}(t_q) &:= \min_{t_r \in [t_q, t_q+\mu(P)/2]} s(t_q,t_r) \\ &= \min_{t_r \in [t_q, t_q+\mu(P)/2]} \frac{1}{\delta^+_{P(t_q)}(P(t_r))} \\ &= \frac{1}{\max_{r \in P^+[q]} \delta^+_q(r)} \\ &= \frac{1}{\tilde{\delta}^+_q}. \end{aligned} \tag{3.3}$$

Obviously, $\tilde{s}(t) \in (0,1]$ because it is strictly positive and $\tilde{s}(t) \leq s(t,t) = 1$. This enables us to estimate the dilation by looking at the slope of the line we just defined.

Lemma 62. *For any two distinct values of t_1 and t_2,*

$$|h(t_2) - h(t_1)| \leq |t_2 - t_1|.$$

Proof. From the triangle inequality we have $|h(t_2) - h(t_1)| \leq |P(t_1)P(t_2)|$, and from the parametrization by distance along P it follows that $|P(t_1)P(t_2)| \leq |t_2 - t_1|$. These inequalities readily imply the statement of the lemma. □

So far, $\tilde{s}(t_q)$ was defined as the minimum only among the slopes $s(t_q,t_r)$ where t_r belongs to the interval $[t_q, t_q + \mu(P)/2]$. However, from Lemma 62 follows that $s(t_q, t_r)$ cannot be less than 1 when $t_r \in [o(t_q), t_q]$, and $\tilde{s}(t_q)$ is at most 1, so this interval can be extended, and we have

$$\tilde{s}(t_q) = \min_{t_r \in [o(t_q), t_q + \mu(P)/2]} s(t_q, t_r).$$

This extension of the interval enables us to define the *lever* and to construct the following *sliding lever algorithm* without having to worry about certain unwanted cases.

3.4 Sliding lever algorithm

In this section we define the *lever*, the central object of our analysis, as well as the possible states that the lever can have. We want to simulate the movement of the lever, so we analyze all the events that lead to state changes. That allows us to construct a sweep algorithm; however, the most straightforward algorithm will not run in linear time. To fix that we design an additional algorithm to precompute the set of the so-called *retargeting positions*, which we use for certain helper events (*jump destination change events*).

3.4.1 Lever

For a fixed t, consider the line segment having the slope $\tilde{s}(t)$, with one endpoint at $O(t)$ and the other at $(t + \mu(P)/2, \tilde{s}(t)(\mu(P)/2 + h(t)))$, see Figure 34. Let us call that line segment the *lever for t*. Note that the lever only touches the plot, never intersecting it properly.

lever for t

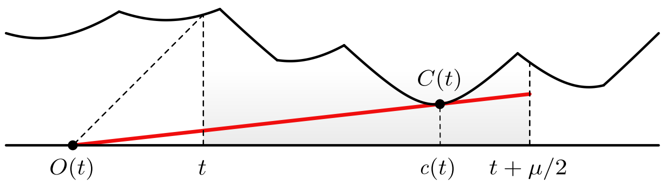


Figure 34. Lever.

Let $C(t)$ be the leftmost point in which the lever for t touches the plot, and let $c(t)$ be such that $H(c(t)) = C(t)$. Then $c(t) \in [o(t), t + \mu(P)/2]$ and it is the lowest value in this interval for which $\tilde{s}(t) = s(t, c(t))$. Coming back to the original setup, this means that the left dilation via $P(t)$ reaches its maximum for $P(c(t))$.

We now continuously decrease parameter t and observe what is happening with the updated lever. The following monotonicity lemma states that when t is decreasing $o(t)$ and $c(t)$ are decreasing as well, which means that decreasing t corresponds to “dragging" the lever in the leftward direction.

Lemma 63. *For $t_1 < t_2$ we have $o(t_1) \leq o(t_2)$ and $c(t_1) \leq c(t_2)$.*

Proof. Suppose $t_1 < t_2$. Using Lemma 62 we get

$$\begin{aligned} h(t_2) - h(t_1) &\leq t_2 - t_1, \\ t_1 - h(t_1) &\leq t_2 - h(t_2), \\ o(t_1) &\leq o(t_2). \end{aligned}$$

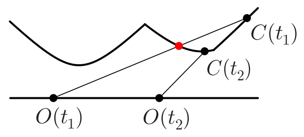


Figure 35. A situations leading to a contradiction (the depicted points cannot be in this arrangement, resulting in invalid pictures)

To show that $c(t_1) \leq c(t_2)$, assume the opposite, that $c(t_1) > c(t_2)$. Then, $t_1 < t_2 < c(t_2) < c(t_1) \leq t_1 + \mu(P)/2$, and $c(t_1) \in [t_2, t_2 + \mu(P)/2]$.

For a contradiction, suppose first that line segments $O(t_1)C(t_1)$ and $O(t_2)C(t_2)$ do not intersect, see Figure 35. Since $o(t_1) \leq o(t_2)$, the segment $O(t_2)C(t_2)$ lies completely under $O(t_1)C(t_1)$, and because $O(t_2)C(t_2)$ touches the plot in $C(t_2)$, the plot must intersect $O(t_1)C(t_1)$ in some point left of $C(t_2)$ and, hence, left of $C(t_1)$. That is a contradiction since $C(t_1)$ is the leftmost point where the lever for t_1 intersects the plot.

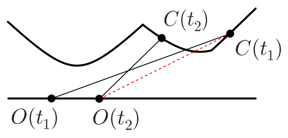


Figure 36. A situations leading to a contradiction (the depicted points cannot be in this arrangement, resulting in invalid pictures)

If line segments $O(t_1)C(t_1)$ and $O(t_2)C(t_2)$ do intersect, see Figure 36, then $C(t_1)$ lies under the line $O(t_2)C(t_2)$, and we we have

$$s(t_2, c(t_1)) < s(t_2, c(t_2)) = \min_{t \in [t_2, t_2 + \mu(P)/2]} s(t_2, t) \leq s(t_2, c(t_1)),$$

which again is a contradiction. This concludes the proof of the lemma. □

3.4.2 States

In order to be able to simulate the continuous leftward motion of the lever, we transform it to an iteration over a discrete sequence of states. We define different lever states depending on how the lever is positioned relative to the sequence of hyperbola segments.

phase

When $t \in [e_i, e_{i+1})$ and $c(t) \in [e_j, e_{j+1})$, we say that the lever for t is in the *phase* $\langle i, j \rangle$. The phase in which the lever is, together with the manner in which the lever touches the plot define the *state of the lever*. There are three possible ways for the lever to touch the plot, denoted by $\mathscr{K}$ (arc tangency), $\mathscr{Y}$ (endpoint sliding), and $\mathscr{V}$ (wedge touching).

state of the lever

- *State* $\langle i,j\rangle^{\mathscr{K}}$: $c(t) < t + \mu(P)/2$ and the lever is the tangent to $\mathscr{H}_j$.

  When t is decreasing, the lever is sliding to the left along h_j maintaining tangency, thus continuously decreasing its slope.

State $\langle i,j\rangle^{\mathscr{K}}$

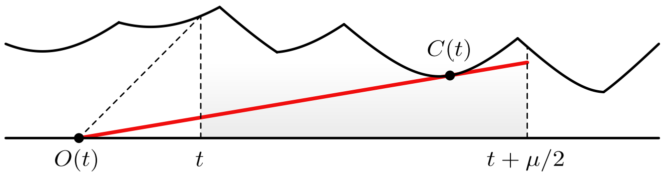


Figure 37. State $\langle i,j\rangle^{\mathscr{K}}$

- *State* $\langle i,j\rangle^{\mathscr{Y}}$: $c(t) = t + \mu(P)/2$.

  Point $C(t)$ is the right endpoint of the lever. It is the only point where the lever touches the plot. When t is decreasing, the lever is moving to the left while keeping its right endpoint on h_j.

State $\langle i,j\rangle^{\mathscr{Y}}$

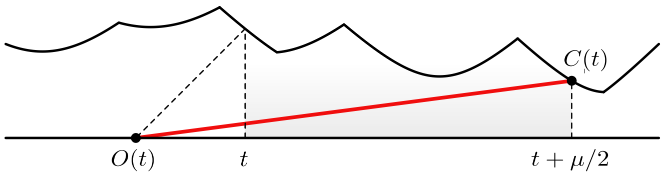


Figure 38. State $\langle i,j\rangle^{\mathscr{Y}}$

- *State* $\langle i,j\rangle^{\mathscr{V}}$: $c(t) < t + \mu(P)/2$ and the lever is passing through the point $H(e_j)$, the endpoint between hyperbola segments $\mathscr{H}_{j-1}$ and $\mathscr{H}_j$.

  This situation occurs only if $m_{j-1} > m_j$. The two neighboring hyperbola segments then form a “wedge" pointing downwards, and when t is decreasing the lever is sliding to the left while maintaining contact with the tip of the wedge, thus continuously decreasing its slope.

State $\langle i,j\rangle^{\mathscr{V}}$

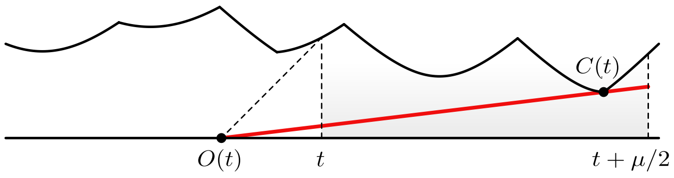


Figure 39. State $\langle i,j\rangle^{\mathscr{V}}$

3.4.3 Jumping and retargeting

In the process of decreasing t and dragging the left endpoint of the lever towards left, various events can take place. We will first devote some attention to the most challenging kind of events, which we call *jumping events*. These are events in which $C(t)$ abruptly changes its position by switching to a different hyperbola segment. In order to efficiently find state transition events that include jumps, we always need to know to which hyperbola segment we can jump from the current position. There is always at most one such target hyperbola segment, and we will show how to keep track of it.

jumping events

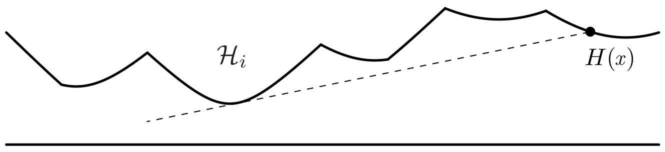


Figure 40. Jump destination for x. $jump(x) = i$

Consider some point $H(x)$ on the plot. Let jump(x), the *jump destination for x*, be the index of the hyperbola segment which contains the rightmost point $H(w)$ on the plot such that $w < x$ and the ray from $H(x)$ through $H(w)$ only touches the plot, but does not intersect it properly, see Figure 40. That is, jump(x) is the index of the lowest visible hyperbola segment when looking from the point $H(x)$ to the left. If there is no such w, because hyperbola segments on the left are obscured by the segment containing $H(x)$, then we set jump(x) to be the index of the hyperbola segment containing $H(x)$.

jump(x), *jump destination for x*

Consider only the values of x at which jump(x) changes value. We call such values *retargeting positions*, and points $H(x)$ *retargeting points*. There are two types of retargeting points. *Retargeting points of the first type* are the points on $\mathscr{H}_k$ in which jump destination changes from i to j, where $i < j < k$, see Figure 41.*Retargeting points of the second type* are the points on $\mathscr{H}_k$ in which jump destination changes from k to j, where $j < k$, see Figure 42.

retargeting positions

retargeting points

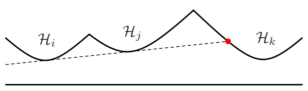


Figure 41. Retargeting point of the first type.

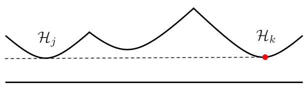


Figure 42. Retargeting point of the second type.

We construct an algorithm for finding all retargeting points. It is similar to finding lower convex chain in Andrew's monotone chain convex hull algorithm [13]. Our algorithm, however, runs in linear time because the sequence of hyperbola segments is already sorted. Before we give the algorithm, we describe the process and the supporting structure in more detail.

Given a set A of hyperbola segments, we take a look at the convex hull of their union and divide its boundary into the upper and lower part (i.e., the part lying above the segments, and the one below the segments). We are interested only in those hyperbola segments from A that have a nonempty intersection with the lower part of the convex hull boundary, having in mind that each hyperbola segment contains its left endpoint, but not the right one. Let us call the sequence of all such hyperbola segments, ordered from left to right, the *convex support* for A, see Figure 43.We say that three hyperbola segments from the plot of h are in convex position if no line segment connecting a point from the left and a point from the right hyperbola segment passes completely below the middle hyperbola segment. Note that any three hyperbola segments of any convex support are in convex position.

convex support

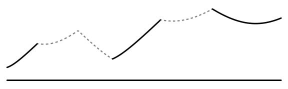


Figure 43. Convex support for all shown hyperbola segments is marked with solid lines.

Let j_0 be the index of a hyperbola segment that contains one of the global minima of h. Starting from $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}\}$, we process segments from left to right and maintain the convex support for the set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_k\}$, where k is the index of the segment being processed.

We use a stack to represent the convex support (only the indices of hyperbola segments are stored). Suppose the stack already contains the convex support for $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_{k-1}\}$, and we want to add a new segment $\mathcal{H}_k$. We need to make changes to the stack so that it now represents the convex support for the new, extended, set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_k\}$. To achieve this, we pop segments from the stack until the last two segments still in the stack, together with $\mathcal{H}_k$, are in convex position. (Note that $\mathcal{H}_{j_0}$ will never be popped this way, as it contains a global minimum.) Finally, in the case where $\mathcal{H}_k$ belongs to the convex support of $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_k\}$, we push it on the stack. (The intersection of $\mathcal{H}_k$ with the lower part of the convex hull is possibly empty since $\mathcal{H}_k$ does not contain its right endpoint.)

Let $\mathrm{cl}(X)$ denote the closure of a point set X, so $\mathrm{cl}(\mathcal{H}_i) = \mathcal{H}_i \cup \{E_{i+1}\}$, and let us consider the line l touching both $\mathrm{cl}(\mathcal{H}_i)$ and $\mathrm{cl}(\mathcal{H}_j)$, $i < j$, from below. If such a line is not unique, which can possibly happen only when $j = i + 1$, we take l to be the line with the smallest slope (that is, the line tangent to $\mathrm{cl}(\mathcal{H}_i)$ in E_{i+1}). We call the line l the *common tangent* of $\mathcal{H}_i$ and $\mathcal{H}_j$. It can be computed in a constant time, and in the following algorithm it is obtained as the return value of the function TANGENT(i, j).

$\mathrm{cl}(X)$

common tangent

Note that if Z is a point with larger first coordinate than the point $l \cap \mathrm{cl}(\mathcal{H}_j)$ (i.e., Z is to the right of $l \cap \mathrm{cl}(\mathcal{H}_j)$) and below the plot, then the point Z sees $\mathcal{H}_i$ lower than $\mathcal{H}_j$ if Z is below l, and $\mathcal{H}_j$ lower than $\mathcal{H}_i$ if Z is above l. We will use this fact in the analysis of the algorithm.

Now we are ready to construct Algorithm 3, which shows how do we get retargeting points as intersections of hyperbola segments and common tangents of successive segments from the convex support.

Algorithm 3 Retargeting Points

```
Input: Description of a sequence of hyperbola segments.
Output: $RetargetingPoints$ – the sequence of all retargeting
  points.
  $RetargetingPoints \leftarrow [\ ]$
  PUSH($j_0 - 1$)
  PUSH($j_0$)
  for $k \leftarrow j_0 + 1$ to $j_0 + n$ do
      loop
          $i \leftarrow$ Second-to-top element of the stack
          $j \leftarrow$ Top element of the stack
          $l_1 \leftarrow$ TANGENT($i, j$)
          if $l_1 \cap \mathcal{H}_k \neq \emptyset$ then
              Let $g$ be the leftmost point of $l_1 \cap \mathcal{H}_k$.
              Append $g$ to $RetargetingPoints$.
              Set jump destination of $g$ to $j$.
              POP()
          else
              $l_2 \leftarrow$ TANGENT($j, k$)
              if $l_2 \cap \mathcal{H}_k \neq \emptyset$ then
                  Let $g$ be the only point of $l_2 \cap \mathcal{H}_k$.
                  Append $g$ to $RetargetingPoints$.
                  Set jump destination of $g$ to $j$.
                  PUSH($k$)
              break loop
```

Theorem 64. *Algorithm 3 finds all retargeting positions, ordered from left to right, together with jump destinations of those positions, in $O(n)$ time.*

Proof. To show the correctness of this algorithm, we first observe that each reported point must be a retargeting point since the jump destination changes at it.

Indeed, points reported in the outer “if" branch lie on the common tangent of two successive hyperbola segments $\mathcal{H}_i$ and $\mathcal{H}_j$ from the convex support, and $\mathcal{H}_k$ is the first segment to be intersected by that tangent. The point of the intersection is the boundary between the region of $\mathscr{H}_k$ from which $\mathscr{H}_i$ is the lowest segment when looking to the left and the region of $\mathscr{H}_k$ for which such lowest segment is $\mathscr{H}_j$, as shown in Figure 41. Thus, a point g reported in this branch has the property that points on $\mathscr{H}_k$ just left and just right of the point g have $\mathscr{H}_j$ and $\mathscr{H}_i$ as their jump destinations, respectively, so it is a retargeting point of the first type.

The inner “if" branch occurs when the segment $\mathscr{H}_k$ is appended to the convex support, in which case there is a point on $\mathscr{H}_k$ acting as a boundary between the region of $\mathscr{H}_k$ from which no other segment is visible (when looking to the left), and the region of $\mathscr{H}_k$ from which $\mathscr{H}_j$ is visible, and such point lies on the common tangent of $\mathscr{H}_j$ and $\mathscr{H}_k$, as shown in Figure 42. Therefore, the point g reported in this branch has the property that points on $\mathscr{H}_k$ just left and just right of the point g have $\mathscr{H}_j$ and $\mathscr{H}_k$ as their jump destinations, respectively, so it is a retargeting point of the second type.

Next, let us make sure that no retargeting points were omitted by this algorithm. Consider a retargeting point g lying on $\mathscr{H}_k$.

If g is retargeting point of the first type, it must lie on $\textsc{Tangent}(i, j)$ for some $i < j < k$ (Figure 41). Note that there can be no $\mathscr{H}_r$ with $r < k$ such that it reaches below $\textsc{Tangent}(i, j)$; otherwise r would be a jumping destination for g. That implies that both $\mathscr{H}_i$ and $\mathscr{H}_j$ are the part of the lower convex hull of the segments left from $\mathscr{H}_k$, which further means that $\mathscr{H}_i$ and $\mathscr{H}_j$ are two consecutive elements of the convex support for the set $\{\mathscr{H}_{j_0-1}, \mathscr{H}_{j_0}, \dots, \mathscr{H}_{k-1}\}$. Since $\textsc{Tangent}(i, j)$ intersects $\mathscr{H}_k$, the same must be true for any pair of consecutive segments $\mathscr{H}_{i'}$ and $\mathscr{H}_{j'}$ from the convex support, with $i < i' < j' < k$. Otherwise, there would be three segments from the convex support not in convex position. The algorithm starts from the last two segments in the convex support and repeats moving to the previous pair as long as there is an intersection of the pair’s common tangent with $\mathscr{H}_k$. That guarantees g will be found and reported as the retargeting point in the outer “if” branch.

The second case, when g is retargeting point of the second type, is treated similarly. In this case g lies on $\textsc{Tangent}(j, k)$ for some $j < k$ Figure 42. There can be no $\mathscr{H}_r$ with $r < k$ such that it reaches below $\textsc{Tangent}(j, k)$; otherwise r would be a jumping destination for g. That implies that $\mathscr{H}_j$ is a part of the lower convex hull of the segments left from $\mathscr{H}_k$, which further means that $\mathscr{H}_j$ is an element of the convex support for the set $\{\mathscr{H}_{j_0-1}, \mathscr{H}_{j_0}, \dots, \mathscr{H}_{k-1}\}$. Since TANGENT$(j,k)$ touches $\mathscr{H}_k$ from below, the common tangent of each pair of consecutive segments $\mathscr{H}_{i'}$ and $\mathscr{H}_{j'}$, with $j \le i' < j' < k$, from the convex support must intersect $\mathscr{H}_k$. Otherwise, there would be three segments from the convex support not in convex position. For the same reason the common tangent of $\mathscr{H}_j$ and the segment immediately before it in the convex support cannot intersect $\mathscr{H}_k$. The algorithm starts from the last two segments in the convex support and repeats moving to the previous pair as long as there is an intersection of the pair's common tangent with $\mathscr{H}_k$. Finally, the algorithm reaches the rightmost pair of two consecutive segments from convex support whose common tangent does not intersect $\mathscr{H}_k$. The right segment from that pair is exactly $\mathscr{H}_j$. In that moment the point g is found and reported as retargeting point in the inner "if" branch.

The running time of algorithm is $O(n)$, since each index $k \in \{j_0 + 1, \dots, j_0 + n\}$ is pushed on the stack and popped from the stack at most once, and output of TANGENT() and intersections can be computed in a constant time. The number of retargeting points reported is, therefore, also $O(n)$.

Retargeting points reported by the algorithm come in sorted order, from left to right, which is explained by following observations. Retargeting points reported in a single iteration of the outer for-loop belong to the same hyperbola segment, and segments come in left-to-right order. Retargeting points reported on the same hyperbola segment are also in the same order: inside the inner loop, $\mathscr{H}_k$ is consecutively intersected with lines such that each line is of lower slope than previous and lies beneath it under $\mathscr{H}_k$. Hence, each subsequent intersection point lies to the right of the previous one.

This algorithm finds only retargeting points from a single period of the plotted function, but all other retargeting points can be obtained by simply translating these horizontally by the integral number of periods $\mu(P)$. □

3.4.4 Events

In the process of decreasing t and dragging the left endpoint of the lever towards left, the lever state changes at certain moments.

We call such events *state transition events*. It is crucial for us to be able to efficiently calculate where these events can occur. If the current lever position, t_c, and the current state are known, the following event can be determined by maintaining the set of conceivable future events of which at least one must be realized, and proceeding to the one that is the first to take place, i.e., the one with the largest t not larger than t_c. To do that, we must know how to calculate the value of t for each event.

state transition events

We list all types of events that can happen while moving the lever leftwards, and for each we show how to calculate the value of t, the lever position at which the event occurs. We will give a polynomial equation describing each event type, which will be solved either for t or for $o_i(t)$. Once we have $o_i(t)$, it is easy to obtain t, as

$$t = \frac{o_i(t)^2 - d_i^2 - m_i^2}{2(o_i(t) - m_i)}. \tag{3.4}$$

In the process of determining t, we will repeatedly encounter fixed degree polynomial equations. Solving them can be assumed to be a constant time operation, see [28].

We will frequently use the following two utility functions, $c_j(o)$ and $s_j(o)$.

For $o < m_j$, let $c_j(o)$ be such that $H(c_j(o)) := (c_j(o), h(c_j(o)))$ is the contact point of the hyperbola h_j and its tangent through the point $(o, 0)$. Given o, the value $c_j(o)$ can be calculated by solving the equation $h'_j(c_j(o)) = h_j(c_j(o))/(c_j(o) - o)$, which results in

$c_j(o)$

$$c_j(o) = \frac{d_j^2 + m_j^2 - m_j o}{m_j - o}. \tag{3.5}$$

Function $s_j(o)$ is defined as the slope of the tangent to the hyperbola h_j through the point $(o, 0)$, that is, the line through points $(o, 0)$ and $H(c_j(o))$,

$s_j(o)$

$$s_j(o) = \frac{h_j(c_j(o))}{c_j(o) - o} = 1\Big/\sqrt{\left(\frac{m_j - o}{d_j}\right)^2 + 1}. \tag{3.6}$$

These functions are used when we know that the lever for t is a tangent to some hyperbola segment $\mathscr{H}_j$. Then we know that the lever touches $\mathscr{H}_j$ at the point with coordinate $c_j(o(t))$, and that its slope equals $s_j(o(t))$.

Let us first consider *jump destination change event*, a type of event

jump destination change event

which is not a state transition event. Nevertheless, it is still necessary to react to events of this kind in order to update a parameter needed for calculating events that do change state. As we decrease t, we keep track of jump destination for position $c(t)$ in variable $j_m :=$ jump$(c(t))$.

Jump destination change event

Jump destination j_m changes whenever $C(t)$ passes over some retargeting point. At that moment it is necessary to recalculate all future events involving jumps, since j_m is used for their calculation. Let z be the next retargeting position, i.e., the rightmost one that lies to the left of $c(t_c)$, where t_c is current lever position. Depending on the lever state, we calculate the event position in one of the following ways.

- The current state is $\langle i, j\rangle^{\mathscr{V}}$

  The jump destination change event cannot occur before leaving this state since $C(t)$ stands still at the “wedge tip", so it cannot pass over any retargeting point.

- The current state is $\langle i, j\rangle^{\mathscr{K}}$

  Here the lever is tangent to $\mathscr{H}_j$, so this event can only happen if $z > m_j$. Otherwise, the lever would have nonpositive slope when touching the plot at $H(z)$. The equation describing this event is
  $$c_j(o_i(t)) = z,$$
  and it solves to
  $$o_i(t) = \frac{d_j^2 - zm_j + m_j^2}{m_j - z}.$$

- The current state is $\langle i, j\rangle^{\mathscr{Y}}$

  The right endpoint of the lever slides over $\mathscr{H}_j$ and will coincide with $H(z)$ at
  $$t = z - \mu(P)/2.$$

Note that j_m is not used in the description of the lever state, so, as already noted, jump destination change event does not change the current state.

All the other events that need to be considered are state transition events.

State transition events

In the following list we give all possible types of state transition events, and we show how to calculate corresponding t value for each of them. To make this enumeration easier to follow, we group event types in four groups, depending on the resulting phase of the event: $\langle i, j\rangle$, $\langle i-1, j\rangle$, $\langle i, j-1\rangle$ and $\langle i, j_m\rangle$.

Event types leading to $\langle i, j\rangle$ phase:

- $\langle i, j\rangle^{\mathscr{Y}} \to \langle i, j\rangle^{\mathscr{K}}$ *and* $\langle i, j\rangle^{\mathscr{K}} \to \langle i, j\rangle^{\mathscr{Y}}$

  $\langle i, j\rangle^{\mathscr{Y}} \to \langle i, j\rangle^{\mathscr{K}}$ *and* $\langle i, j\rangle^{\mathscr{K}} \to \langle i, j\rangle^{\mathscr{Y}}$

  In this event the lever changes from being a tangent to $\mathscr{H}_j$ to touching $\mathscr{H}_j$ with its right endpoint, or the other way round. The corresponding equation for this event is

  $$c_j(o_i(t)) = t + \mu(P)/2,$$

  which can be transformed into a cubic equation in t.

  Since there can be at most three real solutions to that equation, it is possible that this event takes place at most three times for some fixed i and j. On each occurrence of the event, the lever switches between being a tangent and touching the plot with its right endpoint.

- $\langle i, j\rangle^{\mathscr{K}} \to \langle i, j\rangle^{\mathscr{V}}$

  $\langle i, j\rangle^{\mathscr{K}} \to \langle i, j\rangle^{\mathscr{V}}$

  This event happens when the point in which the lever is touching $\mathscr{H}_j$ reaches e_j. Here, the lever is tangent to $\mathscr{H}_j$, and since it must have a positive slope, this will only happen if $e_j > m_j$. The equation for the event is

  $$c_j(o_i(t)) = e_j,$$

  which solves to

  $$o_i(t) = \frac{d_j^2 - e_j m_j + m_j^2}{m_j - e_j}.$$

Event types leading to $\langle i-1, j\rangle$ phase:

- $\langle i, j\rangle^x \to \langle i-1, j\rangle^x$, where $x \in \{\mathscr{Y}, \mathscr{K}, \mathscr{V}\}$

  $\langle i, j\rangle^x \to \langle i-1, j\rangle^x$

  This is the event when the interval to which t belongs changes from $[e_i, e_{i+1})$ to $[e_{i-1}, e_i)$, so this event happens at e_i,

  $$t = e_i.$$

Event types leading to $\langle i, j-1\rangle$ phase:

$\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j-1\rangle^{\mathscr{Y}}$

- $\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j-1\rangle^{\mathscr{Y}}$

  Here, the right endpoint of the lever slides continuously from one hyperbola segment to another,

$$t = e_j - \mu(P)/2.$$

$\langle i,j\rangle^{\mathscr{V}} \to \langle i,j-1\rangle^{\mathscr{Y}}$

- $\langle i,j\rangle^{\mathscr{V}} \to \langle i,j-1\rangle^{\mathscr{Y}}$

  This event happens when the lever stops touching the tip of the wedge and starts to slide its right endpoint over the hyperbola segment on the left of the wedge,

$$t = e_j - \mu(P)/2.$$

$\langle i,j\rangle^{\mathscr{V}} \to \langle i,j-1\rangle^{\mathscr{K}}$

- $\langle i,j\rangle^{\mathscr{V}} \to \langle i,j-1\rangle^{\mathscr{K}}$

  This event happens when the lever stops touching the tip of the wedge and becomes a tangent of the hyperbola segment on the left of the wedge. This can only happen if $e_j > m_{j-1}$,

$$c_{j-1}(o_i(t)) = e_j,$$

  which solves to

$$o_i(t) = \frac{d_{j-1}^2 - e_j m_{j-1} + m_{j-1}^2}{m_{j-1} - e_j}.$$

Event types leading to $\langle i, j_m\rangle$ phase:

$\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j_m\rangle^{\mathscr{K}}$

- $\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j_m\rangle^{\mathscr{K}}$

  This event happens when the lever state changes from having an endpoint on $\mathscr{H}_j$ to being a tangent to $\mathscr{H}_{j_m}$. The corresponding equation is

$$s_{j_m}(o_i(t)) = \frac{h_j(t+\mu(P)/2)}{h_i(t)+\mu(P)/2},$$

  which further transforms into a polynomial equation in t.

  The line through $o_i(t)$ with the slope $s_{j_m}(o_i(t))$ touches the hyperbola h_{j_m}, but we need to be sure that it actually touches the segment $\mathscr{H}_{j_m}$ of that hyperbola. It may as well be the case that $\mathscr{H}_{j_m}$ is not wide enough to have a common point with the line. More precisely, the first coordinate, u, of the touching point between the line and h_{j_m} must belong to the interval $[e_{j_m}, e_{j_m+1})$. To get that coordinate, we solve the equation

$$h'_{j_m}(u) = s_{j_m}(o_i(t)).$$

Having in mind that $o_i(t) < m_{j_m} < u$ must hold, we get a single solution

$$u = m_{j_m} + \frac{d_{j_m}^2}{m_{j_m} - o_i(t)}.$$

If $u \notin [e_{j_m}, e_{j_m+1})$, we do not consider this event.

Checking if the line through $o_i(t)$ with the slope $s_{j_m}(o_i(t))$ actually touches the hyperbola segment $\mathscr{H}_{j_m}$ will also be used in the calculation for the following event types, where we will refer to it by the name *collision check*.

collision check

- $\langle i, j\rangle^{\mathscr{K}} \to \langle i, j_m\rangle^{\mathscr{K}}$

$\langle i, j\rangle^{\mathscr{K}} \to \langle i, j_m\rangle^{\mathscr{K}}$

This event happens when the lever becomes a tangent to two hyperbola segments, $\mathscr{H}_j$ and $\mathscr{H}_{j_m}$ simultaneously. It can only happen if $\mathscr{H}_{j_m}$ is lower than $\mathscr{H}_j$, i.e., $d_{j_m} < d_j$,

$$s_{j_m}(o_i(t)) = s_j(o_i(t)).$$

Since $o_i(t) < m_{j_m}$ and $o_i(t) < m_j$, the only solution is

$$o_i(t) = \frac{d_j m_{j_m} - d_{j_m} m_j}{d_j - d_{j_m}}.$$

Here we need to apply the collision check described earlier to see if the common tangent actually touches $\mathscr{H}_{j_m}$. If the check fails, we do not consider this event.

- $\langle i, j\rangle^{\mathscr{V}} \to \langle i, j_m\rangle^{\mathscr{K}}$

$\langle i, j\rangle^{\mathscr{V}} \to \langle i, j_m\rangle^{\mathscr{K}}$

This event happens when the lever stops touching the tip of the wedge and becomes a tangent of the hyperbola segment $\mathscr{H}_{j_m}$,

$$s_{j_m}(o_i(t)) = \frac{h_j(e_j)}{e_j - o_i(t)}.$$

This can only happen if $h_j(e_j) > d_{j_m}$. From that we get a quadratic equation in $o_i(t)$. The two solutions correspond to the two tangents to h_{j_m} from the point $(e_j, h(e_j))$, so we consider only the smaller solution, which corresponds to the left tangent. Once again we perform the collision check to see if the tangent actually touches $\mathscr{H}_{j_m}$, otherwise we disregard this event.

$\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j_m\rangle^{\mathscr{V}}$

- $\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j_m\rangle^{\mathscr{V}}$

  The event when the lever state changes from having an end-point on $\mathscr{H}_j$ to touching the wedge between $\mathscr{H}_{j_{m-1}}$ and $\mathscr{H}_{j_m}$ is described by

$$\frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)} = \frac{h_j(t+\mu(P)/2)}{h_i(t)+\mu(P)/2},$$

  which again transforms into a polynomial equation in t.

$\langle i,j\rangle^{\mathscr{K}} \to \langle i,j_m\rangle^{\mathscr{V}}$

- $\langle i,j\rangle^{\mathscr{K}} \to \langle i,j_m\rangle^{\mathscr{V}}$

  The event in which the lever touches the wedge tip at e_{j_m} while being a tangent to h_j is represented by the following equation,

$$\frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)} = s_j(o_i(t)).$$

  This can only happen if $h_{j_m}(e_{j_m}) < d_j$. It can be transformed into a quadratic equation in $o_i(t)$. The two solutions correspond to the two tangents to h_j from the point $(e_{j_m}, h(e_{j_m}))$. The smaller of the two solutions is where this event happens.

$\langle i,j\rangle^{\mathscr{V}} \to \langle i,j_m\rangle^{\mathscr{V}}$

- $\langle i,j\rangle^{\mathscr{V}} \to \langle i,j_m\rangle^{\mathscr{V}}$

  This event happens when the lever touches two wedges, at points E_{j_m} and E_j simultaneously. This condition can be written as

$$\frac{h_j(e_j)}{e_j - o_i(t)} = \frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)},$$

  which is a linear equation in $o_i(t)$.

3.4.5 Sequence of realized states

We want to efficiently find the sequence of states through which the lever will pass on its leftward journey, together with the positions where these state changes happen. Let the obtained sequence be $p_1, \mathscr{S}_1, p_2, \mathscr{S}_2, p_3, \ldots, p_r, \mathscr{S}_r$, where $p_1 \le p_2 \le \ldots \le p_r$. Each state

$\mathscr{S}_k$ occurs when the lever position is exactly between p_k and p_{k+1}, where $p_{r+1} = p_1 + \mu(P)$. We call this sequence the *sequence of realized states*.

sequence of realized states

To calculate the sequence of realized states, we will start from a specific lever position that has a known state. Let p_{low} be any of the values for which h attains its global minimum, and let $\mathscr{H}_{j_0}$ be the hyperbola segment above it. The algorithm starts with the lever in the position $t_c = t_0 = p_{low} - \mu(P)/2$. This lever has its right endpoint on the plot at the point $H(p_{low})$, which means that its state is $\langle i_0, j_0 \rangle^{\mathscr{Y}}$, where i_0 is the index of the hyperbola segment over t_0. We note that p_{low} must also be a retargeting position, so we also know our initial jump destination.

p_{low}

The algorithm then iterates with the following operations in its main loop. It first calculates all possible events that could happen while in the current state. Among those events let E be the one with the largest t that is not larger than t_c. It is the event that must occur next. The algorithm sets t_c to t, and it either updates jump destination if E is a jump destination change event, or switches to the new state if the event is a state transition event. In the latter case, the position t and the new state are added to the sequence of realized states. These operations are repeated until one full period of the plot is swept, ending with $t_c = t_0 - \mu(P)$ in the state $\langle i_0 - n, j_0 - n \rangle^{\mathscr{Y}}$. The procedure described is given in Algorithm 4.

Theorem 65. SLIDING LEVER ALGORITHM *finds the sequence of realized states in $O(n)$ time. The length of the produced sequence is $O(n)$.*

Proof. During the execution of the algorithm we must encounter all realized states, since states can change only at events and by always choosing the first following possible event to happen, we eventually consider all realized events. Realized states are encountered in order, since t_c is never increasing.

While choosing the following event we did not consider the possibility that there can be several events with the same minimal t. However, if that happens we can choose an arbitrary one to be the next event. This choice can influence the output sequence only by including or excluding some states of the length zero. Importantly, such zero-length states are irrelevant for further considerations, and no other state in the output of

Algorithm 4 Sliding Lever Algorithm

```
Input: Description of a sequence of hyperbola segments.
Output: The sequence of realized states.
  Find $p_{low}$ and $j_0$.
  $t_0 \leftarrow p_{low} - \mu(P)/2$
  Find $i_0$.
  Run RETARGETING POINTS to find retargeting points and their jump destinations.
  $t_c \leftarrow t_0$
  $i \leftarrow i_0$
  $j \leftarrow j_0$
  $j_m \leftarrow$ jump destination of $p_{low}$
  Set the current state to $\langle i_0, j_0 \rangle^{\mathcal{Y}}$.
  Add $t_0$ and $\langle i_0, j_0 \rangle^{\mathcal{Y}}$ to the output sequence.
  while $t_c > t_0 - \mu(P)$ do
      Calculate all the events for the current state. Ignore jumping events if $j = j_m$.
      Let $E$ be the first event to happen (the one with the largest $t$ not larger than $t_c$).
      $t_c \leftarrow t$ of the event $E$.
      if $E$ is jump destination change event then
          Update $j_m$.
      else
          Set the current state to the destination state of $E$.
          Add $t$ and the current state to the output sequence.
          if $E$ is a jumping event then
              $j_m \leftarrow j$
```

the algorithm is influenced by this choice.

Each event is either a jump destination change event or a state transition event. From Theorem 64 we have that there are $O(n)$ jump destination change events, and now we will show that there are $O(n)$ state transition events as well.

Each state transition event transitioning from some $\langle i, j \rangle$ state decreases either i, or j or both. The only exception are the events $\langle i, j \rangle^{\mathcal{X}} \to \langle i, j \rangle^{\mathcal{Y}}$ and $\langle i, j \rangle^{\mathcal{Y}} \to \langle i, j \rangle^{\mathcal{X}}$, however those events can happen at most three times in total for the same i and j. Note that $j_m \le j$, but when $j_m = j$, we do not consider events involving j_m. Variables i and j start with values i_0 and j_0, and, after the loop finishes, they are decreased to $i_0 - n$ and $j_0 - n$. Hence, no more than $O(n)$ state transition events occurred, implying the linear length of the sequence of realized states.

Calculation of each state transition event takes a constant time, at each iteration a constant number of potential events is considered, and loop is iterated $O(n)$ times. To find the next jump destination change event, we move through the sorted list of retargeting positions until we encounter the first retargeting position not greater than t_c. The total time for calculating jump destination change events, over all iterations, is linear. Therefore, the running time of the whole algorithm is also linear. □

3.5 Merging the two dilations

In this section we will show how to use output of the sliding lever algorithm to give an answer to our original question. The algorithm is run once for the left and once the right dilation, and the obtained sequences of realized states are merged into a single sequence. Finally, we will explain how to find the overall minimum dilation and the optimal feed-link by iterating through the merged sequence.

Knowing the sequence of realized states is sufficient to determine the exact lever slope at any position. Remember, the lever slope at position t is the inverse of the left dilation via $P(t)$, as shown in (3.3). But, to know the dilation via some point we need both the left and the right dilations via that point (3.1).

Our sliding lever algorithm was initially designed only for the left dilation, but an analogous algorithm can obviously be designed for the right dilation (or, we can perform the *exact* same algorithm for the left dilation on the mirror image of the polygon P, and then transform obtained results appropriately). This implies the concept of the right dilation lever for t (as opposed to the left dilation lever, or just the lever, as we have been calling it until now), which has negative slope and touches the plot on the left side of t. We will use + and − in superscript denoting relation with the left and the right dilation, respectively.

Let $p_1^+, \mathscr{S}_1^+, p_2^+, \mathscr{S}_2^+, \ldots, p_{r^+}^+, \mathscr{S}_{r^+}^+$ and $p_1^-, \mathscr{S}_1^-, p_2^-, \mathscr{S}_2^-, \ldots, p_{r^-}^-, \mathscr{S}_{r^-}^-$ be the sequences of realized states for the left and the right dilation, respectively, where both sequences p^+ and p^- are in nondecreasing order. For simplicity, let us call them the *left* and the *right* sequence, respectively. States for right dilation are described by $\langle i, j\rangle$ notation as well, with the meaning analogous to the meaning of the notation

left sequence
right sequence

for the left dilation states. We say that the (right dilation) lever for t is in the phase $\langle i, j\rangle$, when $\mathscr{H}_i$ is the hyperbola segment above t, and the (right dilation) lever touches the hyperbola segment $\mathscr{H}_j$.

We now merge the two obtained sequences by overlapping them into a new sequence $p_1, \mathscr{S}_1, p_2, \mathscr{S}_2, p_3, \ldots, p_r, \mathscr{S}_r$. In the merged sequence, $p_1 \leq p_2 \leq \ldots \leq p_r$ is the sorted union of $\{p_1^+, p_2^+, \ldots, p_{r^+}^+\}$ and $\{p_1^-, p_2^-, \ldots, p_{r^-}^-\}$. States in the merged sequence are pairs consisting of one state from the left sequence and one state from the right sequence. Each state $S_k = (\mathscr{S}_{k^+}^+, \mathscr{S}_{k^-}^-)$ in the merged sequence is such that $\mathscr{S}_{k^+}^+$ and $\mathscr{S}_{k^-}^-$ are states covering the interval between p_k and p_{k+1} in the left and in the right sequence, respectively.

By Theorem 65, both r^+ and r^- are $O(n)$, so the length of the merged sequence is also linear in n. Because the left and the right sequences are sorted, the merged sequence can be computed in $O(n)$ time.

For each state $\mathscr{S}_k = (\mathscr{S}_{k^+}^+, \mathscr{S}_{k^-}^-)$ there is a single expression for computing the lever slope as a function of t, when $p_k \leq t \leq p_{k+1}$, both for the left and the right dilation. To find the minimal dilation while in that state, we want to find t which maximizes the minimum of the two slopes for the left and the right lever. This observation readily follows from (3.1) and (3.3), so

$$\min_{p_k \leq t \leq p_{k+1}} \tilde{\delta}_{P(t)} = \frac{1}{\max_{p_k \leq t \leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\}}, \tag{3.7}$$

where $\tilde{s}^+(t)$ is the slope for the left dilation lever for t, and $\tilde{s}^-(t)$ is the slope for the right dilation lever for t.

This means that to get the final value of the optimal dilation we essentially need to compute the minimum of the upper envelope of two functions, which is a standard procedure that can be done efficiently. In the following, we take a closer look at the computations needed to complete this step in our setting.

For all possible combinations of the left and the right state types in a combined state, we show how to find the maximum of the lower envelope of the slope functions $\tilde{s}^+(t)$ and $\tilde{s}^-(t)$ by analyzing the shape of those functions. The following lemmas help us describe them.

Let us assume that the corresponding state to which t belongs is $\mathscr{S} = (\mathscr{S}^+, \mathscr{S}^-)$, and let $s^+_{\langle i,j\rangle^{\mathscr{K}}}(t)$ be a function which maps t to the slope of a lever, assuming that the lever is in $\langle i, j\rangle^{\mathscr{K}}$ state. Analogously we define $s^+_{\langle i,j\rangle^{\mathscr{V}}}(t)$, $s^+_{\langle i,j\rangle^{\mathscr{Y}}}(t)$, $s^-_{\langle i,j\rangle^{\mathscr{K}}}(t)$, $s^-_{\langle i,j\rangle^{\mathscr{V}}}(t)$ and

$s^-_{\langle i,j\rangle^{\mathscr{Y}}}(t)$.

Lemma 66. *If $\mathscr{S}^+$ is a $\langle i,j\rangle^{\mathscr{X}}$ state, then $\tilde{s}^+(t)$ is a monotonically nondecreasing function.*

Proof. If $\mathscr{S}^+$ is an $\langle i,j\rangle^{\mathscr{X}}$ state, then, from equation (3.6), we have

$$\begin{aligned}\tilde{s}^+(t) = s^+_{\langle i,j\rangle^{\mathscr{X}}}(t) = s_j(o_i(t)) &= \frac{h_j(c_j(o_i(t)))}{c_j(o_i(t)) - o_i(t)}\\ &= 1/\sqrt{\left(\frac{m_j - o_i(t)}{d_j}\right)^2 + 1}.\end{aligned}$$

We see that the function s_j is monotonically increasing for parameter values less than m_j. In the specified state, $o_i(t) < m_j$ holds, and since $o(t)$ is monotonically nondecreasing (Lemma 63), it means that $\tilde{s}^+(t)$, being the composition of s_j and $o(t)$, is monotonically nondecreasing as well. □

Lemma 67. *If $\mathscr{S}^+$ is a $\langle i,j\rangle^{\mathscr{V}}$ state, then $\tilde{s}^+(t)$ is a monotonically nondecreasing function.*

Proof. If $\mathscr{S}^+$ is an $\langle i,j\rangle^{\mathscr{V}}$ state, then we have

$$\tilde{s}^+(t) = s^+_{\langle i,j\rangle^{\mathscr{V}}}(t) = \frac{h_j(e_j)}{e_j - o_i(t)}.$$

We see that $\tilde{s}^+(t)$ is monotonically increasing in terms of $o_i(t)$ when $o_i(t) < e_j$, which holds in the specified state. Since $o(t)$ is monotonically nondecreasing (Lemma 63), it means that $\tilde{s}^+(t)$ is monotonically nondecreasing in terms of t as well. □

Similar observations hold for the right dilation analogues: $\tilde{s}^-(t)$ is monotonically nonincreasing if $\mathscr{S}^-(t)$ is $\langle i,j\rangle^{\mathscr{X}}$ or $\langle i,j\rangle^{\mathscr{V}}$ state.

Lemma 68. *If $\mathscr{S}^-$ is a $\langle i,j\rangle^{\mathscr{Y}}$ state then $\tilde{s}^+(t) \leq \tilde{s}^-(t)$.*

Proof. From equation (3.3), using the fact that $h_j(t) = h_{j+n}(t+\mu(P))$ holds because of the periodicity, we have

$$\begin{aligned}\tilde{s}^-(t) &= \frac{h_j(t-\mu(P)/2)}{h_i(t)+\mu(P)/2}\\ &= \frac{h_{j+n}(t+\mu(P)/2)}{h_i(t)+\mu(P)/2}\\ &= s(t, t+\mu(P)/2)\\ &\geq \min_{t_r\in[t,t+\mu(P)/2]} s(t,t_r)\\ &= \tilde{s}^+(t).\end{aligned}$$

□

Analogously, if $\mathscr{S}^+$ is a $\langle i,j\rangle^{\mathscr{Y}}$ state then $\tilde{s}^-(t) \leq \tilde{s}^+(t)$.

For equation (3.7) we need to calculate $\max_{p_k\leq t\leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\}$. This calculation depends on the types of the states $\mathscr{S}^+$ and $\mathscr{S}^-$ in the specified interval. We analyze nine possible type combinations.

- If $\mathscr{S}^+$ is $\langle i,j^+\rangle^{\mathscr{Y}}$ state and $\mathscr{S}^-$ is $\langle i,j^-\rangle^{\mathscr{Y}}$ state:

  Using Lemma 68 we get $\tilde{s}^+(t) = \tilde{s}^-(t)$, so

  $$\max_{p_k\leq t\leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\} = \max_{p_k\leq t\leq p_{k+1}} s^+_{\langle i,j^+\rangle^{\mathscr{Y}}}(t).$$

  The maximum is achieved either at interval ends or at a local maxima, if one exists, which is found by solving a polynomial equation.

- If $\mathscr{S}^+$ is $\langle i,j^+\rangle^{\mathscr{X}}$ state and $\mathscr{S}^-$ is $\langle i,j^-\rangle^{\mathscr{Y}}$ state:

  Using Lemma 68 and Lemma 66 we get

  $$\max_{p_k\leq t\leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\} = \max_{p_k\leq t\leq p_{k+1}} s^+_{\langle i,j^+\rangle^{\mathscr{X}}}(t) = s^+_{\langle i,j^+\rangle^{\mathscr{X}}}(p_{k+1}).$$

- If $\mathscr{S}^+$ is $\langle i,j^+\rangle^{\mathscr{X}}$ state and $\mathscr{S}^-$ is $\langle i,j^-\rangle^{\mathscr{X}}$ state:

  From Lemma 66 we know that $s^+_{\langle i,j^+\rangle^{\mathscr{X}}}(t)$ is monotonically nondecreasing, and $s^-_{\langle i,j^-\rangle^{\mathscr{X}}}(t)$ is monotonically nonincreasing. The highest point of the lower envelope of these functions on $[p_k, p_{k+1}]$ is thus located either at one end of the interval, or at the point of the intersection of the two functions, which can be found by solving a polynomial equation.

The other six combinations of state types are not listed, but each of them is resolved in a manner very similar to the one of the above three combinations. Cases with $\langle\cdot,\cdot\rangle^{\mathscr{V}}$ states are resolved analogously to the cases having $\langle\cdot,\cdot\rangle^{\mathscr{H}}$ states instead by using Lemma 67 in place of Lemma 66, and the remaining cases are analogous to the cases having "pluses" and "minuses" swapped.

Finally, by taking the smallest of all dilation minima from $[p_k, p_{k+1}]$ intervals for $k \in \{1, 2, \dots r\}$ we obtain the overall minimum dilation,

$$\delta = \min_{k \in \{1,2,\dots,r\}} \min_{p_k \leq t \leq p_{k+1}} \tilde{\delta}_{P(t)}.$$

While going through calculated interval minima we maintain the value of t for which the minimum is achieved, so we also get the point on P which is the endpoint of the optimal feed-link.

3.6 Conclusion and further work

The problem we considered asked for the optimal extension of a polygonal network by connecting a specified point to the rest of the network via a feed-link. We gave a linear time algorithm for solving this problem, thus improving upon the previously best known result of Aronov et al. presented in [14].

On the way to solution, we performed several steps. First, we divided the concept of dilation into the left and the right dilation, so that the two can be analyzed separately. Then we transformed them into the problem which considers the plot of the distance function and lever slopes, an gave an algorithm for event based simulation of the lever movement. The output of the algorithm is the description of the changes in the lever slope presented as a sequence of states, each of which can be expressed analytically. Finally, we explained how those state sequences for the left and the right dilation can be merged and how the optimal feed-link can be found from it.

We note that the method we used for solving the original problem can be easily adapted to work with any network shaped as an open polygonal chain.

Aronov et al. in [14] discuss polygons with obstacles. They show how polygonal obstacles with total number of vertices equal to b induce a partition of the polygon boundary of the size $O(nb)$, and how that partition can be computed in $O(nb + b \log b)$ time. Each segment of the partition has a distance function to p similar to function $h_i(t)$, the only difference being an additive constant.

In our setting, this makes the hyperbola segments in the plot to shift upward by that constant, which means that the problem with obstacles can also be handled by our approach. This variation would require a more elaborate case study from Section 3.4.4. The equations in respective cases would also change, but they would still remain polynomial and thus efficiently solvable. We will not pursue the details here.

A natural way to generalize the problem is to assume that the polygon edges are not line segments, but curves (second order curves, for example). The abstraction behind our method can be applied in this case provided that there is an efficient way to determine the event times and to find optimal values in the merged sequence.

Another obvious generalization is asking for a set of $k > 1$ feed-links that minimize the dilation. Aronov et al. in [14] give an approximate algorithm for computing such a set. However, finding an efficient algorithm that solves this problem exactly is still open. One could try to apply our approach here, but we do not see that this can be done in a straightforward fashion.

It would also be interesting to see whether a similar method can be applied to a network which is not necessarily polygonal, i.e., to a network whose vertices can have degree greater than two.

Prošireni izvod (in Serbian)

Monohromatski bottleneck mečinzi

Neka je P skup od n tačaka u ravni, gde je n paran broj. Neka je M savršeni mečing tačaka u P koji se dobija povezivanjem tačaka koristeći $n/2$ duži koje se ne presecaju. Najdužu duž u M označavamo sa $bn(M)$, i tu vrednost nazivamo *vrednost mečinga* mečinga M. Naš cilj je pronaći mečing za koji je $bn(M)$ minimalno. Svaki takav mečing se naziva *bottleneck mečing* skupa tačaka P.

vrednost

bottleneck mečing

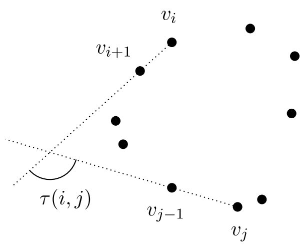
Rezultati

U onome što sledi posmatramo slučaj gde su sve tačke iz P u konveksnom položaju, to jest, predstavljaju temena konveksnog poligona $\mathscr{P}$. Tačke su monohromatske, što znači da svake dve mogu biti uparene u mečingu. Obzirom na to da ćemo raditi isključivo sa mečinzima bez presecanja, ubuduće kada koristimo pojam mečing mislićemo isključivo na mečinge u kojima se duži ne seku.

Označimo tačke iz P sa $v_0, v_1, \ldots, v_{n-1}$, u pozitivnom smeru (smer suprotan kazaljki na satu). Radi jednostavnosti notacije, često ćemo pisati samo indekse kada govorimo o tačkama. Sa $\{i, \ldots, j\}$ ćemo označavati niz $i, i+1, i+2, \ldots, j-1, j$, gde se sve operacije računaju po modulu n. Primetimo da i nije obavezno manje od j, i da $\{i, \ldots, j\}$ nije isto što i $\{j, \ldots, i\}$. Kažemo da je (i, j) *dopustiv par* ako postoji mečing koji sadrži (i, j), što u monohromatskom slučaju jednostavno znači da je $\{i, \ldots, j\}$ parne veličine.

dopustiv par

Problem nalaženja bottleneck mečinga tačaka u konveksnom položaju može biti rešen u polinomijalnom vremenu koristeći dinamičko programiranje, kao što je prezentovano u [5].



Slika 44. Ugao skretanja.

U ovoj tezi prezentujemo brži algoritam za nalaženje bottleneck mečinga za monohramatski skup tačaka u konveksnom položaju, sa vremenskom složenošću od svega $O(n^2)$. Da bismo konstruisali taj algoritam, dokazujemo niz rezultata koji daju uvid u osobine i strukturu bottleneck mečinga. Ovaj rezultat je objavljen u [43].

Struktura bottleneck mečinga

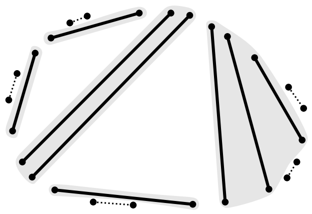
Podelimo sve parove tačaka u dve kategorije. Parovi koji se sastoje od dve susedne tačke se nazivaju *ivice*, dok se svi drugi parovi nazivaju *dijagonale*. Svaki mečing se, stoga, sastoji od ivica i dijagonala.

ivice

dijagonale

Ugao skretanja

Ugao skretanja niza $\{i, \ldots, j\}$, u oznaci $\tau(i, j)$, predstavlja ugao za koji je potrebno rotirati vektor $\overrightarrow{v_i v_{i+1}}$ u pozitivnom smeru da bi se poklopio sa vektorom $\overrightarrow{v_{j-1} v_j}$, videti Sliku 44.

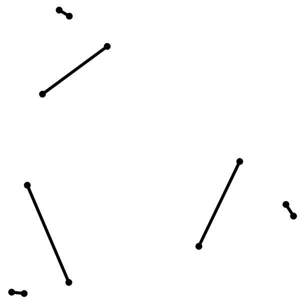

Slika 45. Dijagonale unutar svake osenčene oblasti čine jednu kaskadu. Postoje tri kaskade sa jednom dijagonalom, jedna kaskada sa dve dijagonale, i jedna kaskada sa tri dijagonale.

Lema 69. *Postoji bottleneck mečing M skupa P u kome za sve dijagonale $(i,j) \in M$ važi $\tau(i,j) > \pi/2$.*

Posmatrajmo podelu unutrašnjosti poligona $\mathscr{P}$ na regione koji se dobijaju tako što poligon presečemo dijagonalima datog mečinga M. Svaki region dobijen ovom podelom je ograničen nekim dijagonalama mečinga M i obodom poligona $\mathscr{P}$. Ako tačno k dijagonala ograničavaju region, kažemo da je taj region *k-ograničen*. Svaki maksimalni niz dijagonala povezan 2-ograničenim regionima se naziva *kaskada*,

kaskada

Lema 70. *Postoji bottleneck mečing koji ima najviše tri kaskade.*


Slika 46. Konfiguracija tačaka za koju jedini bottlenech mečing ima tačno tri kaskade.

Slučaj bottleneck mečinga koji ima tačno tri kaskade je moguć, kao što je prikazano na slici 46. Lako se vidi da mečing ne može imati tačno dve kaskade.

Nalaženje bottleneck mečinga

Mečinzi sa najviše jednom kaskadom

Kada govorimo o mečinzima sa minimalnom vrednošću pod nekim ograničenjima, nazivaćemo ih *optimalni mečinzi*.

optimalni mečinzi

Za neparno $j-i$, neka je MATCHING(i,j) problem nalaženja optimalnog mečinga $M_{i,j}$ tačaka $\{i,\ldots,j\}$, takvog da $M_{i,j}$ ima najviše jednu kaskadu, pri čemu duž (i,j) pripada regionu ograničenom najviše jednom dijagonalom iz $M_{i,j}$ različitom od (i,j).

Ovi problemi imaju optimalnu podstrukturu, pa možemo primeniti dinamičko programiranje da ih rešimo. Rešenje problema $bn(M_{i,j})$ izražavamo rekurentnom formulom i čuvamo ga u $S[i,j]$.

Ukupno imamo $O(n^2)$ potproblema od kojih svaki zahteva $O(1)$ vreme za izračunavanje. Stoga izračunavanje svih potproblema zajedno zahteva $O(n^2)$ vremena i istu tu količinu memorije. Optimalni mečing sa samo jednom kaskadom se nalazi tražeći minimum među svim $S[i+1,i]$i i uzimajući $M_{i+1,i}$ za koji se taj minimum postiže. Ovaj korak zahteva samo linearno vreme.

neophodan par

Biće korisno utvrditi i da li je par (i, j) neophodan za konstruisanje $M_{i,j}$, odnosno da li sva rešenja za MATCHING(i, j) sadrže (i, j). Ako to važi, onda takav orijentisan par (i, j) zovemo *neophodan par*. Računanje vrednosti potproblema se lako proširuje tako da se dobiju i svi neophodni parovi, ne menjajući složenost celog postupka.

Mečinzi sa tri kaskade

Tri različite tačke i, j i k, takve da je $j \in \{i+1, \dots, k-1\}$, gde su (i, j), $(j+1, k)$ i $(k+1, i-1)$ dopustivi parovi, mogu da se iskoriste za konstruisanje mečinga sa tri kaskade uzimajući uniju $M_{i,j}$, $M_{j+1,k}$ i $M_{k+1,i-1}$. Nalaženje optimalnog mečinga bi se moglo izvesti prolaskom kroz sve trojke (i, j, k) i uočavanjem one koje minimizira $\max\{S[i, j], S[j+1, k], S[k+1, i-1]\}$. Međutim, ovo zahteva $O(n^3)$ vreme. Pokazaćemo da je umesto da posmatramo sve (i, j) dovoljno da biramo (i, j) iz skupa linearne veličine, što rezultuje pretragom koja uzima svega $O(n^2)$ vreme.

unutrašnje dijagonale

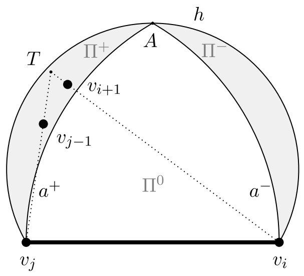
U mečingu sa tri kaskade sa *unutrašnjim dijagonalama* označavamo tri orijentisane dijagonale koje sa svoje leve strane ograničavaju jedinstveni 3-ograničeni region.

Lema 71. *Ako ne postoji bottleneck mečing sa najviše jednom kaskadom, tada postoji bottleneck mečing sa tri kaskade čija je svaka unutrašnja dijagonala neophodna.*

kandidat dijagonala

Orijentisanu dijagonalu (i, j) nazivamo *kandidat dijagonalom*, ako je $neophodan(i, j)$ i $\tau(i, j) \leq 2\pi/3$.

Lema 72. *Ako ne postoji bottleneck mečing sa najviše jednom kaskadom, tada postoji bottleneck mečing sa tri kaskade čija je bar jedna unutrašnja dijagonala kandidat dijagonala.*



Slika 47. Sve tačke $v_{i+1}, \dots, v_{j-1}$ leže ili unutar Π^- ili unutar Π^+.

Sledeći zaključak je u vezi sa pozicijama tačaka $\{i+1, \dots, j-1\}$ u odnosu na kandidat dijagonalu (i, j). Konstruišemo kružni luk h sa desne strane usmerene linije $v_i v_j$, sa koga se ta linija vidi pod uglom $\pi/3$, videti Sliku 47. Sredinu h označimo sa A. Tačke v_i, A i v_j čine jednakostraničan trougao, pa možemo konstruisati luk a^- između A i v_i sa centrom u v_j, i luk a^+ između A i v_j sa centrom u v_i. Ovi lukovi definišu tri oblasti: Π^-, ograničenu sa h i a^-, Π^+, ograničenu sa h i a^+, i Π^0, ograničenu sa a^-, a^+ i duži $v_i v_j$, sve kao što je prikazano na slici 47.

Lema 73. *Ako je (i, j) kandidat dijagonala, tada tačke $v_{i+1}, \ldots, v_{j-1}$ ili sve pripadaju Π^- ili sve pripadaju Π^+.*

Sa $\Pi^-(i, j)$ i $\Pi^+(i, j)$ respektivno označavamo oblasti Π^- and Π^+ koje odgovaraju kandidat dijagonali (i, j).

Ako tačkes $\{i + 1, \ldots, j - 1\}$ leže u $\Pi^-(i, j)$, kažemo da kandidat dijagonala (i, j) ima *negativnu polarnost* i tačku i za svoj *pol*. U suprotnom, ako ove tačke leže u in $\Pi^+(i, j)$, kažemo da (i, j) ima *pozitivnu polarnost* i tačku j za svoj pol.

negativna polarnost
pol
pozitivna polarnost

Lema 74. *Nijedne dve kandidat dijagonale iste polarnosti ne mogu imati istu tačku za svoj pol.*

Jednostavna posledica Leme 74 je da postoji najivše linearan broj kandidat dijagonala.

Lema 75. *Postoji $O(n)$ kandidat dijagonala.*

Konačno, kombinujući Lemu 72 i Lemu 75 možemo konstruisati Algorithm 5.

Algorithm 5 Bottleneck Mečing

```
Izračunati $S[i, j]$ i $neophodan(i, j)$ za sve dopustive parove $(i, j)$.
$najbolji \leftarrow \min\{S[i + 1, i] : i \in \{0, \ldots, n - 1\}\}$
for sve dopustive parove $(i, j)$ do
    if $neophodan(i, j)$ i $\tau(i, j) \leq 2\pi/3$ then
        for $k \in \{j + 1, \ldots, i - 1\}$ takvo da je $(j + 1, k)$ dopustiv par do
            $najbolji \leftarrow \min\{best, \max\{S[i, j], S[j + 1, k], S[k + 1, i - 1]\}\}$
```

Teorema 76. *Algoritam 5 nalazi vrednost bottleneck mečinga u $O(n^2)$ vremenu.*

Bihromatski bottleneck mečinzi

Za razliku od monohromatske verzije, ovde nam tačke dolaze iz dva skupa i nije dozvoljeno uparivati tačke unutar istog skupa. Preciznije, neka su R i B skupovi od n crvenih i n plavih tačaka u ravni, respektivno, i $P = R \cup B$. Neka je M savršeni bipartitni mečing između skupova R i B, koristeći duži za spajanje tačaka tako da se ni jedne dve duži ne seku. Naš cilj je da nađemo mečing koji pod datim ograničenjima minimizira $\mathrm{bn}(M)$, tj. bihromatski bottleneck mečing.

Rezultati

U cilju jednostavnijeg rada sa bihromatskim nepresecajućim mečinzima konveksnog skupa tačaka, uvodimo strukturu koju nazivamo *orbita*, za koju se ispostavlja da dobro predstavlja neke strukturalne osobine ovakvih mečinga. Dolazimo do brojnih osobina orbita koje, u kombinaciji sa idejama za monohromatske mečinge, nam omogućavaju da efikasno rešimo bihromatske probleme.

Konstruišemo algoritam koji rešava problem bihromatskog nepresecajućeg mečinga tačaka u konveksnom položaju u $O(n^2)$ vremenu, poboljšavajući prethodno najbolji poznati algoritam $O(n^3)$ vremenske složenosti, predstavljen u [20] i [22]. Takođe, koristeći ovaj skup alata, dajemo optimalan $O(n)$ algoritam za slučaj kada tačke leže na krugu, za koji prethodno najbolji poznati algoritam ima $O(n \log n)$-vremensku složenost

Orbite i njihove osobine

balansiran, plavo-težak, crveno-težak

Definicija 77. Skup tačaka je *balansiran* ako sadrži isti broj crvenih i plavih tačaka. Ako skup ima više crvenih tačaka nego plavih, kažemo da je *crveno-težak*, a ako ima više plavih nego crvenih, kažemo da je *plavo-težak*.

Lema 78. *Nad svakim balansiranim skupom tačaka može biti konstruisan mečing.*

Lema 79. *Par (i, j) je dopustiv ako i samo ako su i i j različite boje i $\{i, \ldots, j\}$ je balansiran.*

Funkcije o i o^{-1}

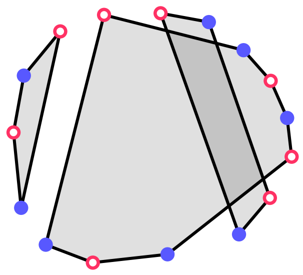
Definicija 80. Sa $o(i)$ označavamo prvu tačku počevši od i u pozitivnom smeru za koju važi da je $(i, o(i))$ dopustiv par. Sa $o^{-1}(i)$ označavamo prvu tačku počevši od i u negativnom smeru za koju važi da je $(o^{-1}(i), i)$ dopustiv par.

Osobina 81. *Funkcija o je bijekcija, i o^{-1} je njena inverzna funkcija.*

Definicija 82. *Orbitu* od i, u oznaci $\mathcal{O}(i)$, definišemo sa $\mathcal{O}(i) := \{o^k(i) : k \in \mathbb{Z}\}$. Sa $\mathcal{O}(P)$ označavamo skup svih orbita konveksnog skupa tačaka P, to jest $\mathcal{O}(P) := \{\mathcal{O}(i) : i \in P\}$.

orbita

Primer balansiranog bihromatskog konveksnog skupa tačaka zajedno sa svojim skupom orbita se može videti na Slici 48. Primetimo da je iz definicije orbita jasno da za svako $j \in \mathcal{O}(i)$ imamo $\mathcal{O}(j) = \mathcal{O}(i)$, i samim tim skup svih orbita, $\mathcal{O}(P)$, je *particija* skupa svih tačaka.



Slika 48. Orbite – primer.

Dalje, dokazujemo niz osobina orbita.

Osobina 83. *Tačke i i j čine dopustiv par ako i samo ako imaju različite boje i $\mathcal{O}(i) = \mathcal{O}(j)$.*

Osobina 84. *Dopustivi par deli tačke* svake *orbite na dva balansirana dela.*

Neformalno govoreći, sledeća osobina nam garantuje da uzastopnom primenom funkcije o pratimo tačke orbite u redosledu u kom se pojavljuju na $\mathcal{P}$, čime posećujemo *sve* tačke orbite u *tačno jednom* obilasku oko poligona.

Osobina 85. *Ni jedna tačka orbite $\mathcal{O}(i)$ ne leži između i i $o(i)$, to jest, $\{i, \ldots, o(i)\} \cap \mathcal{O}(i) = \{i, o(i)\}$.*

Osobina 86. *Svake dve susedne tačke orbite imaju različite boje.*

Osobina 87. *Svaka orbita je balansirana.*

Dalje razmatramo strukturalne osobine dve različite orbite.

Osobina 88. *Neka su i i j tačke iz dve različite orbite takve da se nijedna druga tačka iz njihovih orbita ne nalazi između njih, to jest, $\{i, \ldots, j\} \cap \mathcal{O}(i) = i$ i $\{i, \ldots, j\} \cap \mathcal{O}(j) = j$. Tada, i i j imaju istu boju.*

Prelazeći na algoritamsku stranu priče, pokazujemo da možemo efikasno odredit sve orbite, odnosno preciznije – sve vrednosti funkcije o.

Lema 89. *Funkcija $o(i)$, za svako i, može biti izračunata u $O(n)$ vremenu.*

Definišemo dve kategorije dopustivih parova prema njihovoj relativnoj poziciji u orbiti.

ivica, dijagonala

Definicija 90. Dopustivi par (i, j) je *ivica* ako i samo ako $i = o(j)$ ili $j = o(i)$; u suprotnom taj par nazivamo *diagonal*.

Osobina 91. *Ako je $\{i, \ldots, j\}$ balansirani skup, tada tačke u $\{i, \ldots, j\}$ mogu biti uparene koristeći samo ivice.*

Kada pričamo o ivicama, smatramo ih za uređene parove tačaka, tako da je ivica $(i, o(i))$ usmerena od i ka $o(i)$. Kažemo da su tačke $\{i, \ldots, o(i)\} \setminus \{i, o(i)\}$ sa desne strane ivice $(i, o(i))$, a tačke $\{o(i), \ldots, i\} \setminus \{i, o(i)\}$ sa njene leve strane. Usmerenost ivica i obojenost tačaka zajedno definišu dva moguća tipova ivica.

crveno-plave ivice, plavo-crvene ivice

Definicija 92. Kažebo ia je $(i, o(i))$ *crveno-plava* ivica ako $i \in R$, odnosno *plavo-crvena ivica* ako $i \in B$.

Osobina 93. *Dve grane istog tipa (obe crveno-plave, ili obe plavo-crvene) iz različitih orbita se ne ne seku.*

Osobina 94. *Za svake dve orbite $\mathscr{A}, \mathscr{B} \in \mathcal{O}(P)$, $\mathscr{A} \neq \mathscr{B}$, ili su sve tačke iz $\mathscr{B}$ sa desne strane crveno-plavih ivica iz $\mathscr{A}$, ili su sve tačke iz $\mathscr{B}$ sa desne strane plavo crvenih ivica iz $\mathscr{A}$.*

Sledeća osobina nam govori o uzajamnoj sinhronizovanosti orbita.

Osobina 95. *Neka $\mathscr{A}, \mathscr{B} \in \mathscr{O}(P)$. Ni jedna tačka iz $\mathscr{B}$ se ne nalazi sa desne strane crveno-plave ivice iz $\mathscr{A}$ ako i samo ako ni jedna tačka iz $\mathscr{A}$ se ne nalazi sa desne strane plavo-crvene ivice iz $\mathscr{B}$.*

Definicija 96. Relacija $\preceq$ na skupu svih orbita, $\mathscr{O}(P)$, se definiše sa $\mathscr{A} \preceq \mathscr{B}$ ako i samo ako nema tačaka iz $\mathscr{B}$ sa desne strane crveno-plavih ivica iz $\mathscr{A}$ (što je, prema Osobini 95, ekvivalentno sa nepostojanjem tačaka iz $\mathscr{A}$ sa desne strane plavo-crvenih ivica iz $\mathscr{B}$).

relacija $\preceq$

Osobina 97. *Relacija $\preceq$ na $\mathscr{O}(P)$ je totalni poredak.*

Graf orbita

Definicija 98. *Graf orbita* $\mathscr{G}(P)$ je usmereni graf čiji skup čvorova je skup svih orbita, $\mathscr{O}(P)$, a postoji usmerena grana od orbite $\mathscr{A}$ do orbite $\mathscr{B}$ ako i samo ako se $\mathscr{A}$ i $\mathscr{B}$ seku i važi $\mathscr{A} \preceq \mathscr{B}$.

Graf orbita

Osobina 99. *Svaka slabo povezana komponenta grafa orbita $\mathscr{G}(P)$ sadrži jedinstven Hamiltonov put.*

Lema 100. *Totalni poredak orbita i Hamiltonovi putevi za sve slabo povezane komponente grafa orbita mogu biti nađeni u ukupnom $O(n)$ vremenu.*

Nalaženje bottleneck mečinga

Pristup rešavanju problema bihromatskog bottleneck mečinga je sličan pristupu koji smo koristili za monohromatsku verziju problema. Tvrđenja data u monohromatskom kontekstu važe i ovde, ali koristeći definicije ivica i dijagonala u obliku koji smo dali za bihromatski problem. Dokazi za tvrđenja u bihromatskoj verziji problema se značajno oslanjaju na izvedene osobine orbita i grafa orbita.

Optimalno postavljanje feed-linka

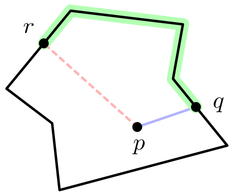
feed-link

Neka su u ravni dati poligon listom svojih temena $v_0, v_1, \dots, v_{n-1}$ i tačka p. Obod poligona označimo sa P. Duž pq koja spaja p sa nekom tačkom iz P nazivamo *feed-link*.

Dilacija tačke r preko q

Definicija 101. Za svake dve tačke $q, r \in P$, *dilacija tačke r preko q* se definiše kao

$$\delta_q(r) = \frac{|pq| + \mathrm{dist}(q,r)}{|pr|},$$



Slika 49. Koncept dilacije.

gde je $\mathrm{dist}(q,r)$ dužina najkraće putanje između q i r preko oboda poligona, a $|ab|$ je Euklidsko rastojanje između tačaka a i b, videti Sliku 49.

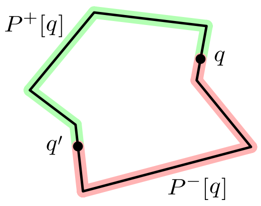
Dilacija preko q

Definicija 102. Za tačku $q \in P$, *dilacija preko q* se definiše kao

$$\tilde{\delta}_q = \max_{r \in P} \delta_q(r).$$

Problem postavljanja optimalnog feed-linka predstavlja određivanje tačke q za koju je $\tilde{\delta}_q$ minimizirano.

Rezultati



Slika 50. Levi i desni deo P posmatrani u odnosu na tačku q.

Za problem optimalnog postavljanja feed-linka dajemo algoritam linearne vremenske složenosti. Prethodno najbolji poznat algoritam za ovaj problem je predstavljen u [14]. Njegova vremenska složenost je $O(\lambda_7(n) \log n)$, gde je $\lambda_7(n)$ funkcija koja raste nešto brže od linearne.

Leva i desna dilacija

$\mu(q,r)$

$\mu(P)$

Za dve date tačke $q, r \in P$, $P[q,r]$ je deo P koji se dobija kada se od q ide ka r oko poligona u pozitivnom smeru, uključujući tačke q i r. Neka je $\mu(q,r)$ dužina od $P[q,r]$, a $\mu(P)$ obim od P.

Za dato $q \in P$, neka je q' tačka na P za koju je njeno mrežno rastojanje od tačno polovina obima, to jest, $\mu(q,q') = \mu(q',q) = \mu(P)/2$, videti Sliku 50. Sa $P^+[q]$ označavamo $P[q,q']$, a sa $P^-[q]$ označavamo $P[q',q]$. Očigledno, $P^+[q] \cup P^-[q] = P$ i $P^+[q] \cap P^-[q] = \{q, q'\}$.

Leva i desna dilacija

Definicija 103. Za datu tačku $q \in P$ i $r \in P^+[q]$, *leva dilacija tačke* *r preko q* se definiše kao

$$\delta_q^+(r) = \frac{|pq| + \mu(q,r)}{|pr|}.$$

Sa druge strane, za $r \in P^-[q]$, *desna dilacija tačke r preko q* se definiše kao

$$\delta_q^-(r) = \frac{|pq| + \mu(r,q)}{|pr|}.$$

Najkraća putanja od q do r leži ili cela u $P^+[q]$ ili cela u $P^-[q]$. Ovo nam omogućava da izrazimo dilaciju tačke r preko q koristeći levu i desnu dilaciju tačke r preko q.

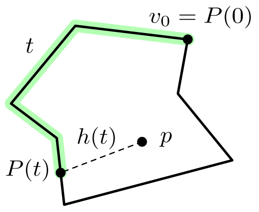
$$\delta_q(r) = \begin{cases} \delta_q^+(r), & \text{if } r \in P^+[q] \\ \delta_q^-(r), & \text{if } r \in P^-[q] \end{cases}.$$

Leva i desna dilacija preko q

Definicija 104. Za datu tačku $q \in P$, *leva dilacija preko q* se definiše kao $\tilde{\delta}_q^+ = \max_{r \in P^+[q]} \delta_q^+(r)$, a *desna dilacija preko q* kao $\tilde{\delta}_q^- = \max_{r \in P^-[q]} \delta_q^-(r)$.

Sada možemo izraziti dilaciju preko q.

$$\tilde{\delta}_q = \max(\tilde{\delta}_q^+, \tilde{\delta}_q^-) = \max_{r \in P} \delta_q(r). \tag{1.8}$$



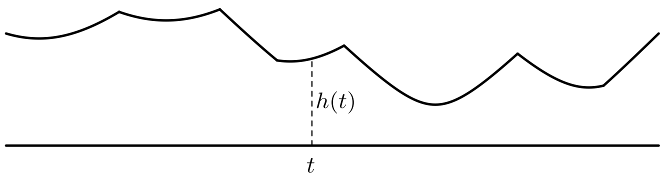
Slika 51. Parametrizacija poligona P.

Dalje ćemo posmatrati samo levu dilaciju, obzirom na to da je problem nalaženja desne dilacije analogan. Radi pojednostavljenja notacije, u narednom delu izostavljamo natpis $+$, i smatramo da radimo sa levom dilacijom.

Drugačiji pogled na problem

$P(t)$

Najpre parametrizujemo tačke na P definišući $P(t)$, $t \in \mathbb{R}$, kao tačku na P za koju je $\mu(v_0, P(t)) \equiv t \pmod{\mu(P)}$, videti Sliku 51.



Slika 52. Grafik funkcije $h(t)$.

Rastojanje tačke od tačaka na pravi je hiperbolična funkcija. Grafik *funkcije udaljenosti* $h(t) := |pP(t)|$ je beskonačan niz hiperboličnih

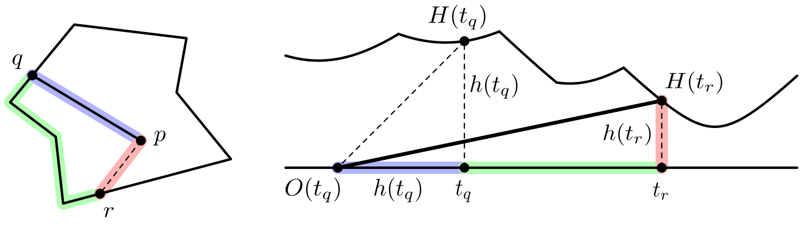
$h(t)$

segmenata spojenih u svojim krajnjim tačkama, gde segment $(kn+r)$ odgovara r-toj stranici poligona P, za $r \in \{0, 1, \dots, n-1\}$ i $k \in \mathbb{Z}$, videti Sliku 52.

Za svako $i = kn + r$, hiperbola h_i je oblika $h_i(t) = \sqrt{(t - m_i)^2 + d_i^2}$, za neke vrednosti m_i i d_i, pa $m_{kn+r} = m_r + k\mu(P)$, $d_{kn+r} = d_r$, i d_r je udaljenost između p i prave koja sadrži r-tu stranicu poligona P. Leva krajnja tačka i-tog hiperboličnog segmenta je $E_i := (e_i, h(e_i))$, gde $e_i = k\mu(P) + \mu(v_r)$, a desni kraj je u $(e_{i+1}, h(e_{i+1}))$. Smatramo da svaki hiperbolični segment sadrži svoju levu krajnju tačku, ali ne i desnu. Sa $H(t)$ Označavamo tačku na grafiku funkcije h koja odgovara parametru t, dakle $H(t) := (t, h(t))$. Grafik je periodičan sa periodom $\mu(P)$, to jest, $H(t) = H(t + k\mu(P))$. Sa $\mathscr{H}_i$ označavamo i-ti hiperbolični segment.

E_i, e_i

$H(t)$



Slika 53. Relacija između dilacije i nagiba.

$o(t)$, $O(t)$, $o_i(t)$, $O_i(t)$

slope, $s(t_q, t_r)$

Neka je $o(t) = t - h(t)$, i $O(t) = (o(t), 0)$, videti Sliku 53. Takođe definišemo $o_i(t) = t - h_i(t)$, i $O_i(t) = (o_i(t), 0)$. Za date tačke $q \in P$ i $r \in P^+[q]$, imamo njihove odgovarajuće parametre t_q i t_r, takve da $t_q \le t_r \le t_q + \mu(P)/2$. *Nlope* prave koja prolazi kroz tačke $O(t_q)$ i $H(t_r)$ je

$$\begin{aligned} s(t_q, t_r) &:= \operatorname{slope}\big(\ell(O(t_q), H(t_r))\big) \\ &= \frac{h(t_r)}{t_r - t_q + h(t_q)} \\ &= \frac{|pP(t_r)|}{\mu(P(t_q), P(t_r)) + |pP(t_q)|} \qquad (1.9) \\ &= \frac{1}{\delta^+_{P(t_q)}(P(t_r))}, \end{aligned}$$

pa je stoga nagib od $O(t_q)$ do $H(t_r)$ jednak inverzu leve dilacije tačke r preko q.

$\tilde{s}(t_q)$

Definicija 105. Definišemo $\tilde{s}(t_q)$ kao najmanji nagib od $O(t_q)$ do $H(t_r)$ kada $t_r \in [t_q, t_q + \mu(P)/2]$.

Iz prethodnog razmatranja sledi da je ovaj nagib jednak inverzu leve dilacije preko q,

$$\begin{aligned}\tilde{s}(t_q) &:= \min_{t_r\in[t_q,t_q+\mu(P)/2]} s(t_q,t_r)\\ &= \min_{t_r\in[t_q,t_q+\mu(P)/2]} \frac{1}{\delta^+_{P(t_q)}(P(t_r))}\\ &= \frac{1}{\max_{r\in P^+[q]}\delta^+_q(r)}\\ &= \frac{1}{\tilde{\delta}^+_q}.\end{aligned} \tag{1.10}$$

Očigledno, $\tilde{s}(t)\in(0,1]$ jer je strogo pozitivno i $\tilde{s}(t)\le s(t,t)=1$.

Lema 106. *Za svake dve različite vrednosti t_1 i t_2,*

$$|h(t_2)-h(t_1)|\le|t_2-t_1|.$$

Do sada, $\tilde{s}(t_q)$ je bilo definisano kao minimum među nagibima $s(t_q,t_r)$ gde t_r pripada intervalu $[t_q,t_q+\mu(P)/2]$. Međutim, iz Leme 106 sledi da $s(t_q,t_r)$ ne može biti manje od 1 kada $t_r\in[o(t_q),t_q]$, a $\tilde{s}(t_q)$ je najviše 1, pa ovaj interval nad kojim tražimo minimum može biti proširen i dobijamo

$$\tilde{s}(t_q)=\min_{t_r\in[o(t_q),t_q+\mu(P)/2]} s(t_q,t_r).$$

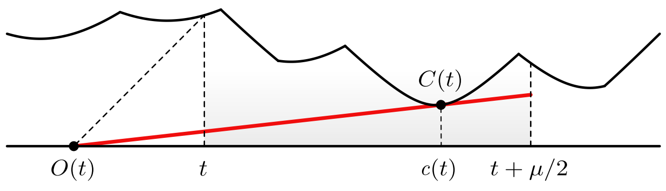
Ovo proširenje intervala nam omogućava da definišemo *polugu*

Algoritam klizeće poluge

Poluga

Za fiksirano t, posmatrajmo duž sa nagibom $\tilde{s}(t)$, sa jednom krajnjom tačkom u $O(t)$, a drugom u $(t+\mu(P)/2,\tilde{s}(t)(\mu(P)/2+h(t)))$, vidi Sliku 54. Nazovimo tu duž *poluga za t*.

poluga za t



Slika 54. Poluga.

Neka je $C(t)$ najlevlja tačka u kojoj poluga za t dodiruje grafik, i neka je $c(t)$ takvo da je $H(c(t)) = C(t)$. Tada $c(t) \in [o(t), t + \mu(P)/2]$ i to je najmanja vrednost u ovom intervalu za koju je $\tilde{s}(t) = s(t, c(t))$. U originalnoj postavci problema ovo znači da leva dilacija preko $P(t)$ dostiže svoj maksimum za $P(c(t))$.

Ideja algoritma klizeće poluge je da kontinualno smanjujemo parametar t i posmatramo šta se dešava sa polugom za t. Sledeća lema nam govori o tome da smanjivanje parametra t odgovara "klizanju" poluge ulevo.

Lema 107. *Za $t_1 < t_2$ važi $o(t_1) \leq o(t_2)$ i $c(t_1) \leq c(t_2)$.*

Stanja poluge

Na kontinualno kretanje poluge u levo možemo gledati kao iteraciju kroz diskretan niz stanja. Definišemo ralzičita stanja u zavisnosti od položaja poluge u odnosu na niz hiperboličnih segmenata.

Kada $t \in [e_i, e_{i+1})$ i $c(t) \in [e_j, e_{j+1})$, kažemo da je poluga za t u *fazi* $\langle i, j\rangle$. Faza i način na koji poluga dodiruje grafik zajedno čine *stanje poluge*. Postoje tri moguća načina da poluga dodiruje grafik.

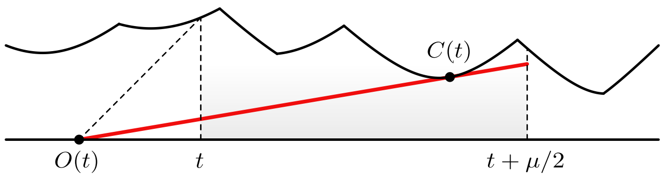
faza

stanje poluge

Stanje $\langle i, j\rangle^{\mathscr{K}}$

- *Stanje* $\langle i, j\rangle^{\mathscr{K}}$: $c(t) < t + \mu(P)/2$ i poluga je tangenta na $\mathscr{H}_j$.

Slika 55. Stanje $\langle i, j\rangle^{\mathscr{K}}$

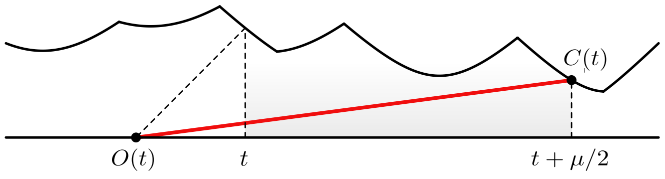


Kada se t smanjuje, poluga klizi ulevo ostajući tangenta na h_j. Nagib se smanjuje.

Stanje $\langle i, j\rangle^{\mathscr{Y}}$

- *Stanje* $\langle i, j\rangle^{\mathscr{Y}}$: $c(t) = t + \mu(P)/2$.

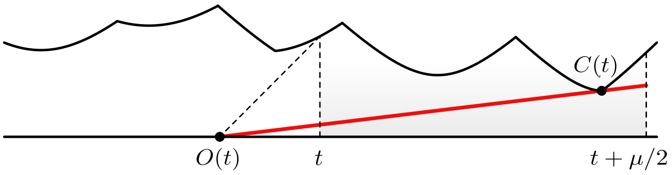
Slika 56. Stanje $\langle i, j\rangle^{\mathscr{Y}}$



Tačka $C(t)$ je desni kraj poluge. To je jedina tačka gde poluga dodiruje grafik. Kada se t smanjuje, poluga klizi ulevo, održavajući svoju desnu krajnju tačku na h_j.

- *Stanje* $\langle i,j\rangle^{\mathcal{V}}$: $c(t) < t + \mu(P)/2$ i poluga prolazi kroz tačku $H(e_j)$, koja je zajednička krajnja tačka hiperboličnih segmenata $\mathscr{H}_{j-1}$ i $\mathscr{H}_j$.

Stanje $\langle i,j\rangle^{\mathcal{V}}$



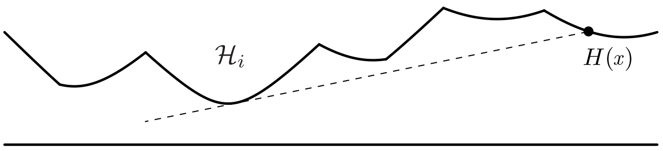
Slika 57. Stanje $\langle i,j\rangle^{\mathcal{V}}$

Ova situacija se dešava samo ako je $m_{j-1} > m_j$. Kada se t smanjuje, poluga klizi ulevo održavajući kontakt sa tom zajedničkom tačkom između dva hiperbolična segmenta, stoga se nagib smanjuje.

Skokovi

Prilikom smanjivanja t i klizanja poluge ulevo, dešavaju se *skokovi*, događaji u kojima $C(t)$ diskontinualno menja svoju poziciju prebacujući se na drugi hiperbolični segment. U svakom trenutku nam je neophodno da znamo na koji hiperbolični segment poluga može skočiti iz trenutne pozicije. Uvek postoji najviše jedan takav potencijalni segment.

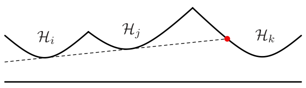
skok



Slika 58. Doskok za x. $jump(x) = i$

Posmatrajmo neku tačku $H(x)$ na grafiku. Neka je jump(x), *doskok za x* indeks hiperboličnog segmenta koji sadrži najdešnju tačku $H(w)$ sa grafika za koju je $w < x$ i poluprava iz $H(x)$ kroz $H(w)$ samo dodiruje grafik, videti Sliku 58. To jest, jump(x) je indeks najnižeg vidljivog hiperboličnog segmenta kada se iz tačke $H(x)$ gleda na levo. Ako ne postoji takvo w, jer je su segmenti sa leve strane zaklonjeni segmentom koji sadrži $H(x)$, postavljamo jump(x) na indeks segmenta koji sadrži $H(x)$.

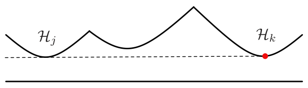
jump(x), *doskok za x*



Slika 59. Tačke promene doskoka prvog tipa.

Posmatrajmo vrednosti x u kojima se jump(x) menja. Takve vrednosti nazivamo *pozicije promene doskoka*, a tačke $H(x)$ *tačke promene doskoka*. Postoje dve vrste tačaka promene doskoka. *Tačke promene doskoka prvog tipa* su tačke na $\mathscr{H}_k$ u kojima se doskok menja sa i na j, gde je $i < j < k$, videti Sliku 59. *Tačke promene*

pozicije promene doskoka
tačke promene doskoka



Slika 60. Tačke promene doskoka drugog tipa.

doskoka drugog tipa su tačke na $\mathscr{H}_k$ u kojima se doskok menja sa k na j, gde je $j < k$, videti Sliku 60.

Teorema 108. *Moguće je odrediti sve tačke promene doskoka uređene s leva na desno, zajedno sa doskocima svih tih tačaka u $O(n)$ vremenu.*

Događaji

događaji promene stanja

Tokom procesa smanjivanja vrednosti t, stanje poluge se menja u određenim momentima. Te događaje nazivamo *događaji promene stanja*. Ako je poznata trenutna pozicija poluge, t_c i trenutno stanje, sledeći događaj se može odrediti održavanjem skupa svih potencijalnih budućih događaja i uzimanjem onog događaja iz tog skupa koji ima najveće t ne veće od t_c.

Postoji više različitih tipova događaja koji se mogu dogoditi prilikom pomeranja poluge ulevo. Za svaki umemo izračunati položaj poluge t u kome se taj događaj dešava. To radimo tako što postavljamo i rešavamo polinomnu jednačinu koja opisuje taj tip događaja.

Događaji promene doskoka

Ovi događaji se dešavaju kad god $C(t)$ pređe preko neke tačke promene doskoka. U tom momentu potrebno je ponovo izračunati sve buduće događaje koje uključuju skokove. Način na koji se računaju zavisi od tipa trenutnog stanja poluge.

Događaji promene stanja

Dajemo pregled svih mogućih tipova događaja promene stanja. Za svaki umemo da izračunamo odgovarajuću t vrednost.

- Tipovi događaja koji vode u $\langle i,j\rangle$ oblik stanja:
  $\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j\rangle^{\mathscr{K}}$, $\langle i,j\rangle^{\mathscr{K}} \to \langle i,j\rangle^{\mathscr{Y}}$, $\langle i,j\rangle^{\mathscr{K}} \to \langle i,j\rangle^{\mathscr{V}}$
- Tipovi događaja koji vode u $\langle i-1,j\rangle$ oblik stanja:
  $\langle i,j\rangle^{x} \to \langle i-1,j\rangle^{x}$, gde $x \in \{\mathscr{Y},\mathscr{K},\mathscr{V}\}$
- Tipovi događaja koji vode u $\langle i,j-1\rangle$ oblik stanja:
  $\langle i,j\rangle^{\mathscr{Y}} \to \langle i,j{-}1\rangle^{\mathscr{Y}}$, $\langle i,j\rangle^{\mathscr{V}} \to \langle i,j{-}1\rangle^{\mathscr{Y}}$, $\langle i,j\rangle^{\mathscr{V}} \to \langle i,j{-}1\rangle^{\mathscr{K}}$

- Tipovi događaja koji vode u $\langle i, j_m \rangle$ oblik stanja:
  $\langle i, j\rangle^{\mathscr{Y}} \to \langle i, j_m\rangle^{\mathscr{H}}$, $\langle i, j\rangle^{\mathscr{H}} \to \langle i, j_m\rangle^{\mathscr{H}}$, $\langle i, j\rangle^{\mathscr{V}} \to \langle i, j_m\rangle^{\mathscr{H}}$, $\langle i, j\rangle^{\mathscr{Y}} \to \langle i, j_m\rangle^{\mathscr{V}}$, $\langle i, j\rangle^{\mathscr{H}} \to \langle i, j_m\rangle^{\mathscr{V}}$, $\langle i, j\rangle^{\mathscr{V}} \to \langle i, j_m\rangle^{\mathscr{V}}$

Niz realizovanih stanja

Želimo da efikasno nađemo niz stanja kroz koje poluga prolazi na svom putovanju ulevo, zajedna sa pozicijama u kojima se premene stanja dešavaju Neka je dobijeni niz $p_1, \mathscr{S}_1, p_2, \mathscr{S}_2, p_3, \ldots, p_r, \mathscr{S}_r$, gde je $p_1 \leq p_2 \leq \ldots \leq p_r$. Svako stanje $\mathscr{S}_k$ se dešava kada je pozicija poluge tačno između p_k i p_{k+1}, gde je $p_{r+1} = p_1 + \mu(P)$. Ovaj niz nazivamo *niz realizovanih stanja*.

niz realizovanih stanja

Teorema 109. *Možemo odrediti niz realizovanih stanja u $O(n)$ vremenu. Dužina dobijenog niza je $O(n)$.*

Spajanje leve i desne dilacije

Vraćamo se na originalni problem gde dilacija predstavlja kombinaciju leve i desne dilacije. Algoritam za dobijanje niza realizovanih stanja pokrećemo jednom za levu i jednom za desnu dilaciju, i potom ta dva niza spajamo u jedan. Ovo spajanje je moguće izvršiti u linearnom vremenu.

Poznavanje ovog spojenog niza realizovanog stanja je dovoljno da se u svakoj poziciji t odredi nagib leve i desne poluge, a samim tim i leva i desna dilacija, odnosno njihov maksimum, dilacija preko $P(t)$.

Za svako kombinovano stanje u spojenom nizu u konstantnom vremenu umemo da odredimo minimum, a kako kombinovanih stanja ima samo linearno mnogo, to je ukupno vreme potrebna do se odredi dilacija jednako $O(n)$.

Bibliography

[1] Bernardo M. Ábrego, Esther M. Arkin, Silvia Fernández-Merchant, Ferran Hurtado, Mikio Kano, Joseph S. B. Mitchell, and Jorge Urrutia. Matching points with circles and squares. In Jin Akiyama, Mikio Kano, and Xuehou Tan, editors, *Discrete and Computational Geometry*, pages 1–15, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[2] A Karim Abu-Affash, Sujoy Bhore, and Paz Carmi. Monochromatic plane matchings in bicolored point set. In *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, Carleton University, Ottawa, Ontario, Canada*, pages 7–12, 2017.

[3] A Karim Abu-Affash, Sujoy Bhore, Paz Carmi, and Dibyayan Chakraborty. Bottleneck bichromatic full steiner trees. In *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, Carleton University, Ottawa, Ontario, Canada*, pages 13–18, 2017.

[4] 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.

[5] 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.

[6] P.K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, and M. Soss. Computing the detour and spanning ratio of paths, trees and cycles in 2d and 3d. *Discrete and Computational Geometry*, 39(1–3):17–37, 2008.

[7] Hee-Kap Ahn, Mohammad Farshi, Christian Knauer, Michiel Smid, and Yajun Wang. Dilation-optimal edge deletion in

polygonal cycles. In Takeshi Tokuyama, editor, *Algorithms and Computation*, pages 88–99, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] A.M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.

[14] B. Aronov, K. Buchin, M. Buchin, B. Jansen, T. de Jong, M. van Kreveld, M. Löffler, J. Luo, R.I. Silveira, and B. Speckmann. Connect the dot: Computing feed-links for network extension. *Journal of Spatial Information Science*, (3):3–31, 2012.

[15] Sergey Bereg, Nikolaus Mutsanas, and Alexander Wolff. Matching points with rectangles and squares. *Computational Geometry*, 42(2):93 – 108, 2009.

[16] Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. An optimal algorithm for the euclidean bottleneck full steiner tree problem. *Computational Geometry*, 47(3, Part A):377 – 380, 2014.

[17] Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Matchings in higher-order gabriel graphs. *Theoretical Computer Science*, 596:67 – 78, 2015.

[18] Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Bottleneck matchings and hamiltonian cycles in higher-order gabriel graphs. In *Proceedings of the 32nd European Workshop on Computational Geometry (EuroCG'16)*, pages 179 – 182. EuroCG, 2016.

[19] Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Strong matching of points with geometric shapes. *Computational Geometry*, 68:186 – 205, 2018. Special Issue in Memory of Ferran Hurtado.

[20] Ahmad Biniaz, Anil Maheshwari, and Michiel H. M. Smid. Bottleneck bichromatic plane matching of points. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada*, 2014.

[21] P. Bose, K. Dannies, J.L. De Carufel, C. Doell, C. Grimm, A. Maheshwari, S. Schirra, and M. Smid. Network farthest-point diagrams and their application to feed-link network extension. *Journal of Computational Geometry*, 4(1):182–211, 2013.

[22] John Gunnar Carlsson, Benjamin Armbruster, Saladi Rahul, and Haritha Bellam. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applications*, 25(4):245–262, 2015.

[23] 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.

[24] A. Dahlgren and L. Harrie. Evaluation of computational methods for connecting points to large networks. *Mapping and Image Science*, (4):45–54, 2006.

[25] A. Dahlgren and L. Harrie. Development of a tool for proximity applications. In *Proceedings of AGILE*, 2007.

[26] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[27] Satyan L. Devadoss and Joseph O'Rourke. *Discrete and Computational Geometry*. Princeton University Press, 2011.

[28] A. Dickenstein and I.Z. Emiris. *Solving polynomial equations: Foundations, algorithms, and applications*, volume 14. Springer Verlag, 2005.

[29] Michael B. Dillencourt. Toughness and delaunay triangulations. *Discrete & Computational Geometry*, 5(6):575–601, Dec 1990.

[30] A. Ebbers-Baumann, A. Grüne, and R. Klein. On the geometric dilation of finite point sets. *Algorithmica*, 44(2):137–149, 2006.

[31] A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A fast algorithm for approximating the detour of a polygonal chain. *Computational Geometry Theory and Applications*, 27(2):123–134, 2004.

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

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

[34] Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2008, https://doi.org/10.1137/050635675.

[35] Tomáš Kaiser, Maria Saumell, and Nico Van Cleemput. 10-gabriel graphs are hamiltonian. *Information Processing Letters*, 115(11):877 – 881, 2015.

[36] Rolf Klein and Martin Kutz. Computing geometric minimum-dilation graphs is np-hard. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing*, pages 196–207, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[37] 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.

[38] Jiří Matoušek. *Lectures on Discrete Geometry*. Springer, 2002.

[39] G. Narasimhan and M. Smid. Approximating the stretch factor of euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.

[40] Marko Savić and Miloš Stojaković. Linear time algorithm for optimal feed-link placement. *Computational Geometry*, 48(3):189 – 204, 2015.

[41] Marko Savić and Miloš Stojaković. Non-crossing bottleneck matchings of points in convex position. In *Proceedings of the 32nd European Workshop on Computational Geometry (EuroCG'16)*, pages 175 – 178. EuroCG, 2016.

[42] Marko Savić and Miloš Stojaković. Bottleneck bichromatic non-crossing matchings using orbits. In *Proceedings of the 34th European Workshop on Computational Geometry (EuroCG'18)*, pages 425 – 430. EuroCG, 2018.

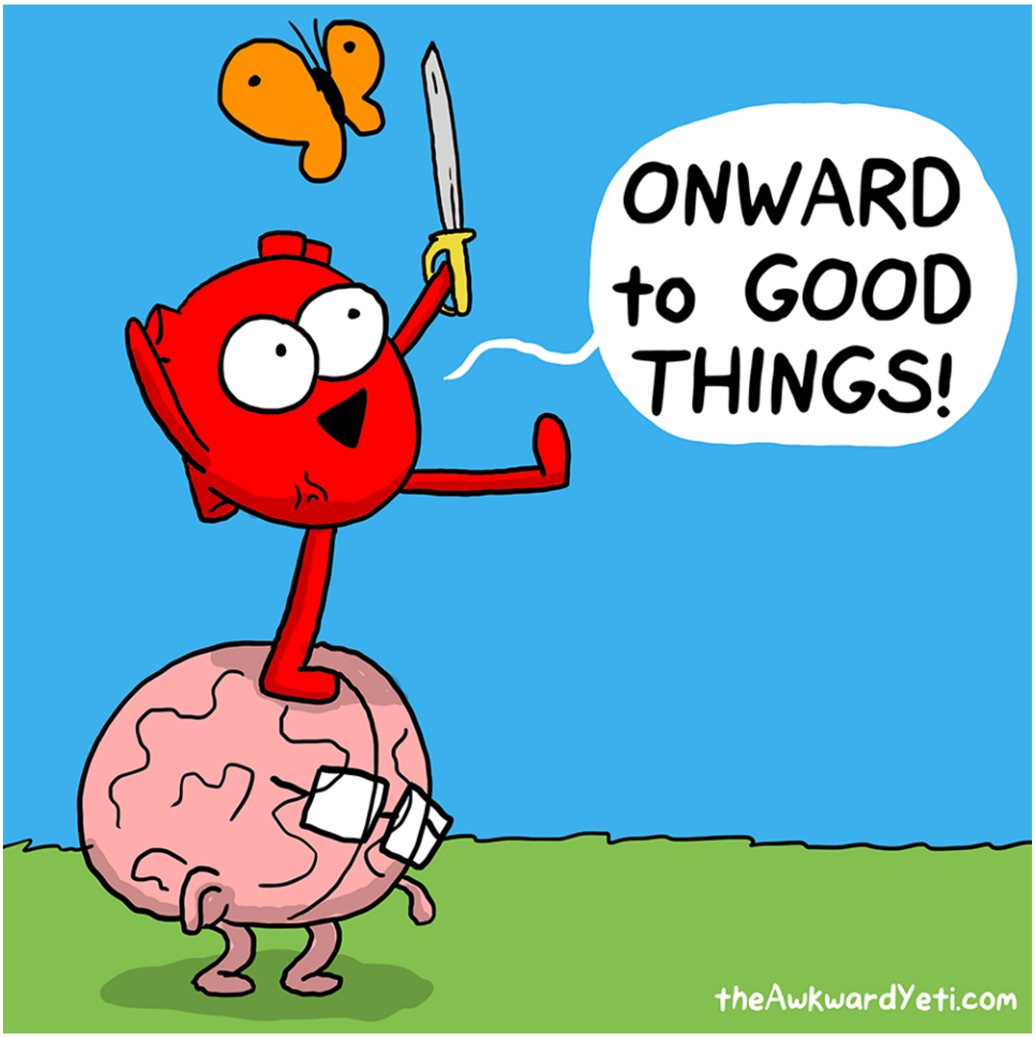
[43] Marko Savić and Miloš Stojaković. Faster bottleneck non-crossing matchings of points in convex position. *Computational Geometry*, 65:27–34, 2017.

[44] Marko Savić and Miloš Stojaković. Bottleneck bichromatic non-crossing matchings using orbits. *ArXiv e-prints*, 2018, 1802.06301.

[45] Christian Wulff-Nilsen. Computing the maximum detour of a plane graph in subquadratic time. In *Algorithms and Computation*, pages 740–751, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[46] Christian Wulff-Nilsen. Computing the dilation of edge-augmented graphs in metric spaces. *Computational Geometry*, 43(2):68 – 72, 2010. Special Issue on the 24th European Workshop on Computational Geometry (EuroCG'08).

[47] Christian Wulff-Nilsen, Ansgar Grüne, Rolf Klein, Elmar Langetepe, D. T. Lee, Tien-Ching Lin, Sheung-Hung Poon, and Teng-Kai Yu. Computing the stretch factor and maximum detour of paths, trees, and cycles in the normed space. *Int. J. Comput. Geometry Appl.*, 22:45–60, 2012.


ONWARD to GOOD THINGS!
theAwkwardYeti.com

Kratka Biografija

Marko Savić je rođen 1982. godine u Novom Sadu, gde je završio osnovnu i srednju školu. Za vreme školovanja postigao je brojne uspehe na takmičenjima iz informatike, matematike i fizike, od kojih je najveći osvajanje srebrne medalje na međunarodnoj informatičkoj olimpijadi 2000. godine u Kini.

Prirodno-matematički fakultet u Novom Sadu, smer informatika, završio je sa prosečnom ocenom 10, kao najbolji student u svojoj generaciji. Za postignut uspeh u toku studija nagrađivan je od strane Fakulteta i Univerziteta u Novom Sadu, a tokom studija bio je i stipendista Ministarstva za nauku i tehnološki razvoj.

Nakon završetka osnovnih studija, na istom fakultetu je završio master studije, modul teorijsko računarstvo, a potom upisao i doktorske studije, 2010. godine. Od 2011. godine zaposlen je na matičnom fakultetu kao istraživač-saradnik na projektu Ministarstva prosvete, nauke i tehnološkog razvoja, a od 2014. kao asistent. Oblasti njegovog naučnog rada i interesovanja su diskretna i kombinatorna geometrija i diskretni algoritmi. Iz tih oblasti ima objavljene radove u međunarodnim časopisima, i više puta je izlagao na međunarodnim naučnim skupovima. Učestvovao je u više međunarodnih naučnih škola, kurseva i radionica.

Od 2007. do 2011. godine je radio u kompaniji Wowd (SAD, odeljenje u Beogradu) na istraživanju i razvoju skalabilnog distribuiranog internet pretraživača. Autor je jednog patenta zavedenog u američkom patentnom birou.

Od 2010. godine je angažovan i u izvođenju nastave na Departmanu za matematiku i informatiku, i od tada je držao vežbe iz više predmeta. Od 2012. godine drži mentorsku nastavu iz informatike talentovanim učenicima u gimnaziji "Jovan Jovanović Zmaj". Član je republičke komisije za organizaciju srednjoškolskih takmičenja iz informatike i član je redakcije časopisa za matematiku i računarstvo "Tangenta".

Univerzitet u Novom Sadu
Prirodno–matematički fakultet

Ključna dokumentacijska informacija

Redni broj (RBR):	
Identifikacioni broj (IBR):	
Tip dokumentacije (TD):	Monografska dokumentacija
Tip zapisa (TZ):	Tekstualni štampani materijal
Vrsta rada (VR):	Doktorska teza
Ime i prezime autora (AU):	Marko Savić
Mentor (MN):	dr Miloš Stojaković, redovni profesor
Naslov rada (MR):	Efikasni algoritmi za probleme iz diskretne geometrije
Jezik publikacije (JP):	Engleski
Jezik izvoda (JI):	Engleski, srpski
Zemlja publikovanja (ZP):	Srbija
Uže geografsko područje (UGP):	Vojvodina
Godina (GO):	2018.
Izdavač (IZ):	autorski reprint
Mesto i adresa (MA):	Novi Sad, Trg Dositeja Obradovića 4
Fizički opis rada (FO):	3/i-x,121/78/0/13/60/0
Naučna oblast (NO):	Računarske nauke
Naučna disciplina (ND):	Teorijsko računarstvo
Predmetna odrednica, ključne reči (PO):	Algoritamska geometrija, diskretna geometrija, algoritmi, geometrijski mečinzi, bottleneck mečinzi, dilacija mreže
UDK:	
Čuva se (ČU):	Biblioteka Departmana za matematiku i informatiku, Prirodno–matamatički fakultet, Novi Sad
Važna napomena (VN):	

Izvod (IZ):

Prva klasa problema koju proučavamo tiče se geometrijskih mečinga. Za dat skup tačaka u ravni, posmatramo savršene mečinge tih tačaka spajajući ih dužima koje se ne smeju seću. Bottleneck mečing je takav mečing koji minimizuje dužinu najduže duži. Naš cilj je da nađemo bottleneck mečing tačaka u konveksnom položaju.
Za monohromatski slučaj, u kom je dozvoljeno upariti svaki par tačaka, dajemo algoritam vremenske složenosti $O(n^2)$ za nalaženje bottleneck mečinga. Ovo je bolje od prethodno najbolji poznatog algoritam, čija je složenost $O(n^3)$.
Takođe proučavamo bihromatsku verziju ovog problema, u kojoj je svaka tačka obojena ili u crveno ili u plavo, i dozvoljeno je upariti samo tačke različite boje. Razvijamo niz alata za rad sa bihromatskim nepresecajućim mečinzima tačaka u konveksnom položaju. Kombinovanje ovih alata sa geometrijskom analizom omogućava nam da rešimo problem nalaženja bottleneck mečinga u $O(n^2)$ vremenu. Takođe, konstruišemo algoritam vremenske složenosti $O(n)$ za slučaj kada sve date tačke leže na krugu. Prethodno najbolji poznati algoritmi su imali složenosti $O(n^3)$ za tačke u konveksnom položaju i $O(n \log n)$ za tačke na krugu.
Druga klasa problema koju proučavamo tiče se dilacije u geometrijskim mrežama. Za datu mrežu predstavljenu poligonom, i tačku p u istoj ravni, želimo proširiti mrežu dodavanjem duži zvane feed-link koja povezuje p sa obodom poligona. Kada je feed-link fiksiran, definišemo geometrijsku dilaciju neke tačke q na obodu kao odnos između dužine najkraćeg puta od p do q kroz proširenu mrežu i njihovog Euklidskog rastojanja. Korisnost feed-linka je obrnuto proporcionalna najvećoj dilaciji od svih tačaka na obodu poligona.
Konstruišemo algoritam linearne vremenske složenosti koji nalazi feed-link sa najmanom sveukupnom dilacijom. Ovim postižemo bolji rezultat od prethodno najboljeg poznatog algoritma složenosti približno $O(n \log n)$.

Datum prihvatanja teme od strane Senata (DP):	14.09.2017.
Datum odbrane (DO):	
Članovi komisije (KO):	**Predsednik:** dr Dragan Mašulović, redovni profesor, Prirodno–matematički fakultet, Univerzitet u Novom Sadu **Mentor:** dr Miloš Stojaković, redovni profesor, Prirodno–matematički fakultet, Univerzitet u Novom Sadu **Član:** dr Dejan Vukobratović, vanredni profesor, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

University of Novi Sad
Faculty of Sciences

Key word documentation

Accession number (ANO):	
Identification number (INO):	
Document type (DT):	Monograph documentation
Type of record (TR):	Textual printed material
Contents Code (CC):	
Author (AU):	Marko Savić
Mentor (MN):	Miloš Stojaković, Ph.D., Full Professor
Title (XI):	Efficient algorithms for discrete geometry problems
Language of text (LT):	English
Language of abstract (LA):	English, Serbian
Country of publication (CP):	Serbia
Locality of publication (LP):	Vojvodina
Publication year (PY):	2018
Publisher (PU):	Author's reprint
Publ. place (PP):	Novi Sad, Trg Dositeja Obradovića 4
Physical description (PD):	3/i-x,121/78/0/13/60/0
Scientific field (SF):	Computer science
Scientific discipline (SD):	Theoretical computer science
Subject, Key words (SKW):	Computational geometry, discrete geometry, algorithms, geometric matchings, bottleneck matchings, network dilation
UC:	
Holding data (HD):	The Library of the Department of Mathematics and Informatics, Faculty of Science, Novi Sad
Note (N):	

Abstract (AB):

The first class of problem we study deals with geometric matchings. Given a set of points in the plane, we study perfect matchings of those 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 are interested in finding a bottleneck matching of points in convex position.
In the monochromatic case, where any two points are allowed to be matched, we give an $O(n^2)$-time algorithm for finding a bottleneck matching, improving upon previously best known algorithm of $O(n^3)$ time complexity.
We also study a bichromatic version of this problem, where each point is colored either red or blue, and only points of different color can be matched. We develop a range of tools, for dealing with bichromatic non-crossing matchings of points in convex position. Combining that set of tools with a geometric analysis enable us to solve the problem of finding a bottleneck matching in $O(n^2)$ time. We also design an $O(n)$-time algorithm for the case where the given 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.
The second class of problems we study deals with dilation of geometric networks. Given a polygon representing a network, and a point p in the same plane, we aim to extend the network by inserting a line segment, called a feed-link, which connects p to the boundary of the polygon. Once a feed link is fixed, the geometric dilation of some point q on the boundary is the ratio between the length of the shortest path from p to q through the extended network, and their Euclidean distance. The utility of a feed-link is inversely proportional to the maximal dilation over all boundary points. We give a linear time algorithm for computing the feed-link with the minimum overall dilation, thus improving upon the previously known algorithm of complexity that is roughly $O(n \log n)$.

Accepted on Senate on (ASB): 14.09.2017.
Defended (DE):