Stabbing and Covering Geometric Objects in the Plane

Dissertation zur Erlangung des Doktorgrades

vorgelegt am

Fachbereich Mathematik und Informatik
der Freien Universität Berlin

2013

von

Lena Marie Schlipf

Institut für Informatik
Freie Universität Berlin
Takustraße 9
14195 Berlin
schlipf@inf.fu-berlin.de

gefördert durch die Deutsche Forschungsgemeinschaft im Rahmen
des Graduiertenkollegs *„Methods for Discrete Structures“* (GRK 1408)
und des Schwerpunktprogramms *„Algorithm Engineering“* (SPP 1307)

Betreuer:

Prof. Dr. Helmut Alt
Institut für Informatik
Freie Universität Berlin
Takustraße 9
14195 Berlin
Germany
alt@inf.fu-berlin.de
phone: +49 30 838 75160

Prof. Dr. Christian Knauer
Institut für Angewandte Informatik
Universität Bayreuth
Universitätstraße 30
95447 Bayreuth
Germany
christian.knauer@uni-bayreuth.de
phone: +49 921 55 7740

Gutachter:

Prof. Dr. Helmut Alt
Institut für Informatik
Freie Universität Berlin
Takustraße 9
14195 Berlin
Germany
alt@inf.fu-berlin.de
phone: +49 30 838 75160

Prof. Dr. Otfried Cheong
Department of Computer Science
KAIST
291 Daehak-ro, Yuseong-gu
Daejeon 305-701
Korea
otfried@kaist.edu
phone: +82 42 350 3542

Prof. Dr. Christian Knauer
Institut für Angewandte Informatik
AG Algorithmen und
Datenstrukturen (AI VI)
Universität Bayreuth
95440 Bayreuth
Germany
christian.knauer@uni-bayreuth.de
phone: +49 921 55 7740

Vorlage zur Begutachtung:	16.05.2013
Termin der Disputation:	11.09.2013
Fassung vom:	26.11.2013

Contents

Abstract

In this thesis, we consider a variety of different geometric covering and stabbing problems in the plane. Stabbing and covering geometric objects are two widely studied problems in computational geometry. These problems are closely related; there are many cases where covering problems are dual to stabbing problems.

We first study a problem that was posed by Tamir in 1987 [109]: "Given a set of geometric objects in the plane, can one decide in polynomial time whether there exists a convex polygon whose boundary stabs every object ?" This boundary is then called a convex stabber. We give an answer to this question by proving that deciding the existence of a convex stabber is NP-hard for many types of geometric objects. Additionally, we consider an optimization version and prove it to be APX-hard for most of the considered objects.

A similar problem is deciding whether geometric objects can be stabbed with the vertices of a rotated, scaled and translated copy of a given polygon. To the best of our knowledge, this problem was not studied so far and we present the first polynomial-time algorithm.

Another stabbing problem studied in this thesis, is the problem of stabbing sequences of geometric objects: Given a distance measure and two sequences of geometric objects, compute two point sequences that stab them under the condition that the distance between these point sequences is as small as possible (using the given distance measure). We state efficient algorithms for this problem where the objects are either line segments or disks and the distance measure is the discrete Fréchet distance.

Then, we consider covering problems. We study a new version of the two-center problem where the input is a set $\mathcal{D}$ of disks in the plane. We first study the problem of finding two smallest congruent disks such that each disk in $\mathcal{D}$ intersects one of these two disks. Then, we study the problem of covering the set $\mathcal{D}$ by two smallest congruent disks. We also investigate an optimization version. For these problems, we give efficient exact and approximation algorithms.

Finally, we investigate the problem of computing a largest area rectangle inscribed in a convex polygon on n vertices. If the order of the vertices of the polygon is given, we state approximation algorithms whose running times are only logarithmically dependent on n.

Zusammenfassung

Wir beschäftigen uns in dieser Arbeit mit verschiedenen geometrischen Überdeckungs- sowie Transversalenproblemen in der Ebene. Beide Gebiete zeigen ihre Relevanz durch zahlreiche Untersuchungen in der algorithmischen Geometrie. Die Fragestellungen sind häufig verwandt, da viele Überdeckungsprobleme dual zu Transversalenproblemen sind.

Die Arbeit beginnt mit der Betrachtung von Transversalenproblemen. Zuerst beschäftigen wir uns mit einer Frage, die bereits 1987 von Tamir gestellt wurde [109]: „Sei eine Menge von geometrischen Objekten in der Ebene gegeben. Kann man in polynomieller Zeit entscheiden, ob ein konvexes Polygon existiert, dessen Rand all diese Objekte aufspießt?“ Den Rand eines solchen Polygons nennen wir konvexe Transversale. Wir zeigen, dass es für viele geometrische Objekte NP-schwer ist zu entscheiden, ob eine konvexe Transversale existiert. Zusätzlich betrachten wir eine Optimierungsvariante für das Problem und zeigen, dass dieses Problem für die meisten betrachteten Objekte APX-schwer ist. Eine ähnliche Fragestellung ist zu entscheiden, ob ein gegebenes Polygon so gedreht, verschoben und gestreckt werden kann, dass seine Ecken eine gegebene Menge von geometrischen Objekten aufspießen. Dieses Problem wurde, soweit uns bekannt ist, bisher noch nicht betrachtet und wir präsentieren den ersten Polynomialzeitalgorithmus. Als Erweiterung dieser Problematik beschäftigen wir uns mit dem Problem Folgen von geometrischen Objekten mit Punktfolgen aufzuspießen. Dazu nehmen wir an, dass zwei Folgen von geometrischen Objekten gegeben sind. Hierzu werden zwei Punktfolgen gesucht, die die gegebenen Folgen aufspießen und deren Abstand minimal ist (wobei wir als Abstandsmaß die diskrete Fréchetdistanz wählen). Wir untersuchen dieses Problem unter der Bedingung, dass die Objekte Strecken oder Kreisscheiben sind.

Im zweiten Teil der Arbeit betrachten wir Überdeckungsprobleme. Insbesondere untersuchen wir neue Versionen des 2-Zentren Problems, bei denen die Eingabe aus einer Menge $\mathcal{D}$ von Kreisscheiben besteht. Bei der ersten Variante sollen zwei kleinste, kongruente Kreisscheiben berechnet werden, die jede Kreisscheibe in $\mathcal{D}$ schneiden. Bei der zweiten Variante sollen alle Kreisscheiben in $\mathcal{D}$ von zwei kleinsten, kongruenten Kreisscheiben überdeckt werden. Zusätzlich untersuchen wir eine Optimierungsvariante des Problems und geben effiziente exakte Algorithmen und Approximationsalgorithmen an. Zum Schluss untersuchen wir das Problem, ein Rechteck mit größtem Flächeninhalt zu finden, das in ein konvexes Polygon einbeschrieben ist. Unter der Annahme, dass die Reihenfolge der Ecken des Polygons gegeben ist, präsentieren wir Approximationsalgorithmen mit logarithmischer Laufzeit.

Acknowledgement

I want to thank all people that have supported me in writing this thesis. First of all, I thank my advisors Helmut Alt and Christian Knauer. Christian for encouraging me in writing this thesis, for a lot of research discussions and for being always very understanding and supporting. I am grateful to Helmut for taking me as his student when Christian left FU, for his support and his advice and, in particular, for giving me the opportunity to teach again.

I also want to thank Frank Hoffmann and Klaus Kriegel for catching my interest in Computational Geometry in the first place; your seminar on visibility problems in 2006 was quite inspiring to me.

I thank Hee-Kap Ahn for being a great host during my stay at Postech in February 2009, and for giving me the opportunity to work with him on many interesting topics. Special thanks go to Otfried Cheong for hosting me during a pleasant stay at KAIST in April 2011 and for showing me how much fun research can be.

Many thanks go to my co-authors for many fruitful and interesting discussions: Hee-Kap Ahn, Esther M. Arkin, Claudia Dieckmann, Sang-Sub Kim, Christian Knauer, Joseph S. B. Mitchell, Valentin Polishchuk, Marc Scherfenberg, Jens M. Schmidt, Chan-Su Shin, Hans Raj Tiwary, Antoine Vigneron, and Shang Yang.

Of course, I am grateful to all my colleagues from the workgroup "Theoretische Informatik" for many discussions, especially during the Kaffeerunde. Moreover, I want to thank Claudia for being the best office mate I can imagine and Jens for having a lot of fun with when he was still at FU. I am deeply grateful to Wolfgang, Daniel, and Jens for proofreading parts of this thesis.

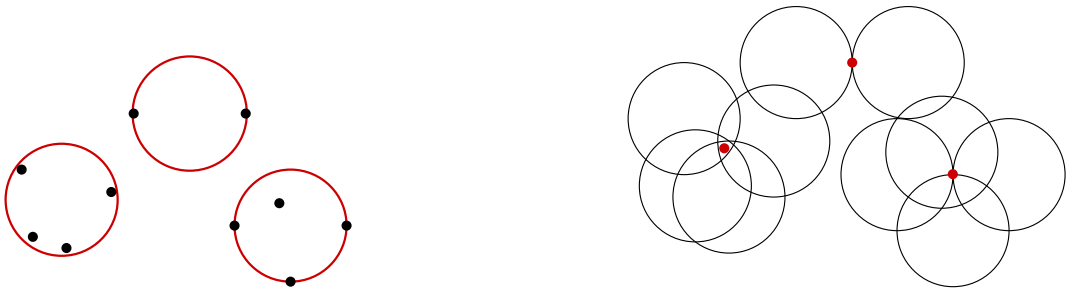
Special thanks go to my parents and my sisters, Sofie and Sarah, for supporting me all the time and always believing in me. I also want to thank my friends for always offering diversion from my work on this thesis. Furthermore, I want to thank Gudrun for still remembering the topic of my Diploma thesis and Swantje for answering all my questions about algebraic geometry.

Finally, I want to thank Panos — $\gamma\iota\alpha$ $\acute{o}\lambda\alpha$.

Introduction

This thesis deals with stabbing and covering geometric objects in the plane. These are two widely studied problems in computational geometry. Both problems have many applications: Typical examples of applications for covering problems are facility location problems, see, e.g., [46]. Possible applications for stabbing problems are motion planning (see [78] for an overview) and curve reconstruction (see [20] and references therein). Even if stabbing and covering problems might look different at the first sight, in many cases they are closely related.

A typical example is the k-center problem: Let P be a set of points in the plane and $k > 0$ an integer. We would like to compute k smallest congruent disks that together cover all points in P. (See Figure 1 (a).)



(a) (b)

Figure 1: A covering problem and its corresponding stabbing problem.

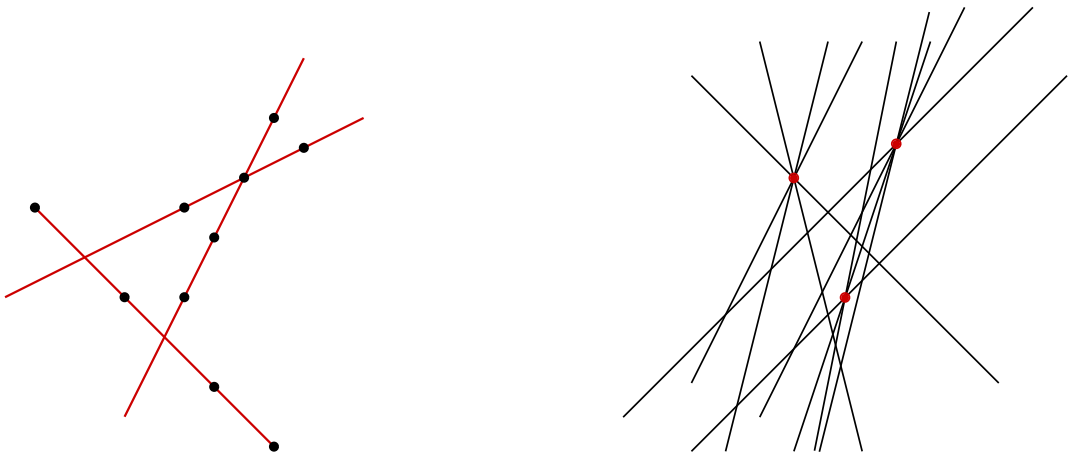
Consider now the decision version of the k-center problem: Let P be a set of points in the plane, $k > 0$ an integer and $r \geqslant 0$ a real value. Is it possible to place k disks of radius r in such a way that they together cover all points in P? Hwang et al. [72] gave an $\mathcal{O}(n^{\sqrt{k}})$ algorithm for this problem. If k is part of the input, the problem was proven to be NP-hard [93]. (See also [5] and references therein.)

In the following we see that the decision version of the k-center problem is dual to the following stabbing problem: Let D be a set of disks in the plane and let $k > 0$ be an integer. We want to decide whether this set D can be stabbed by k points; stabbing problems where the stabbing objects are required to be points are usually referred to as *piercing problems*, see, e.g., [77].

The equivalent stabbing problem for the decision version of the k-center problem can be stated in the following way: We replace each point of P by a disk; each disk has radius r and the centers are the points in P. The result is a set of disks $D = \{p \oplus C_r \mid p \in P\}$ where $\oplus$ is the Minkowski sum and C_r is a disk with radius r. Then, we want to decide whether there are k points such that each disk in D is stabbed by at least one of these points.

These two problems are equivalent because when we solve one of these problems, then we also obtain immediately a result for the other problem: Assume that there exists a set S of k points that together stab all disks in D, then there exist k disks whose centers are the points in S and each disk has radius r and these disks cover all points in P. On the other hand, if there exists a set A of k congruent disks with radius r that together cover all points in P, then there exist k points, namely the center points of the disks in A, that together stab all disks in D. (See also Figure 1.)

Another well-studied example of a covering problem that is dual to a stabbing problem is the problem of covering with hyperplanes: Given a set P of n points in $\mathbb{R}^d$ and an integer $k > 0$, do there exist k hyperplanes that together cover all points in P? By using the common concept of hyperplane-point duality [40], we get the following stabbing problem: Given a set of n hyperplanes in $\mathbb{R}^d$ and an integer $k > 0$, do there exist k points that stab all hyperplanes? This problem is NP-hard, even in the plane [94].



(a) (b)

Figure 2: Another example for a covering problem and its dual stabbing problem.

As mentioned in the beginning of the introduction, in this thesis different geometric stabbing and covering problems are considered. All problems are studied in the plane. While most research on stabbing problems is focused on finding a *line*

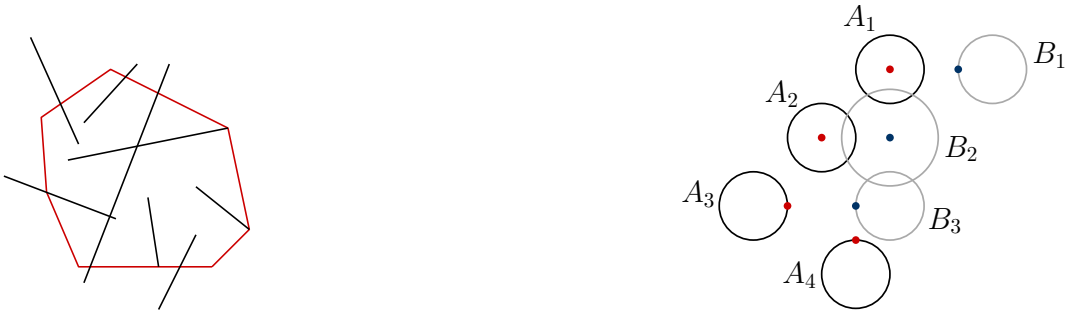
transversal (also called stabbing line), we will study new variants that have rarely been studied so far. For example, we investigate the problem of finding a *convex* transversal, which we also call a convex stabber. This problem was first introduced by Tamir in 1987 [109]. We also consider the problem of stabbing sets of geometric objects with the *vertices* of a given polygon and the problem of stabbing sequences of geometric objects.

As examples for covering problems, we study new variants of the two-center problem where the input is a set of disks instead of a set of points. Finally, we investigate the problem of computing the largest area inscribed rectangle in a convex polygon.

Overview of the Thesis

This thesis consists of two parts. The first part deals with stabbing problems, the second with covering problems. Each part starts with an introduction, which gives a more detailed account of the related literature.

The first part starts with convex stabbing, which is a rarely studied version of stabbing geometric objects in the plane. First, we prove the problem of finding a convex stabber for different kinds of geometric objects to be NP-hard (Chapter 1). We also study an optimization version of this problem and prove it to be APX-hard for most cases. Then, in Chapter 2, we study the problem of stabbing a set of geometric objects with the *vertices* of a polygon.



(a) An example for a convex stabber of a set of segments.

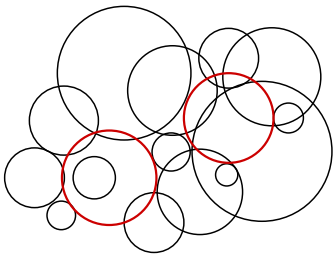
(b) An example for stabbing two sequences of disks with two point sequences.

Figure 3: Stabbing problems.

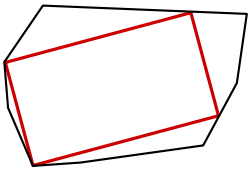
In the last chapter, we investigate the problem of stabbing sequences of geometric objects with point sequences.

The second part deals with covering geometric objects in the plane. There, we study two different variants of covering objects. First, in Chapter 4, we consider a

new version of the well-known *two-center* problem. Here the input is a set of disks instead of a set of points. We investigate different versions and present efficient exact and approximation algorithms. Finally, we give efficient approximation algorithms for the problem of computing the largest inscribed rectangle in a convex polygon (Chapter 5).



(a) An example for a two-center problem for disks.



(b) An example for the largest inscribed rectangle in a convex polygon.

Figure 4: Covering problems.

Part I

Stabbing Geometric Objects in the Plane

Introduction

This part deals with finding transversals for sets of various types of geometric objects. We will investigate a special variant of transversals: *convex* transversals, also called convex stabbers.

Before we explain what a convex transversal is, we give a short overview of the research on transversals in general.

Let S be a set of geometric objects in $\mathbb{R}^d$. A k-transversal to the set S is a k-dimensional flat that intersects each object of S. Research on transversals is an old and rich area, see for instance [38, 62, 115]. If $k = 0$ the problem reduces to deciding whether the set has a common intersection point. If the set consists of convex objects, the *existence* of such a point can be easily answered by using the following theorem:

Helly's Theorem [68]. *Let S be a finite family of n convex sets in $\mathbb{R}^d$, $n > d$. Then if every $d+1$ of the sets have a common intersection point, all sets in S have a common intersection point.*

The problem of *computing* an intersection point can be solved in $\mathcal{O}(n^{d+1}T)$ time where T is the time to compute a point in the intersection of $d+1$ convex sets [26].

However, most of the works consider 1-transversals. In the literature 1-transversals are mostly called line transversals or *stabbing lines*. There is no Helly-type theorem for line transversal as there are families of n convex sets in the plane where every subset of $n-1$ sets has a line transversal but the family has no line transversal. See for instance [115] for an example of such a family. But Hadwiger proved a similar theorem adding an ordering condition.

Hadwiger's Theorem [65]. *Let S be a finite family of n disjoint convex sets in $\mathbb{R}^2$. If there exists a linear ordering on S such that every 3 of the sets are intersected by a directed line in the given order, then S has a line transversal.*
(Notice that this line transversal to S does not necessarily respect the ordering on S.)

Such a transversal meets the members of S in a specific order; each transversal determines two permutations of the sets of S where one is the reverse of the other.

The number of such permutations has been widely studied [50, 76, 114].

There has also been a fair amount of work on *computing* stabbing lines for sets of various types of geometric objects. Edelsbrunner et al. [48] presented in 1982 an algorithm that computes the description of all stabbing lines for a set of n given line segments in the plane in $\mathcal{O}(n \log n)$ time. In 1985 Edelsbrunner [47] stated an $\mathcal{O}(n \log n)$ algorithm that decides whether a set of translates of a simple object in the plane can be stabbed by a line. If the input is a set of axis-aligned rectangles the running time can even be improved to $\mathcal{O}(n)$.

Compared to the work on line transversal the work on convex transversal is rather sparse. The problem was originally proposed by Arik Tamir at the Fourth NYU Computational Geometry Day (March, 1987) [109]: "Given a collection of compact sets, can one decide in polynomial time whether there exists a convex body whose boundary intersects every set in the collection ?" Goodrich and Snoeyink [63] studied the problem with the restriction that the input set is a set of parallel line segments. They proposed an $\mathcal{O}(n \log n)$ algorithm (where n is the number of line segments) which computes a convex polygon whose boundary intersects each line segment at least once, or decides that no such polygon exists. Fekete [55] studied the problem when the input is a set of translates of a convex regular k-gon (for fixed k), he stated a polynomial-time algorithm for this case. Rappaport [100] considered another variant of the problem. A set of n line segments in a fixed number of orientations is given and one wants to find a convex polygon with minimum perimeter that weakly intersects this set, meaning each line segment is either intersected by the boundary of the polygon or lies completely inside the polygon. Rappaport presented an $\mathcal{O}(n \log n)$ algorithm for this problem.

Besides these results, there has been nearly no progress on Tamir's problem in the last 25 years.

Transversal problems are interesting from various perspectives. As already mentioned there has been a lot of work on the combinatorial aspect of transversals, like the complexity of a set of transversals or the order induced by the stabbers. Also the algorithmic part was widely studied. Possible applications for transversals are in curve reconstruction, for line simplification, or in motion planning.

In some of these applications it is natural to consider convex transversals as generalizations of line transversals, like for instance for curve reconstruction or function approximation.

In Chapter 1 we study Tamir's problem for different kind of input sets, i.e., line segments, squares and regular polygons. We show that Tamir's problem is NP-hard for these variants and we also consider an optimization version.

In Chapter 2 we study the problem of stabbing a set of geometric objects with the vertices of a regular polygon. This problem is closely related to the approximate symmetry detection problem [43, 73, 74]. Later on we show that it is not crucial that

the stabbing polygon is regular, and generalize our algorithm to scaled, translated, rotated copies of a given polygon. To the best of our knowledge, this problem was not studied so far.

In Chaper 3 we consider the problem of stabbing two sequences of geometric objects with two point sequences such that the discrete Fréchet distance between these point sequences is as small as possible. This problem can also be considered as a *shape matching* problem; for a survey on shape matching see [16, 112].

Chapter 1

Convex Stabbers

Consider a finite set of geometric objects in the plane. We call this set *stabbable* if there exists a convex polygon whose boundary intersects every object. The boundary is then called *convex transversal* or *convex stabber*. As mentioned in the introduction of this part, the problem of finding a convex stabber was originally proposed by Tamir in 1987 [109] and only restricted versions of the problem have been solved so far.

In Section 1.1 we show that Tamir's problem is NP-hard when the geometric objects are line segments. The problem remains NP-hard when the objects are squares or disjoint simple polygons. This is proven in Section 1.2 and Section 1.3. In Section 1.4 we study an optimization version of the problem: Given a set of geometric objects in the plane, how many of these objects can be simultaneously stabbed by the boundary of a convex polygon? We prove this problem to be APX-hard if the set of geometric objects is a set of segments or squares.

The results of Section 1.1 and Section 1.2 are part of research that has been published in [22, 23], together with additional results by Arkin, Dieckmann, Knauer, Mitchell, Polishchuk, and Yang. The results of Section 1.3 and Section 1.4 have been published in [104]. All results in this chapter have been exclusively found by the author of this thesis.

Notation. We say a convex stabber stabs or *traverses* the given objects. The line segment connecting two points p and q is denoted by $\overline{pq}$.

Closed stabbers vs. terrains. As mentioned in [23], in many applications it might be more relevant to consider stabbers that represent the graph of a function. Possible applications for this problem are, e.g., in statistics or function approximation. Thus, there are cases where it is more useful to search for a convex *x-monotone* stabber, that is, the stabber intersects every vertical line in at most one point. Since an x-monotone polygonal chain is also called *1.5-dimensional terrain* (see, e.g., [80]), we will say that we search for a convex 1.5-dimensional terrain stabber, short a *con-*

vex terrain stabber. Notice that a convex terrain stabber is a part of the boundary of a convex polygon.

It turns out, that finding a convex terrain stabber is a special case of finding a convex stabber. One can see this by placing a point far below the input set and noticing that there exists a convex stabber for this set if and only if there exists a convex terrain stabber for the input set, see Figure 1.1 right.

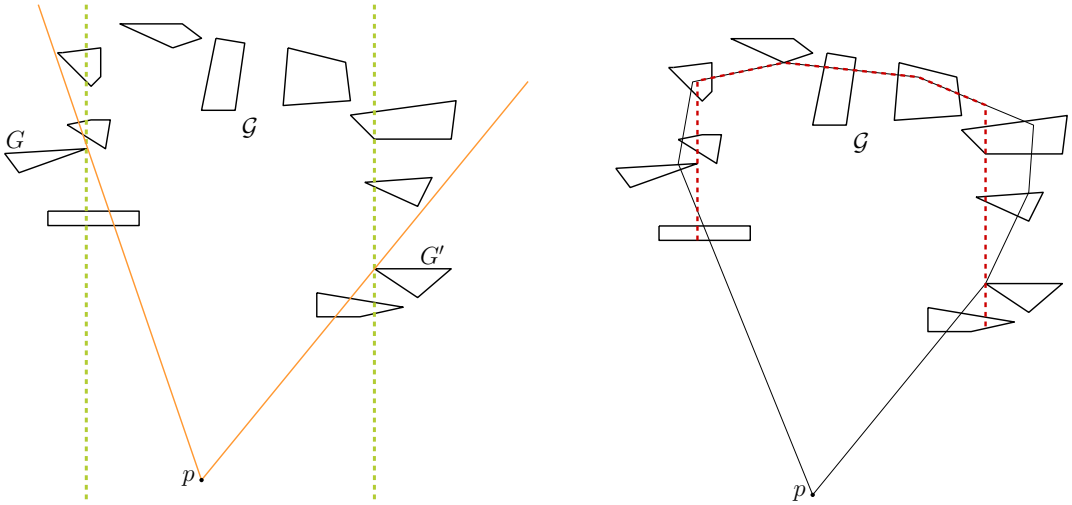


Figure 1.1: From left to right: Let $\mathcal{G}$ be the input set. The tangent between p and G and the tangent between p and G' are marked; also the vertical lines through the rightmost point of G and the leftmost point of G' are depicted. On the right: A convex stabber for $\mathcal{G} \cup \{p\}$ gives immediately a convex terrain stabber for $\mathcal{G}$ (and also the other way round).

The remaining question is how to construct this point; in the following the point is denoted by p. Let $\mathcal{G}$ be a set of geometric objects in the plane. Let $G \in \mathcal{G}$ be the object containing the leftmost rightmost point and let $\mathcal{G}_G$ be the set of objects in $\mathcal{G}$ that are intersected by a vertical line through the rightmost point of G. Let $G' \in \mathcal{G}$ be the object containing the rightmost leftmost point and let $\mathcal{G}_{G'}$ be the objects in $\mathcal{G}$ that are intersected by the vertical line through the leftmost point of G'. Then p has to fulfill the following conditions: The rightmost tangent between p and G must not intersect any object in $\mathcal{G} \setminus \mathcal{G}_G$. Similarly, the leftmost tangent between p and G' must not intersect any object of $\mathcal{G} \setminus \mathcal{G}_{G'}$. (See Figure 1.1 left.)

A convex stabber for the set $\mathcal{G} \cup \{p\}$ gives a convex terrain stabber for $\mathcal{G}$ in the following way: Either the part of the convex stabber traversing the objects of $\mathcal{G}$ is already a convex terrain stabber for $\mathcal{G}$. Otherwise, take the upper part of the convex stabber lying between the two vertical lines through the rightmost point of G and the leftmost point of G'. This part of the convex stabber combined with the two vertical rays from the endpoints (i.e., the intersection points of the vertical lines and

the convex stabber) directed downwards represents a convex terrain stabber for $\mathcal{G}$, see Figure 1.1 right.[1]

On the other hand, a convex terrain stabber for $\mathcal{G}$ gives us immediately a convex stabber for $\mathcal{G} \cup \{p\}$: A convex terrain stabber for $\mathcal{G}$ has to stab G and G'. Hence, the leftmost point of this stabber lies on or to the left of the vertical line through the rightmost point of G and the rightmost point lies on or to the right of the vertical line through the leftmost point of G'. Connecting both endpoints of the convex terrain stabber with the point p gives a convex stabber for $\mathcal{G} \cup \{p\}$.

Knowing these conditions of the point p, it is easy to compute p: Take the vertical line through the rightmost point of G and rotate it slightly in counterclockwise direction, keeping it tangent to G. The line is rotated by a very small angle, such that it does not intersect any object in $\mathcal{G} \setminus \mathcal{G}_G$ and all these objects lie to the right of this line (while G still lies to the left of it). Similarly, rotate the vertical line through the leftmost point of G' slightly in clockwise direction, such that it does not intersect any object in $\mathcal{G} \setminus \mathcal{G}'_G$ and all these objects lie to the left of this line while G' lies to the right of it. The intersection point of these two lines is the point p.

We achieve the following result.

Lemma 1.0.1. *Given a set $\mathcal{G}$ of objects in the plane, we can compute a point p in polynomial time such that: there exists a convex terrain stabber for $\mathcal{G}$ if and only if there exists a convex stabber for $\mathcal{G} \cup \{p\}$.*

This shows that finding a convex terrain stabber is a special case of finding a convex stabber. And so, if it is NP-hard to find a convex terrain stabber, it follows immediately that it is NP-hard to find a convex stabber.

The hardness proofs given in the following sections hold actually for both cases: finding a convex stabber or a convex terrain stabber.

1.1 Convex (Terrain) Stabbers for Line Segments

In this section we prove that Tamir's problem is NP-hard if the set of objects is a set of line segments. That is, the problem we consider is the following:

> Given a set of line segments in the plane, decide whether there exists a convex (terrain) stabber that stabs every line segment.

We prove this problem to be NP-hard. This result was independently proven by Arkin, Mitchell, Polishchuk, and Yang. For the NP-hardness proof we use a reduction from *3SAT*. 3SAT is a decision problem from Boolean logic: The input is

[1] This actually proves that there is a convex *weakly* x-monotone 1.5-dimensional terrain stabbing $\mathcal{G}$. Instead of using vertical lines through the rightmost point of G and the leftmost point of G', we rotate them by a suitable small angle in opposite directions. Then the same constructions shows that there is a convex *strongly* x-monotone 1.5-dimensional terrain stabbing $\mathcal{G}$.

a Boolean expression in *conjunctive normal form*, that is a conjunction of clauses where a clause is a disjunction of literals, and each clause contains at most 3 *literals*. A literal is either a variable or the negation of a variable. The question is, given such an expression ϕ, does there exist an assignment that satisfies all clauses in ϕ? 3-SAT is well known to be NP-complete [75]. (We actually use a reduction from the special variant of 3-SAT where each clause contains *exactly* 3 literals. This version of 3-SAT is also NP-complete [60].)

Given a 3SAT formula ϕ with n variables and m clauses. In the following we show how to build a set of line segments L such that there exists a convex (terrain) stabber for L if and only if ϕ is satisfiable. We proceed similarly to the techniques used in [86] where the basic idea of the construction is to place variable and clause gadgets on a circular arc as shown in Figure 1.4. We start with a half circle with unit radius and divide half of it into n arcs of equal size and the other half into m arcs of equal size. Later on, each gadget will be placed along such an arc. (Actually, the arcs do not have to have the same length, we choose them to be equal for convenience.) In the following, we denote this half circle by K.

Variable gadgets. A variable gadget consists of three points (degenerate line segments of length 0) and one line segment, see Figure 1.2. A variable gadget is fit into an arc as follows:

- The two non-middle points are put on the arc at distance ϵ away from their closest endpoints of the arc, for some sufficiently small $\epsilon > 0$.
- The middle point and the segment lie inside the circle.
- The left non-middle point, the middle point and the right endpoint of the segment are aligned. Also, the right non-middle point, the middle point and the left endpoint of the segment are aligned.

There are two ways to traverse the gadget by a convex (terrain) stabber. They differ in the order in which the middle point and the segment are traversed: One way traverses the left non-middle point, then the middle point, the right endpoint of the segment and finally the right non-middle point (the dashed path in Figure 1.2). This way corresponds to setting the variable to False and is called the *False path*. The other way traverses the left non-middle point, then the left endpoint of the segment, the middle point and then the right non-middle point (the dashed-dotted path in Figure 1.2). This corresponds to setting the variable to True and is called the *True path*.

Notice that the True path and the False path are the only two possibilities to traverse the gadget with a convex (terrain) stabber while stabbing all segments of the gadget. This is because the middle point lies inside the convex hull of the two non-middle points and any point on the segment that is not one of its endpoints.

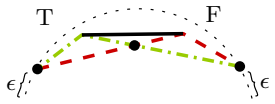


Figure 1.2: There are two ways to traverse the gadget: The dashed-dotted path corresponds to setting the variable to True, the dashed path corresponds to setting the variable to False. The dashed and the dashed-dotted segments and the dotted circular arc are not part of the construction.

All variable gadgets are placed next to each other along the half circle K. Recall that we divided half of K into n arcs of equal size; in each such arc we place one variable gadget and, hence, each gadget is placed in an arc of length $1/(4n)$ of a unit circle, see Figure 1.4. We call this part of K the *variable arc*.

Recall that the two non-middle point of each variable gadget lie on the arc. A convex (terrain) stabber has to stab all these points and, hence, it is (partially) aligned to the convex hull of these points. Thus, a convex (terrain) stabber visits the gadgets in the order as they appear on the arc, assigning truth values to the variables in turn in each gadget.

Clause gadgets. A clause gadget is constructed in a way similar to a variable gadget. It consists of two points (segments of length 0) and a segment. Each clause gadget is fit into an arc. The two points are put on the arc at distance ϵ (for some sufficiently small $\epsilon > 0$) away from their closest endpoints of the arc and the segment lies inside the circle. The only way to traverse the gadget is to visit the first point, then the segment and then the second point – the only flexibility is where to touch or intersect the segment.

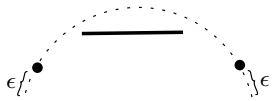


Figure 1.3: A clause gadget.

All clause gadgets are placed next to each other on the half circle K. Recall that half of K was divided into m arcs of equal size and we place a clause gadget in each such arc (thus, each clause gadget is placed into an arc of $1/(4m)$ of a unit circle). We call this part of K the *clause arc*.

Since the points of each clause gadget lie on the arc, and a convex (terrain) stabber has to stab all these points, the only way to traverse these gadgets with a convex (terrain) stabber is to visit them one by one in the order as they appear along the arc.

Recall that the variable arc and the clause arc lie next to each other on K.

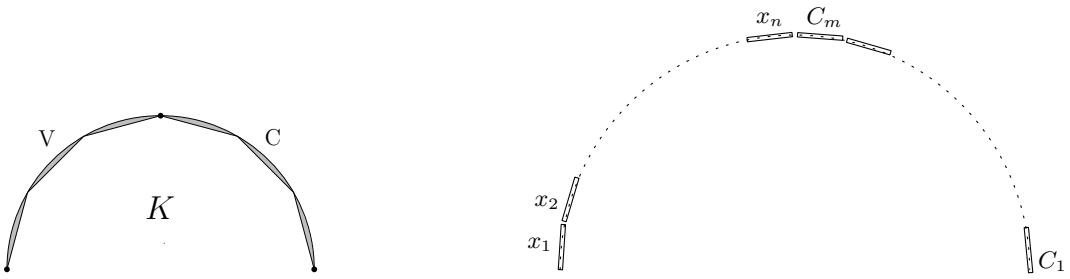


Figure 1.4: The half circle K and the placement of the variable and clause gadgets. From left to right: V and C mark the placement of the variable gadgets and the clause gadgets, respectively. Each gadget is placed in one of the grey regions. On the right: The variable gadgets (marked with x_i) are placed on an arc of one quarter of a unit circle; the clause gadget (marked with C_i) are also placed on an arc of one quarter of a unit circle, next to the variable gadgets.

Observation 1.1.1. *A convex (terrain) stabber stabbing all segments of the variable and clause gadgets has the following properties:*

- *It stabs all points on the arc K and, hence, it cannot stab any object lying inside the convex hull of these points or above K.*
- *It has to take the True or the False path in each variable gadget and stabs the segment of each clause gadget.*

We still have to connect a variable gadget to a clause gadget whenever the variable appears in the clause. For this, we construct some more segments, which we call the *connector segments.*

Connector segments. We place $3m$ connector segments. Each connector segment consists of a simple line segment. A connector segment connects a variable gadget to a clause gadget if the variable appears in the clause. The placement of the endpoints of the connector segments within the variable gadgets is as follows: Suppose the variable appears unnegated in the clause. Then the connector segment touches the True path of the gadget and it does not intersect the False path, see Figure 1.5. If the variable appears negated in the clause the segment touches the False path and not the True path. This ensures that a stabber can either stab the connector segments touching the True path or the segments touching the False path, but never both. The placement of the endpoints of the connector segments within the clause gadget is a bit more involved. We want to ensure that a convex (terrain) stabber can stab any two of these connector segments in each clause gadget, but not all three. Figure 1.5 shows the placement of the connector segments' endpoints and the three possible ways to traverse the gadget.

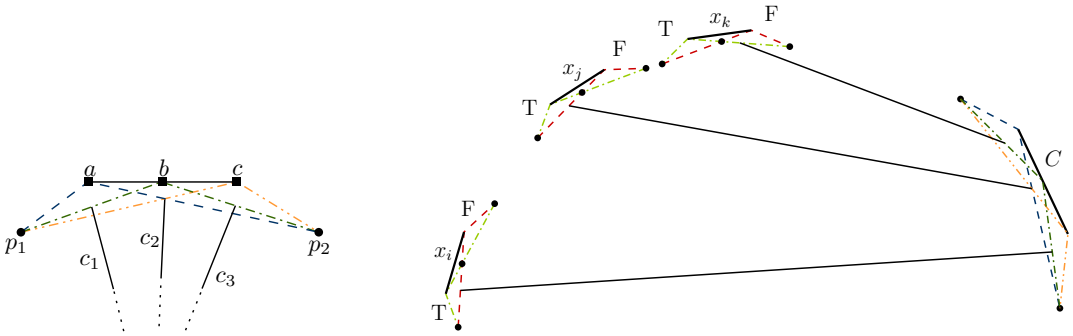


Figure 1.5: From left to right: There are three ways to traverse the clause gadget. Each of them stabs two of the three connector segments. It is possible to stab any subset of two out of the three segments, but never all three. (The points a, b, c are not part of the construction.) On the right: A clause $C = \bar{x_i} \vee \bar{x_j} \vee x_k$ is shown. All not solid segments are not part of the construction.

In the following we refer to the labels given in Figure 1.5. The points p_1, p_2 and the line segment $\overline{ac}$ represent the clause gadget. The midpoint of $\overline{ac}$ is denoted by b. The endpoint of one connector segment is chosen to be the intersecting point of the line through p_1 and c and the line through p_2 and a. The other two endpoints of the connector segments are chosen in the following way: One lies on the *open* segment between p_1 and the intersection point of $\overline{ap_2}$ and $\overline{bp_1}$, the other lies on the open segment between p_2 and the intersection point of $\overline{bp_2}$ and $\overline{cp_1}$.

The following lemma shows that this placement fulfills the required conditions.

Lemma 1.1.2. *In each clause gadget, a convex (terrain) stabber can stab any two of the three connector segments, but not all three.*

Proof. A convex (terrain) stabber traversing a clause gadget has to stab p_1, p_2 and the segment $\overline{ac}$. First notice that a convex (terrain) stabber that intersects the segment $\overline{ac}$ in more than one point, stabs less than two connector segments. So we assume in the following that the stabber intersects $\overline{ac}$ in exactly one point. Then, there are 5 combinatorially different ways to traverse this gadget:

1. The convex (terrain) stabber intersects the segment $\overline{ac}$ in a. Then the stabber can stab c_2 and c_3, but not c_1.

2. The convex (terrain) stabber intersects the segment $\overline{ac}$ in the interval (a, b). The stabber can stab neither c_1 nor c_2, it can only stab c_3.

3. The convex (terrain) stabber intersects the segment $\overline{ac}$ in b. Then the stabber can stab c_1 and c_3, but not c_2.

4. The convex (terrain) stabber intersects the segment $\overline{ac}$ in the interval (b, c). The stabber can stab c_1, but neither c_2 nor c_3.

5. The convex (terrain) stabber intersects the segment $\overline{ac}$ in c. The stabber can stab c_1 and c_2, but not c_3.

This proves that a convex (terrain) stabber can stab any two of these connector segments in a clause gadget, but never all three. □

Correctness. Assume first that the 3SAT instance is satisfiable and so there exists a satisfying assignment. The convex (terrain) stabber traverses the variable gadgets according to this assignment. The clause gadgets are traversed in the following way: The stabber can stab at most two connector segments in each clause gadget. Since in each clause at least one variable is satisfied, the stabber can omit the connector segment connecting the satisfied variable to the clause and stab the other two connector segments. Hence, the stabber is convex and stabs all segments.

Conversely, assume there is a convex (terrain) stabber that stabs all segments. From Observation 1.1.1 follows that the connector segments have to be stabbed either in a variable or a clause gadget and that the stabber takes the True or the False path of each variable gadget. Set the variables according to how the stabber traverses the variable gadgets. If it takes the True path we set the variable True, otherwise we set the variable False. This is a satisfying assignment for the 3SAT instance: The setting is consistent because the stabber omits at least one connector segment in each clause gadget. Thus, these omitted segments have to be stabbed in the variable gadgets. And there the stabber can either take the True path or the False path, but not both.

It remains to show that the construction can be done in polynomial time. Each variable and each clause gadget consists of a constant number of points and line segments. We have to show that the coordinates of the endpoints of the segments and the coordinates of all other constructed points are rational and bounded by a polynomial function of the input size (thus, the number of bits needed to represent these points are polynomially bounded). This can be shown in a very similar way to [86]: The rational points are dense on a unit circle. More precisely, for each $t \in \mathbb{Q}$, the point $(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2})$ is rational and lies on the unit circle (see, e.g., [71, 110]). So every arc of a circle, no matter how small, contains infinitely many rational points.

Recall that the gadgets are placed on a half circle; each variable gadget is fit into an arc of size $(1/4n)$ and each clause gadget into an arc of size $(1/4m)$. The two non-middle points of a variable gadget lie on the half circle, some ϵ-distance away from the vertices of a $4n$-gon, defined by the circle. For each non-middle point a sufficient $\epsilon \in \mathcal{O}(1/n)$ can be found such that this point is rational and polynomial bounded by n (notice that the ϵ-value can be different for each point). The two points of each clause gadget also lie on the half arc, an ϵ-distance away from the vertices of a $4m$-gon. Again, ϵ can be chosen in a sufficient way for each point such that these points are rational and polynomial bounded by m and $\epsilon \in \mathcal{O}(1/m)$. Hence, all these

points are rational and they depend polynomially on n or m. (We can also argue in a different way: we can construct $2n$ points on quarter of a unit circle that are at least $\pi/(8n)$ radians apart with coordinates that are polynomial bounded by n, see [86]. These points are then the non-middle points of the variable gadgets. The non-middle points of the clause gadgets can be constructed analogously.)

To see that the remaining points of the construction are also rational and bounded by the input size, we can construct a constant size grid parallel to the segment defined by the two non-middle points of the variable gadgets or the segment defined by the two points of the clause gadgets. All other points can be chosen to be such grid points.

Thus, these polynomial size representations of all points can be found in polynomial time.

(Actually, there is an even simpler argument why all points of the construction are rational. The points in our construction can be perturbed by a very small $\delta > 0$ in order to become rational and polynomial bounded by the input size and the crucial properties of the construction still hold. Notice that, e.g., the endpoints of connector segments can be slightly perturbed. The only crucial property is that they intersect only one of the possible paths to traverse the clause/variable gadgets. Hence, the placement is not fixed.)

Thus, we have showed the following

Theorem 1.1.3. *Let L be a set of line segments in the plane. It is NP-hard to decide whether there exists a convex (terrain) stabber for L.*

1.2 Convex (Terrain) Stabbers for Regular Polygons

We show now that the construction above can be adapted to sets of squares. Each segment in the construction is replaced by a square in a carefully chosen way, maintaining the properties of the construction. It follows that Tamir's problem remains NP-hard when the objects are squares. Later on, we show that this construction can even be generalized to sets of simple regular polygons (under a certain assumption how these regular polygons are represented). So we start with the following problem:

> Given a set of squares in the plane, decide whether there exists a convex (terrain) stabber that stabs every square.

We prove this problem to be NP-hard by generalizing the above construction for segments to squares. (Hence, the proof is again by a reduction from 3SAT; n denotes the number of variables and m the number of clauses in the 3SAT formula.)

Here we start with an arc of a quarter of a unit circle; this arc will be denoted by K. We divide half of K in n arcs of equal size and the other half in m arcs of equal size. Later on, each gadget will be placed along such an arc.

Variable gadgets. Recall the construction of a variable gadget in the hardness proof for segments. Here the gadget is constructed similarly: The points are placed in exactly the same way (now they are considered as squares of zero area). The segment is replaced by a square in the following way: Each edge of the square has the same length as the segment. The edge which is closest to the points is placed in exactly the same way as the segment before. We call this edge the *basic* edge, see Figure 1.6. Summarizing, the gadget consists of three points (squares of area 0) and a square.

The whole gadget is fit into an arc as before: The two non-middle points are put on the arc at distance ϵ away from its closest endpoint of the arc, for some sufficiently small $\epsilon > 0$. The middle point and the basic edge lie inside the circle; the other edges of the square might lie outside the circle. The left non-middle point, the middle point and the right endpoint of the basic edge are aligned. Also, the right non-middle point, the middle point and the left endpoint of the basic edge are aligned.

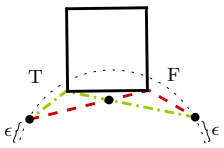


Figure 1.6: A variable gadget: There are two ways to traverse it. The dash-dotted path corresponds to setting the variable to True, the dashed path to setting the variable to False.

There are two ways to traverse a gadget, one corresponds to setting the variable to True and the other corresponds to setting the variable to False.

All variable gadgets are placed next to each other on the quarter circle K, see Figure 1.9. Recall that we divided half of K into n arcs of equal size and we fit a variable gadget in each such arc (thus, a variable gadget is placed into an arc of $1/(8n)$ of a unit circle). We call this half of K the *variable arc*. The only way to traverse these gadgets with a convex (terrain) stabber is to visit them in the order as they appear on the arc, one after the other.

Clause gadgets. A clause gadget consists of two points and a square. Recall the construction of a clause gadget for line segments: We place the points in exactly the same way and the segment is replaced by a square as in the construction for the variable gadget, see Figure 1.7. Each gadget is fit into an arc.

The clause gadgets are placed next to each other on K, more precisely, they are fit on the half of K that is divided into m arcs of equal size. Hence, each gadget is fit into an arc of $1/(8m)$ of a unit circle. We call this part of K the *clause arc*. Recall that the clause arc and the variable arc lie next to each other on K, see Figure 1.9.

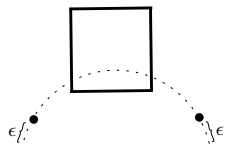


Figure 1.7: A clause gadget.

Connector squares. Instead of connector segments, we place now $3m$ connector squares. Each of them connects a variable gadget to a clause gadget whenever the variable appears in the clause. The crucial part is to place the connector squares in the right way. For this, we place one edge of the square in exactly the same way as we placed the connector segments in the construction above (Figure 1.5). We call this edge the *connector edge*. Hence, the endpoints of this connector edge within the variable gadget are placed as follows: If the variable appears unnegated in the clause, the endpoint lies on the True path of the clause gadget and does not touch the False path. If the variable appears negated the endpoint lies on the False path. The placement for the endpoints of the connector edge within the clause gadgets are shown in Figure 1.8. It is ensured that a convex (terrain) stabber can stab any two of these connector edges inside the clause gadget, but not all three. The argument is the same as in the proof for line segments (Lemma 1.3.1).

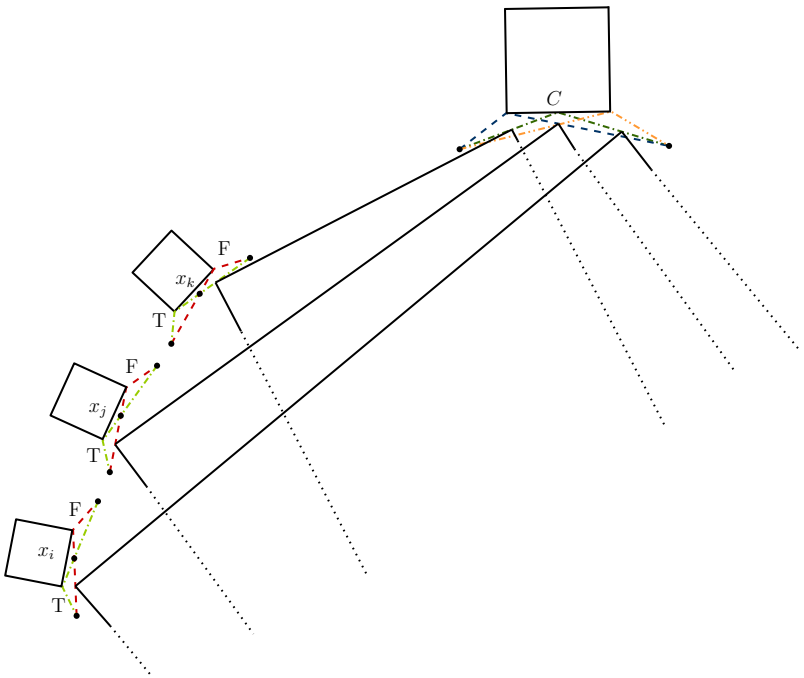


Figure 1.8: A clause $C = \bar{x_i} \vee \bar{x_j} \vee x_k$ is shown. There are three ways to traverse the clause gadget. It is possible to stab any subset of two out of the three connector squares, but never all three.

It remains to show that a convex (terrain) stabber cannot stab the third connector square anywhere else except in the corresponding variable gadget. First recall that the variable arc and the clause arc both occupy an arc of 1/8 of a unit circle; they lie next to each other on the quarter circle K. Consider a connector square. It connects a variable gadget to a clause gadget; and so it lies completely inside the circle defined by K. Thus, it is not possible that such a square intersects any other gadget; by intersecting a gadget we mean that the convex hull of the arc, in which the gadget is placed, is intersected. To ensure that a connector square is not intersected anywhere else (outside all gadgets) we place two more points in the construction, we call these points the *enforcer* points. Let the circle defined by K be divided in 4 equal parts by a horizontal and a vertical line. Let K be the upper left quarter circle. Then one of the enforcer points is placed on the upper horizontal tangent to the circle and the other one on the left vertical tangent to the circle. The line defined by these points does not intersect the circle, Figure 1.9 shows such a placement for these points. Now a convex (terrain) stabber also has to stab these enforcer points, and hence the only possible way to stab a connector square by this stabber is to stab its connector edge in the corresponding clause or variable gadget.

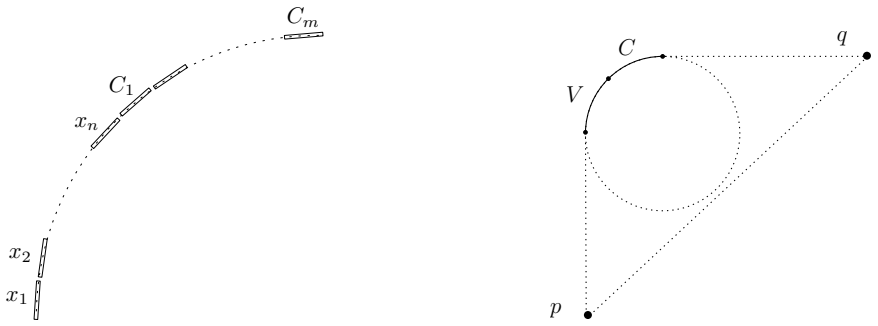


Figure 1.9: From left to right: We start with an arc of 1/4 of a unit circle which is denoted by K: The variable gadgets are placed next to each other on half of K, the clause gadgets are placed on the other half. On the right: V and C mark the placement for the variable and the clause gadgets. The points p and q are the enforcer points that ensure that the connector squares can either be intersected at variable or clause gadgets but nowhere else.

Correctness. Assume first that the 3SAT instance is satisfiable. Then the convex (terrain) stabber traverses the variable gadget according to a satisfying assignment. In each clause gadget the stabber has to omit one of the connector squares, let this be the connector square to the satisfying variable. The stabber stabs this square already in the corresponding variable gadget and the other two in the clause gadget. Thus, the stabber stabs all squares.

Conversely, assume there is a convex (terrain) stabber that stabs all squares.

Set the variables according to how the stabber traverses the variable gadget. If it takes the True path we set the variable True, otherwise False. We argue that this is a satisfying assignment for the 3SAT instance. The setting is consistent since the stabber has to omit at least one connector segment in each clause gadget. And hence these omitted segments have to be stabbed in the variable gadgets. And there the stabber can either take the True path or the False path, but not both.

The construction can be done in polynomial time. Each variable and each clause gadget consists of a constant number of points and squares and all constructed points are rational and bounded by a polynomial function of the input size. This can be shown in the same way as in the construction for segments: All constructed points (besides the two enforcer points) and one edge of each square are placed in the same way as before. The only difference is that now they lie on a quarter circle and not a half circle (but as we already mentioned before: every arc of a circle, no matter how small, contains infinitely many rational points). So we know that all these points are rational. The two enforcer points can easily be chosen to be rational; they can be perturbed slightly in order to become rational. The remaining part is to prove that *all* vertices of the squares are rational points: Each square has two rational vertices, namely the two vertices of the edge that is placed in the same way as the segment before. But then it follows immediately that the other two vertices of the square are also rational. So all constructed points are rational points of polynomial complexity.

Thus, we have shown the following

Theorem 1.2.1. *Let S be a set of squares in the plane. It is NP-hard to decide whether there exists a convex (terrain) stabber for S.*

Generalization. The problem of finding a convex (terrain) stabber for a set of simple regular k-gons (for any $k > 2$) is also NP-hard. This can be shown by adapting the above proof. We just replace the squares by k-gons; this is done in a way similar to replacing the segments by squares. We replace the square by a k-gon by maintaining the crucial edge of the square and completing it to a k-gon; Figure 1.10 shows an example for replacing the square in a variable gadget. We start now with an arc of $(1/k)$ of a unit circle and we fit the variable gadgets into an arc of $1/(2kn)$ of a unit circle and the clause gadgets are fit into an arc of $1/(2km)$ of a unit circle. This ensures that the connector k-gons lie completely inside the circle defined by this arc of $1/k$ of a unit circle. Also the enforcer points have to be adapted.

This construction can only be done in polynomial time if we use a certain representation for the simple regular k-gons: Each regular k-gon is represented by two points and an orientation. The two points are two adjacent vertices of the k-gon and the given order determines the remaining vertices of the k-gon. This representation is crucial for the construction as in general the vertices of regular k-gons are not rational.

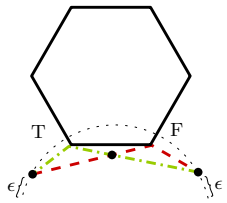


Figure 1.10: A variable gadget for a regular polygon.

With our assumption it is clear that all points of the construction are rational; each simple regular k-gon is given by the edge that is constructed in the same way as the segments in the proof for line segments. Hence, we can use similar arguments as for this construction in order to show that all points are rational.

Theorem 1.2.2. *Let P a set of simple regular k-gons in the plane where $k > 2$. It is NP-hard to decide whether there exists a convex (terrain) stabber for P.*
Remark*: A regular k-gon is represented by two points and an orientation.*

1.3 Convex (Terrain) Stabbers for Disjoint Polygons

We show that the problem of finding a convex (terrain) stabber remains NP-hard for a set of disjoint (non convex) polygons. More precisely, it is already hard to find a convex (terrain) stabber for a set of disjoint bends, where a bend consists of two line segments with a common endpoint. We consider the following problem.

> Given a set of disjoint bends in the plane, decide whether there exists a convex (terrain) stabber that stabs every bend.

We reduce from *planar, monotone 3SAT*. A 3SAT instance can be considered as a bipartite graph: The graph has a vertex for each clause and each variable and an edge between a variable vertex and a clause vertex if the variable occurs in this clause. 3SAT remains NP-hard even when this graph is planar [83]; this problem is called *planar* 3SAT. A *monotone* instance of 3SAT is an instance where each clause has either only positive or only negative variables. In the following we call a clause that contains only positive variables a *positive clause* and a clause that contains only negative variables a *negative clause*. De Berg and Khosravi [41] proved planar, monotone 3SAT to be NP-hard even when a *monotone rectilinear representation* is given. In a monotone rectilinear representation the variable and clause gadgets are represented as rectangles. All variable rectangles lie on a horizontal line. The edges connecting the clause gadgets to the variable gadgets are vertical line segments and no two edges cross. All positive clauses lie above the variables and all negative clauses lie below the variables. See Figure 1.11 for an example of a monotone rectilinear representation of a planar 3SAT instance.

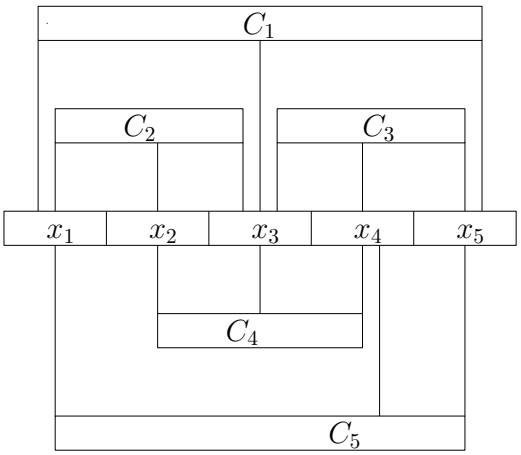


Figure 1.11: A monotone rectilinear representation of the 3SAT instance $C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ where $C_1 = x_1 \vee x_3 \vee x_5$, $C_2 = x_1 \vee x_2 \vee x_3$, $C_3 = x_3 \vee x_4 \vee x_5$, $C_4 = \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}$, and $C_5 = \overline{x_1} \vee \overline{x_4} \vee \overline{x_5}$.

Given a monotone rectilinear representation ϕ of a 3SAT instance, we construct a set of bends B such that there exists a convex (terrain) stabber for B if and only if ϕ is satisfiable.

Let m be the number of clauses and n be the number of variables. Let the number of positive clauses and negative clauses be m_1 and m_2 respectively.

We start with a half circle of unit radius; this half circle will be denoted by K. We divide half of K in $4m_1$ arcs of equal size and the other half in $4m_2$ arcs of equal size. Later on, all gadgets belonging to positive clauses will be placed along the quarter circle divided in $4m_1$ parts, and all gadgets belonging to negative clauses along the other quarter circle.

Variable gadgets. The variable gadgets are constructed in the same way as in the proof for line segments, see Figure 1.2. Hence, there are two ways to traverse them, one corresponds to setting the variable to True, the other to setting the variable to False. Again, the variable gadgets are fit into a circular arc. A variable gadget is placed into an arc of $1/(16m_1)$ of a unit circle if it represents a variable that appears in a positive clause; we call this a *positive variable gadget*. Similarly, a *negative variable gadget* is a gadget representing a variable that appears in a negative clause. Each negative variable gadgets is placed into an arc of $1/(16m_2)$ of a unit circle.

In the construction, we use one gadget for each occurrence of a variable in a clause.

Clause gadgets. The clause gadgets are constructed in the same way as in the proof for line segments, see Figure 1.3. Similarly to the variable gadgets the clause gadgets are fit into a circular arc. Each positive clause gadget is fit into an arc of

$1/(16m_1)$ of a unit circle. Each negative clause gadget is fit into an arc of $1/(16m_2)$ of a unit circle.

Positive arc. We place the gadgets representing a positive variable or a positive clause next to each other on K; more precisely, we place these gadgets on the half of K which we divided into $4m_1$ arcs of equal size. We place a positive variable or a positive clause gadget in each such arc. The gadgets have to be placed in a specific order. Consider the upper part of the monotone rectilinear representation (i.e., the one representing the positive clauses). The clauses and their corresponding variables are connected via edges. We use a variable gadget for each edge. In the rectilinear representation the edges are sorted from left to right. We maintain this order and place the gadgets according to it, starting at the bottom of the arc. If a variable gadget corresponds to an edge that connects a clause to its middle variable, we place the corresponding clause gadget right after this variable gadget. Hence, each clause gadget is placed to the right of the gadget representing its middle variable, see Figure 1.12. Notice that all gadgets that represent the same variable lie next to each other, only a clause gadget can lie between them.

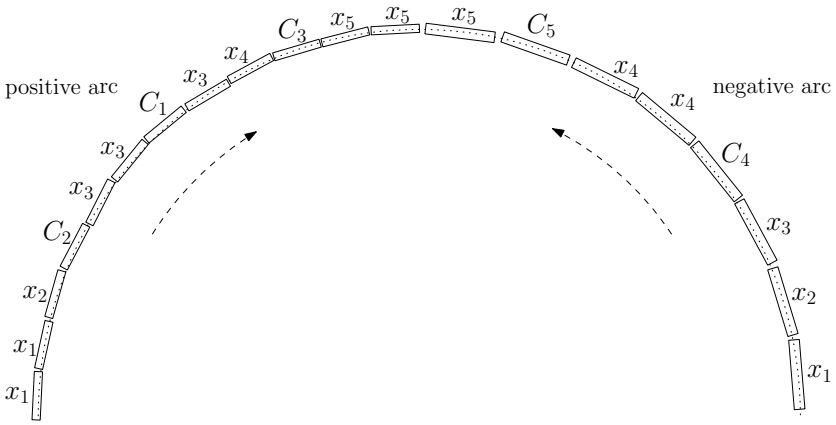


Figure 1.12: The placement of the gadgets for the instance in Fig. 1.11 is shown. The arrows show the direction of the ordering.

Negative arc. We place the gadgets representing a negative variable or a negative clause next to each other on the other half of K; the one we divided into $4m_2$ arcs of equal size. The gadgets are ordered in the same way as for the positive arc, but this time from right to left. Notice that here a clause gadget is placed to the left of the gadget representing its middle variable. The negative and the positive arc lie next to each other on K, see Figure 1.12.

Variable connectors. To ensure that a stabber has to traverse all gadgets representing the same variable in a consistent manner, we place $3m$ variable connectors.

All variable gadgets that represent the same variable are connected via segments in a circular manner. The segment touches the True path of one gadget and the False path of the next gadget (see Figure 1.13). If a stabber traverses the True path of one gadget and the False path of the next gadget, it will never stab the corresponding variable connector segment. Thus, a convex (terrain) stabber traverses all gadgets representing the same variable in a consistent manner.

Recall that on both the positive and the negative arc all variable gadgets representing the same variable lie next to each other (only a clause gadget can lie between them). Hence, these segments are all disjoint and they also do not intersect any other bend of the construction. In total, we place one segment for each variable gadget.

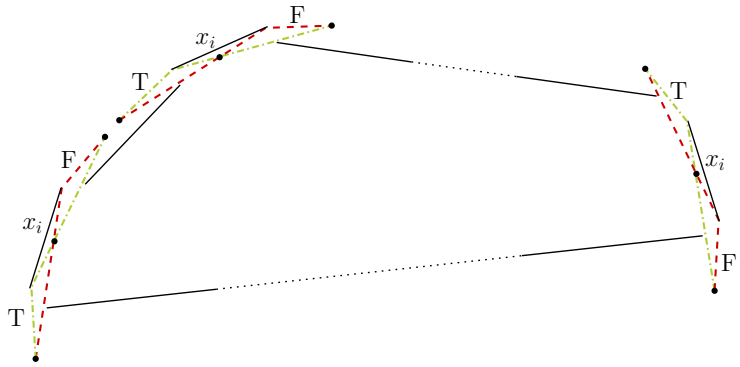


Figure 1.13: The variable connectors are shown. These connectors ensure that each variable gadget that represents the same variable is traversed in the same way: either the stabber traverses the True path or the False path.

Connector bends. We place $3m$ more bends in order to connect a clause gadget with its variables. (Recall that there is a variable gadget for each occurrence of a variable in a clause and they are placed in a specific way, corresponding to the monotone rectilinear representation. Thus, each clause gadget has its own 3 variable gadgets to which it has to be connected.) The connectors either lie inside the circle or outside. We call them *inner connectors* and *outer connectors*, respectively. An inner connector can be a straight line segment whereas an outer connector has to be a bend. The remaining parts of the construction are explained for positive clauses. Negative clauses can be handled similarly. Each clause gadget has two outer connectors and one inner connector. The inner connector connects the clause gadget to the gadget representing its middle variable. Note that this gadget is placed next to the clause gadget in the construction (to its left). The other two variables that occur in this clause are connected via outer connectors. One endpoint of each connector lies within the variable gadget as follows: the segment touches the True path through the gadget and does not intersect the False subpath. The placement of the connector endpoints within the clause gadget is more involved. We have to ensure that a convex (terrain)

stabber can stab any two out of the three connector bends in each clause gadget, but never all three. Hence, it has to stab the third bend at the corresponding variable gadget.

In every clause gadget, the endpoints of these connectors look the same – see Fig. 1.14. In the following we refer to the labels given in Figure 1.14. The points p_1, p_2 and the line segment $\overline{ac}$ represent the clause gadget. The midpoint of $\overline{ac}$ is denoted by b. The endpoint of the inner connector is chosen to be the intersection of $\overline{p_1 c}$ and $\overline{ap_2}$. The endpoints of the other two outer connectors are chosen in the following way: One lies on the *open* segment between p_1 and the intersection point of $\overline{p_1 b}$ and $\overline{ap_2}$, the other lies on the open segment between p_2 and the intersection point of $\overline{p_1 c}$ and $\overline{bp_2}$. The following lemma shows that this placement fulfills the required conditions.

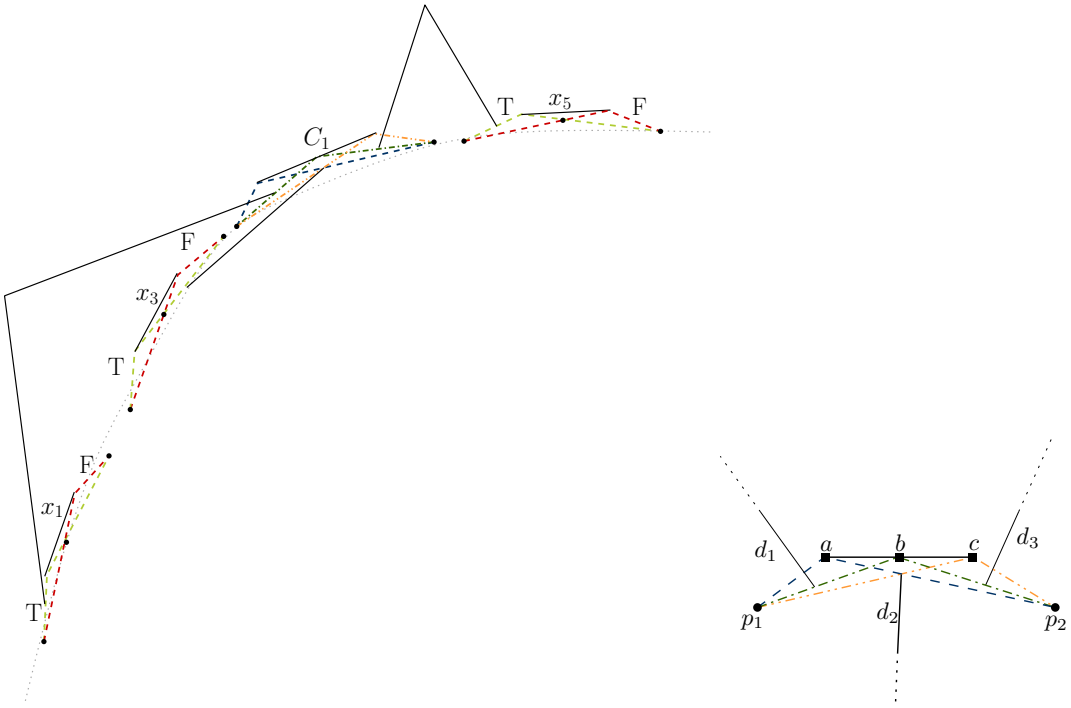


Figure 1.14: From left to right: The placement of the connector bends is shown. (Notice that this is a sketch, actually these gadgets lie on a quarter of a unit circle and, hence, the possible stabbers are convex.) To the right: A clause gadget and the three possible ways to traverse it.

Lemma 1.3.1. *In each clause gadget, a convex (terrain) stabber can stab any two of the three connector bends, but not all three.*

Proof. A convex (terrain) stabber traversing a clause gadget has to stab p_1, p_2 and $\overline{ac}$. Then there are 5 combinatorially different ways to traverse this gadget:

1. The convex (terrain) stabber intersects the segment $\overline{ac}$ in a. Then the stabber can stab d_1 and d_2, but not d_3.

2. The convex (terrain) stabber intersects the segment $\overline{ac}$ in the interval (a, b). The stabber can stab neither d_2 nor d_3, it can only stab d_1.

3. The convex (terrain) stabber intersects the segment $\overline{ac}$ either only in b or in more than one point. Then the stabber can stab d_1 and d_3, but not d_2.

4. The convex (terrain) stabber intersects the segment $\overline{ac}$ in the interval (b, c). The stabber can stab d_3, but neither d_1 nor d_2.

5. The convex (terrain) stabber intersects the segment $\overline{ac}$ in c. The stabber can stab d_2 and d_3, but not d_1.

This proves that a convex (terrain) stabber can stab any two of these connector bends in a clause gadget, but never all three. □

It remains to argue that all bends in the construction are pairwise disjoint. First notice that all bends belonging to a variable or clause gadget can be placed crossing free.

Recall the order in which we placed the gadgets on the positive and the negative arc. This order is crucial for maintaining the disjointness of the construction. The edges in the monotone rectilinear representation do not intersect and the clause gadgets are nested in a specific order. Based on the placement of the gadgets in our construction, we maintain this nesting which is now represented by the outer connectors. Hence, the outer connectors are disjoint and they do not intersect any other bend of the construction. The inner connectors can be placed in such a way that they do not intersect with the variable connectors, see Figure 1.15.

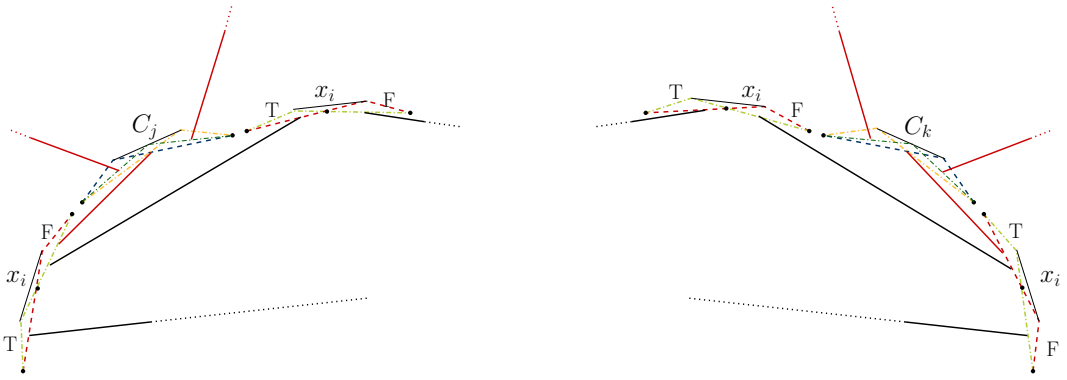


Figure 1.15: The placement of the inner connectors. The left figure shows a positive clause with middle variable x_i, the right figure a negative clause with middle variable x_i. The red segments represent the clause connectors, the black ones the variable connectors.

Also, they do not intersect any other bend of the construction. It follows that our construction can be drawn crossing-free.

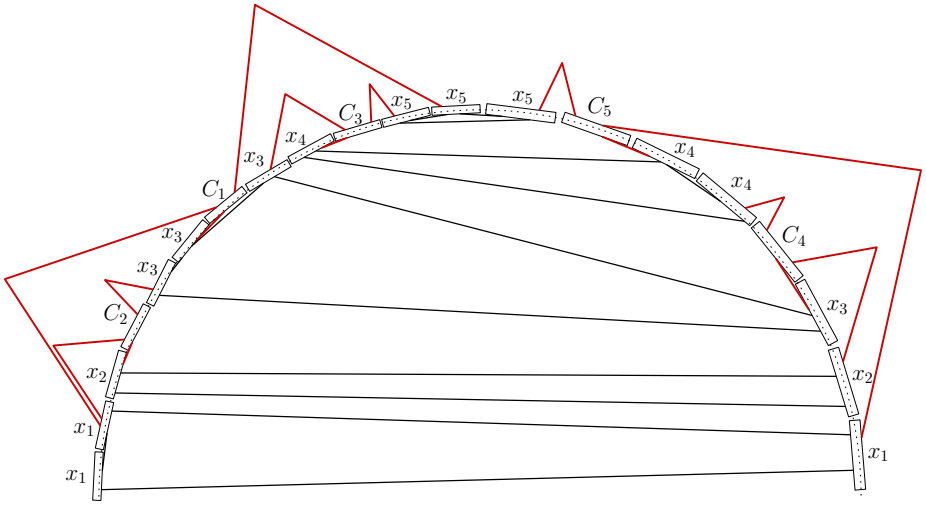


Figure 1.16: A sketch of the construction for the instance in Fig. 1.11 is shown. The clause connectors are marked in fat and red.

Correctness. Assume there exists a satisfying assignment for the 3SAT formula. The convex (terrain) stabber traverses the variable gadgets according to this assignment. In each clause gadget the stabber can stab two connector bends. Since at least one variable is satisfied in each clause, the stabber can omit the connector connecting the satisfied variable to the clause and stab the other two connectors. Hence, the stabber stabs all bends.

On the other hand, assume there is a convex (terrain) stabber that stabs all bends. Set the variables to True or to False depending on how the stabber traverses the variable gadget. This is a satisfying assignment for the 3SAT instance: The setting is consistent since the stabber has to omit at least one connector in each clause gadget. And hence these omitted bends have to be stabbed in the variable gadgets. And there the stabber can either take the True path or the False path, but not both.

The construction can be done in polynomial time. Each variable and each clause gadget consists of a constant number of points and bends and all constructed points are rational and bounded by a polynomial function of the input size. This can be seen in the same way as for the construction for segments. (Notice that the points of the outer bends are not fixed, they can be slightly perturbed in order to become rational.)

Theorem 1.3.2. *Let B be a set of disjoint bends in the plane. It is NP-hard to decide whether there exists a convex (terrain) stabber for B.*

1.4 APX-Hardness

In this section we consider the optimization version of the convex (terrain) stabber problem which we also call the *maximum convex (terrain) stabber problem.*

> Given a set of line segments in the plane, compute a convex (terrain) stabber that stabs the maximum number of segments.

We prove this problem to be APX-hard. APX is the class of all NP optimization problems that have a polynomial-time approximation algorithm with constant approximation factor. Proving a problem to be APX-hard implies that there is no *polynomial-time approximation scheme*, short PTAS, unless P=NP. A PTAS is a family of algorithms, where for each $\epsilon > 0$, there is an algorithm that, given an instance of a problem, computes a solution with relative error ϵ in time polynomial in the size of the input (see also [116]). APX-hardness is defined similar to NP-hardness: A problem is APX-hard if there is a *PTAS-reduction* from every problem in APX to this problem. Thus, a PTAS-reduction is a reduction that preserves the property that a problem has a PTAS, meaning if there exists a PTAS-reduction from problem A to problem B and there exists a PTAS for B then there also exists a PTAS for A (the formal definition of PTAS-reduction can be found in [113, Definition 8.4.1]).

We start with the APX-hardness proof for the problem of finding a convex (terrain) stabber that stabs the maximum number of segments. Later on we show how to generalize this result to the problem where the input is a set of squares. (Notice that it not used in the reduction that we are searching for a convex *terrain* stabber. The reduction works also for a convex stabber.)

We use a reduction from a special version of MAX3-SAT, called MAX-E3SAT(5). MAX-3SAT is defined in the following way: Given a 3SAT formula ϕ (i.e., a boolean expression in conjunctive normal form where each clause contains at most 3 literals), compute an assignment that satisfies the maximum number of clauses in ϕ. MAX-E3SAT(5) is a version of MAX-3SAT where each clause contains exactly 3 literals and every variable occurs in exactly 5 clauses and a variable does not appear in a clause more than once. It is known to be NP-hard to approximate MAX-E3SAT(5) within a factor of $(1-\epsilon)$ for some $\epsilon > 0$ [54]. We first start by reducing MAX-E3SAT(5) to the decision version of the problem (Given a set of line segments and an integer k, does there exists a convex (terrain) stabber that stabs k segments?). The reduction is very similar to the construction of the NP-hardness proof explained before. We use n, m to denote the number of the variables and clauses. Here $5n = 3m$.

We start with a half circle and divide half of it in n arcs of equal size and the other half in m arcs of equal size. Later on, each gadget will be placed along such an arc. In the following, we denote this half circle by K.

Variable gadgets. For each variable we have a gadget that consists of 6 line segments and 18 points (line segments of length 0). The line segments are stacked

on top of each other (by „stacked on top of each other“ we mean identical). The 18 points are partitioned into 3 sets of equal size. The points of each set are stacked on top of each other. The stack of line segments and the 3 stacks of points are arranged in the same way as in the NP-hardness proof for line segments, see Figure 1.2. There are two ways to traverse the gadget (and to stab all segments in the gadget). One way corresponds to setting the variable to True, the other to setting the variable to False. Notice that any other way to traverse the gadget cannot stab all segments/points of this gadget. Thus, a stabber traversing any other way stabs at least 6 segments less than a stabber traversing the True or the False path.

The variable gadgets are placed next to each other on the half circle K. Recall that we divided half of K into n arcs of equal size; in each such arc we place one variable gadget and, hence, each gadget is placed in an arc of length $1/(4n)$ of a unit circle. We call this part of K the *variable arc*.

Clause gadgets. Each clause gadget consists of 4 segments and 8 points. The points are partitioned into two sets of equal size, hence each set consists of 4 points. The segments are stacked on top of each other. The points of a set are also stacked on top of each other. The stack of segments and the stacks of points are arranged in the same way as in the proof for line segments, see Figure 1.3. The clause gadgets are placed on the half of K which was divided into m arcs. Thus, each clause gadget is fit into an arc of $1/(4m)$ of a unit circle. We call this part of K the *clause arc*. The clause arc lies next to the variable arc on K, see Figure 1.4.

Connector segments. We now place $3m$ connector segments, connecting a variable gadget to a clause gadget whenever the variable appears in the clause. The placement of these segments is exactly the same as in the NP-hardness proof for line segments in Section 1.1. (See also Figure 1.5.)

Observe that there exists a convex (terrain) stabber that stabs all segments of the variable and clause gadgets and at least 2 out of the 3 connector segments of each clause gadget. Thus, there always exists a stabber that stabs at least $24n + 14m$ segments.

Lemma 1.4.1. *There is a convex (terrain) stabber stabbing $24n+14m+k$ segments if and only if there is an assignment satisfying k clauses.*

Proof. If there is an assignment that satisfies k clauses, the stabber traverses the variable gadgets according to the assignment. In each satisfied clause at least one of the connectors is already stabbed, hence the stabber stabs the other two connector segments. Thus, it stabs 3 connector segments for each of the k satisfied clauses and 2 connector segments for each of the $m-k$ not satisfied clauses. In total, the stabber stabs $24n + 12m + 3k + 2(m - k) = 24n + 14m + k$ segments.
If there is a stabber stabbing $24n+14m+k$ segments, then there also exists a stabber

that stabs the gadgets in *the right order* and stabs at least $24n + 14m + k$ segments. Stabbing in the right order means that the stabber either takes the True or False path at each variable gadget and stabs at least two connector segments at each clause gadget. A variable gadget consists of 24 segments (18 of them are segments of length 0). Moreover, 5 connecting segments have their endpoints inside any variable gadget because each variable appears in at most 5 clauses. Without traversing the gadget in the right order, a stabber can stab at most 23 of these segments. A stabber traversing the gadget in the right order stabs at least 24 segments. A clause gadget consists of 12 segments, additionally 3 connecting segments have their endpoint inside the variable gadget. Any stabber stabbing the gadget in the right order stabs exactly 14 segments: 2 connector segments and the 12 segments belonging to the clause gadget. Any other stabber stabs at most 11 segments. Hence, if a convex stabber stabs $24n + 14m + k$ segments and it does *not* traverse the gadgets in the right order, than there always exists another convex stabber traversing the gadgets in the right order and stabbing more than $24n + 14m + k$ segments. So in the following we assume that the stabber stabs the gadgets in the right order. This stabber stabs already $24n+14m$ segments by traversing the gadgets in the right order. The k additional segments are stabbed in the variable gadgets and represent connector segments. Recall that this stabber stabs already two connector segments in each clause gadget. Hence, each of these k additional segment is a connector segment that belongs to a different clause gadget. We set the variables according to the stabber traversing these gadgets and so the formula has k satisfied clauses. □

Theorem 1.4.2. *Let L be a set of line segments in the plane. It is APX-hard to compute a convex (terrain) stabber that stabs the maximum number of segments of L.*

Proof. We use a PTAS-reduction from MAX-E3SAT(5). Let n be the number of variables and m be the number of clauses. We reduce the problem to the convex (terrain) stabber problem as explained before. Let k be the maximum number of satisfied clauses. Since there always exists an assignment satisfying at least 7/8 of the clauses we conclude that $k \geqslant 7/8m$. Assume there exists a polynomial-time algorithm for maximum convex (terrain) stabber problem that returns a solution that is at least $(1 - \epsilon)$ times the value of the optimal solution. Then we can approximate MAX-E3SAT(5) by subtracting $24n + 14m$ (note that $3m = 5n$):

$$\begin{aligned}(1-\epsilon)(24n+14m+k) - 24n - 14m &= k - \epsilon k - 142/5m\epsilon \leqslant k - \epsilon k - 1136/35\epsilon k \\ &\leqslant (1-\epsilon')k\end{aligned}$$ □

This construction can be generalized to a set of squares and even to set of simple regular k-gons in the same way as for the NP-hardness proof explained in the section before (assuming that a regular k-gon is represented by two points and a direction). So we get the following results.

Theorem 1.4.3. *Let S be a set of squares in the plane. It is APX-hard to compute a convex (terrain) stabber that stabs the maximum number of segments of S.*

Theorem 1.4.4. *Let P be a set of simple regular k-gons in the plane. It is APX-hard to compute a convex (terrain) stabber that stabs the maximum number of segments of P.*
Remark*: A regular k-gon is represented by two points and an order.*

Since there exists a PTAS for *planar* MAX SAT [79], we cannot use these ideas to show APX-hardness for a set of disjoint bends.

1.5 Conclusions and Open Problems

In this chapter we proved that Tamir's problem is NP-hard if the input set is a set of line segments. We showed that the problem remains NP-hard for a input set of squares or a set of regular polygons. These results answer a long-standing open question.

We also considered the optimization case and proved it to be APX-hard for most of our studied variants.

However, there are still a lot of interesting open problems in the area of convex stabbing. We could not show the problem of stabbing a set of disjoint line segments to be NP-hard. Our reduction for showing that stabbing disjoint bends is NP-hard cannot be extended to line segments. A polynomial-time algorithm for this problem would show that there is a clear difference between the complexity of stabbing convex and non convex polygons regarding disjointness. It is also not known whether finding a convex stabber for a set of disks is NP-hard. Even the simple case of disjoint unit disks is still open.

Another interesting variant of the convex stabbing problem is the following: We are given a set of geometric objects and additionally an order on these objects. We want to find a convex stabber that stabs these objects in the given order. To the best of our knowledge this problem has not been studied so far. However, Guibas et al. [64] studied the problem of computing a minimum-link path that stabs a sequence of convex objects, more precisely they want to compute a path with the minimum number of edges that stabs a sequence of convex objects. They presented a linear time algorithm for a set of convex objects where consecutive objects are disjoint and an $\mathcal{O}(n^2 \log^2 n)$ time algorithm for a set of disks. It is not difficult to see that if the minimum-link path is not convex, then there is no convex path stabbing this set of objects. Thus, the results of Guibas et al. help to decide whether there exists a convex terrain stabber, but it is not obvious whether they can be used in order to find a convex stabber.

In summary, it seems that convex stabbing opens a whole new research area.

Chapter 2

Stabbing with the Vertices of a Polygon

In this chapter we study the problem of stabbing a set of geometric objects with the *vertices of a polygon*. We say the vertices of a polygon stab a set of objects if there is a bijective function assigning each object to a vertex of the polygon contained in it. (Thus, the number of points and the number of vertices has to be the same.) The polygon whose vertices are supposed to stab the set of objects is then called the *stabbing polygon*. In Section 2.1 we assume the stabbing polygon to be regular and we present the first polynomial-time algorithm for this problem. (To the best of our knowledge, until now, it was not known whether the existence of a regular polygon whose vertices stab a set of given objects can be decided in polynomial time.) In Section 2.2 the stabbing polygon is a scaled, translated, rotated copy of a given polygon. We show that the results of Section 2.1 can be generalized to this case.

Although it is not difficult to generalize the results in Section 2.1, we still state the results in two different sections as we believe it is much easier to understand the algorithm for the case of regular polygons.

Partial results of Section 2.1 are joint work with Claudia Dieckmann and are part of research published in [22, 23] (together with additional results by Arkin, Knauer, Mitchell, Polishchuk, and Yang).

Notation. The Euclidean distance between two points p, q in the plane is denoted by $d(p, q)$. The radius of a disk D is denoted by $r(D)$. The segment between two points p, q in the plane is denoted by $\overline{pq}$ and its length by $|\overline{pq}|$.

In the following, a regular polygon will be always simple.

2.1 Stabbing with the Vertices of a Regular Polygon

We consider the following problem:

> Given a set of n geometric objects in the plane, decide whether there exist a regular n-gon whose vertices stab the objects.

We first give an algorithm based on computational algebraic geometry. This algorithm can be used for different families of objects but the running time of the algorithm differs, depending on the complexity of the objects.

Later on, we consider in detail the cases where the objects are (i) line segments or (ii) disks. For the case of line segments we give a faster algorithm.

Before we state our algorithm, we formulate the problem in a different way:

> Given a set of geometric objects $B = \{b_1, b_2, \ldots, b_n\}$ in the plane, can we pick a point in each object such that these points are the vertices of a regular n-gon?

A regular n-gon has rotational symmetry of order n with respect to its center. This means that the polygon is invariant under rotation of an angle of $2\pi/n$ around its center. If the center and a vertex are given, the regular n-gon is determined. In order to compute the remaining vertices of the n-gon, the given vertex is rotated around the center by an angle of $2k\pi/n$ for $k = 1, 2, ..., n-1$. The rotational image points of this given vertex are the remaining vertices of the regular n-gon.

We present now an algorithm that solves the problem above. The basic idea of our algorithm is to compute the rotation center and a vertex of the regular n-gon, if it exists.

We start with introducing some new notation and definitions: Let p, c be points in the plane and k an integer. The counterclockwise rotational image of p around c by the angle $2k\pi/n$ is denoted by $\rho_c^k(p)$. Consider two points p, q in the plane and an integer k. We call a point c the *apex point* of p, q and k if $\rho_c^k(p) = q$. For two geometric objects b_i, b_j and an integer k we compute for every $p \in b_i$ all possible apex points c such that $\rho_c^k(p) \in b_j$. We denote the apex region of b_i, b_j and the angle $2k\pi/n$ by A_{ij}^k. Thus,

$$A_{ij}^k = \{(p, c) \mid c \in \mathbb{R}^2, p \in b_i, \rho_c^k(p) \in b_j\}$$

In the following we assume that each geometric object $b \in B$ has constant description complexity, i.e., it can be described with a constant number of polynomial (in)equalities of constant degree. Because of this, the apex regions can also be described with a constant number of polynomial (in)equalities of constant degree. In the remaining, we assume that each apex region can be described by a constant number of polynomial (in)equalities of degree a with d variables (note that a is constant and $d \leqslant 4$).

Algorithm. Our algorithm proceeds in the following way (the algorithm is also summarized in Algorithm 1): We start by fixing an object, say b_1. There exists a

regular n-gon whose vertices stab B if and only if there exists a point $p \in b_1$ and a center c such that $\rho_c^k(p) \in b'_k$ for $1 \leqslant k < n$ and $\{b'_1, b'_2, ..., b'_{n-1}\} = B \setminus \{b_1\}$. Such a point $p \in b_1$ and a center c exist when the intersection of the $n-1$ apex regions of b_1, b'_k and k is non-empty for $1 \leqslant k < n$. We consider all possible apex regions $A^k_{1,j}$ for $2 \leqslant j \leqslant n$ and $1 \leqslant k \leqslant n-1$. Since there are $n-1$ possible values for j and k, in total there are $\mathcal{O}(n^2)$ such apex regions. Each of these regions can be described by a constant number of polynomial (in)equalities of degree a with d variables (as mentioned before); thus, each region can be described by a constant number of hypersurfaces of degree a in $\mathbb{R}^d$. So the arrangement of these apex regions has polynomial size. We call a point $r = (p, c) \in \mathbb{R}^d$ feasible if it belongs to $n-1$ apex regions, with each region being from a different object with a different angle. Our problem has a feasible solution if and only if there exists a feasible point in $\mathbb{R}^d$. The feasibility of a point does not change when the point moves inside a cell of the arrangement of the apex regions, thus, in order to determine existence of a feasible point it is enough to check the feasibility of an arbitrary representative r inside every cell of the arrangement (notice that we consider all cells of all dimensions).

In order to check the feasibility of a representative r we do the following: For each representative $r = (p, c)$ we compute a corresponding graph G_r. The set of nodes $V = V_1 \dot{\cup} V_2$ consists of two disjoint sets. The set V_1 contains the nodes $\{v_2, v_3, \ldots, v_n\}$, where v_i represents b_i. The set V_2 contains the nodes $\{u_1, u_2, \ldots, u_{n-1}\}$, where u_i represents the angle $2i\pi/n$. There is an edge between a node $v_i \in V_1$ and a node $u_j \in V_2$ if the point p rotated around c by the angle $2j\pi/n$ lies on b_i. The graph G_r is bipartite since each edge connects a vertex of V_1 with a vertex of V_2 and there are no edges between two vertices of the same set. The following lemma shows that if G_r contains a perfect matching, then c is the rotation center and p a vertex of a regular n-gon whose vertices stab B.

Lemma 2.1.1. *G_r contains a perfect matching if and only if c is the rotation center and p a vertex of a regular n-gon whose vertices stab B.*

Proof. Assume G_r contains a perfect matching. Then each vertex in V_1 matches a vertex in V_2. This implies that there is a bijective function between the objects of $B \setminus \{b_1\}$ and the angles $2i\pi/n$, $i = 1, \ldots, n-1$. Let b'_i be mapped to $2i\pi/n$ for $1 \leqslant i \leqslant n-1$ and $\{b'_1, b'_2, \ldots, b'_{n-1}\} = B \setminus \{b_1\}$. Then $\rho_c^i(p) \in b'_i$ for $1 \leqslant i < n$ and so c is the rotation center and p a vertex of a regular n-gon whose vertices stab B.

Assume c is the rotation center and p a vertex of a regular n-gon whose vertices stab B. Then $\rho_c^k(p) \in b'_k$ for $1 \leqslant k < n$ and $\{b'_1, b'_2, ..., b'_{n-1}\} = B \setminus \{b_1\}$. There exists a bijective function between the objects in $B \setminus \{b_1\}$ and the angles $2k\pi/n$, $k = 1, \ldots, n-1$; so there also exists a perfect matching in G_r. □

We decide for every representative of a cell in the arrangement whether the corresponding graph has a perfect matching. If one of the graphs contains a perfect matching, there exists a regular n-gon whose vertices stab B.

Algorithm 1

Input: Set of geometric objects $B = \{b_1, \ldots, b_n\}$ in the plane.

Output: TRUE if there exists a regular n-gon stabbing B, and FALSE otherwise.

$\mathcal{A} = \emptyset$
for $j \leftarrow 2$ **to** n **do**
  for $k \leftarrow 1$ **to** $n-1$ **do**
    $\mathcal{A} = \mathcal{A} \cup \{A^k_{1,j}\}$
Compute a representative $r = (p, c)$ for each cell of the arrangement $\mathcal{A}$.
for all r **do**
  //Build corresponding graph $G_r = (V_1 \dot\cup V_2, E)$*:*
  $V_1 = \{v_2, v_3, \ldots, v_n\}$, $V_2 = \{u_1, u_2, \ldots, u_{n-1}\}$, $E = \emptyset$
  for $i \leftarrow 2$ **to** n **do**
    for $j \leftarrow 1$ **to** $n-1$ **do**
      if $\rho_c^j(p) \in b_i$ **then**
        $E = E \cup \{(v_i, u_j)\}$
  if $G_r = (V_1 \dot\cup V_2, E)$ has a perfect matching **then**
    return TRUE
return FALSE

We still have to analyze the running time. For this, we need the following theorems:

Theorem 2.1.2 ([98])**.** *The number of cells of all dimensions in an arrangement of n algebraic hypersurfaces, each of degree $\leqslant a$, in $\mathbb{R}^d$ is $(\mathcal{O}(na)/d)^d$.*

Theorem 2.1.3 ([27, Theorem 2])**.** *A sample point of each cell (of all dimensions) of the arrangement of n algebraic hypersurfaces, each of degree $\leqslant a$, in $\mathbb{R}^d$ can be computed in $n^{d+1}a^{\mathcal{O}(d)}$ time.*

(An overview on arrangements of algebraic surfaces, i.e, their complexity and how to compute representatives for each cell, can be found in [6].)

In total there are $\mathcal{O}(n^2)$ apex regions; each of these regions can be described by a constant number of hypersurfaces of degree a in $\mathbb{R}^d$ (recall that a is constant and $d \leqslant 4$). From Thereom 2.1.2 follows that the arrangement of all apex regions consists of $\mathcal{O}(n^{2d})$ cells. We have to compute a representative, i.e, a sample point, for each cell, this takes $\mathcal{O}(n^{2(d+1)})$ time (Theorem 2.1.3). For each representative r we compute the corresponding graph G_r which consists of $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^2)$ edges. Then we decide whether this graph has a perfect matching in $\mathcal{O}(n^{2.5}/\sqrt{\log n})$ time [14]. If one of the graphs contains a perfect matching, there exists a regular n-gon whose vertices stab B. This n-gon can be computed in linear time, since we know its center and a vertex of it. Thus, we achieve the following result

Theorem 2.1.4. *Let B be a set of n geometric objects in the plane. Assuming the apex region of two objects in B and an angle can be described by a constant number of polynomial (in)equalities of constant degree with d variables, we can decide in $\mathcal{O}(n^{2d+2.5}/\sqrt{\log n})$ time whether there exists a regular n-gon whose vertices stab B. The n-gon can be computed in the same time, if it exists.*

2.1.1 Stabbing Line Segments

We consider the following problem:

> Given a set of line segments $L = \{l_1, l_2, .., l_n\}$ in the plane, decide whether there exists a regular n-gon whose vertices stab L and compute the n-gon if it exists.

Algorithm 1 can be used to solve this problem. In order to analyze the running time, we have to consider the apex region of l_i, l_j and an angle $2k\pi/n$. This is,

$$A_{ij}^k = \{(p, c) \mid c \in \mathbb{R}^2, p \in l_i, \rho_c^k(p) \in l_j\}$$

This set is characterized by polynomial (in)equalities for $p = (p_1, p_2)$ and $c = (c_1, c_2)$: Without loss of generality, we assume that all segments in L are non-vertical (if a segment is vertical we slightly perturb the coordinates [49]). Let $l_i = \{ax + b \mid x \in [x_1, x_2]\}$ and $l_j = \{sx + t \mid x \in [x_3, x_4]\}$. The image point (x', y') of a point (x, y) rotated around c by an angle α is given by $x' = c_1 + \cos\alpha(x - c_1) - \sin\alpha(y - c_2)$ and $y' = c_2 + \sin\alpha(x - c_1) + \cos\alpha(y - c_2)$ (see, e.g., [57]). Then,

$$p_2 = ap_1 + b \tag{2.1}$$
$$x_1 \leqslant p_1 \leqslant x_2 \tag{2.2}$$
$$c_2 + \sin(2\pi k/n)(p_1 - c_1) + \cos(2\pi k/n)(p_2 - c_2) = s(c_1 + \cos(2\pi k/n)(p_1 - c_1) - \sin(2\pi k/n)(p_2 - c_2)) + t \tag{2.3}$$
$$x_3 \leqslant c_1 + \cos(2\pi k/n)(p_1 - c_1) - \sin(2\pi k/n)(p_2 - c_2) \leqslant x_4 \tag{2.4}$$

We can eliminate p_2 by combining (2.1) and (2.3) and combining (2.1) and (2.4). Thus, A_{ij}^k can be considered as subset of $\mathbb{R}^3$; the variables are p_1, c_1, and c_2. Applying Theorem 2.1.4, we get the following result

Corollary 2.1.5. Let L be a set of n line segments in the plane. We can decide in $\mathcal{O}(n^{8.5}/\sqrt{\log n})$ time whether there exists a regular n-gon whose vertices stab L. The n-gon can be computed in the same time, if it exists.

By studying the geometric properties of these apex regions, we can improve the running time.

Geometric properties of the apex regions. The apex regions are parallelograms in $\mathbb{R}^3$. This can be concluded from the (in)equalities (2.1)-(2.4). But there is also a nice geometric way to prove this property, which we will state in the following.

Lemma 2.1.6 ([74, Lemma 2.4.6])**.** *Let p, q, r be three points in the plane and let $k > 0$ be an integer. Let c_1 be the apex point of p, q and k and let c_2 be the apex point of p, r and k. Then $d(q, r) = 2\sin(k\pi/n)d(c_1, c_2)$.*

Lemma 2.1.7. *Let l be a line segment and p a point in the plane and let $k > 0$ be an integer. The apex region of p, l and k is a line segment with length $|l|/2\sin(k\pi/n)$.*

Proof. Let q be a point on l. We consider the isosceles triangle $\triangle pcq$ where c is the apex point of p, q and k and so $\angle qcp = 2\pi k/n$ (Figure 2.1). Let $q' \neq q$ a point on l and let c' be the apex point of p, q' and k. Clearly $\triangle pcq$ and $\triangle pc'q'$ are similar ($\angle qcp = \angle q'c'p$ and $|\frac{\overline{q'c'}}{\overline{qc}}| = |\frac{\overline{pc'}}{\overline{pc}}|$). This implies that $\triangle pqq'$ and $\triangle pcc'$ are similar since $\angle qpq' = \angle cpc'$. It follows that c' lies on a segment containing c that builds an angle of $\angle pqq'$ with the segment $\overline{pc}$. Since $\angle pqq'$ has the same value for all $q' \in l$, namely the value of the angle between $\overline{pq}$ and the line l, we can conclude that for any point $q'' \in l$ the apex point of p, q'' and k lies on the same segment. Thus, we know that that the apex region is a line segment. The length of this segment

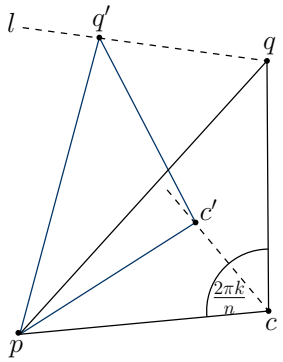


Figure 2.1: The points p and q and their apex point build an isosceles triangle.

follows immediately from Lemma 2.1.6. Hence, we conclude that the apex region is a segment with length $|l|/2\sin(k\pi/n)$. □

We are interested in the apex regions $A^k_{1,j}$ for $2 \leqslant j \leqslant n$ and $1 \leqslant k \leqslant n-1$. We will prove that these apex regions are parallelograms in $\mathbb{R}^3$.

Lemma 2.1.8. *Let l_1, l_2 be two line segments in the plane and $k > 0$ an integer. The apex region A^k_{12} forms a parallelogram in $\mathbb{R}^3$. The sides of the parallelogram have length $|l_1|/2\sin(k\pi/n)$ and $|l_2|/2\sin(k\pi/n)$.*

Proof. Let s and t be the two endpoints of l_1, hence $l_1 = \lambda\vec{s} + (1-\lambda)\vec{t}$, $\lambda \in [0, 1]$. We consider a coordinate system with axes x, y, λ, see Figure 2.2. If we consider a point

$p = \lambda' \vec{s} + (1 - \lambda')\vec{t} \in l_1$, the apex region of p, l_2 and k is a line segment of length $|l_2|/2\sin(k\pi/n)$ in the plane $\lambda = \lambda'$ (Lemma 2.1.7). Also, if we consider any point $q \in l_2$, the apex region of q, l_1 and k is a line segment of length $|l_1|/2\sin(k\pi/n)$ with one endpoint in the plane $\lambda = 0$ and the other one in the plane $\lambda = 1$. It follows that A_{12}^k is a parallelogram with height 1, meaning the lower side of the parallelograms lies in the plane $\lambda = 0$ and the upper side lies in the plane $\lambda = 1$. One side length is $|l_2|/2\sin(k\pi/n)$ and the other $|l_1|/2\sin(k\pi/n)$. □

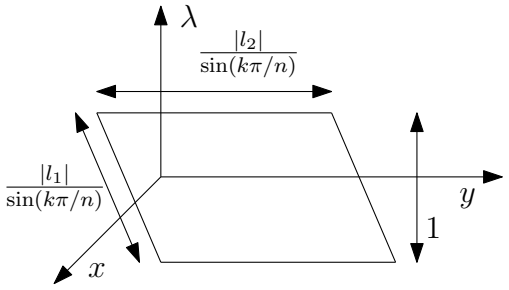


Figure 2.2: The apex region A_{12}^k is shown.

Since we know that the apex regions are parallelograms in $\mathbb{R}^3$, we can use a simple idea in order to improve the running time of the algorithm given before.

Faster Algorithm. Again, we start by fixing a segment, say l_1. The idea now is that we also fix a second segment, say l_2, and an integer $\tilde{k} \in \{1, 2, ..., n-1\}$. We will repeat the following procedure for all possible values for $\tilde{k}$: We want to decide whether there exist a point $p \in l_1$ and a center c such that $\rho_c^{\tilde{k}}(p) \in l_2$ and $\rho_c^k(p) \in l'_k$ for $1 \leqslant k \neq \tilde{k} \leqslant n-1$ and $\{l'_1, l'_2, \ldots, l'_{n-2}\} = L \setminus \{l_1, l_2\}$. Such a point $p \in l_1$ and a center c exist if the intersection of $A_{12}^{\tilde{k}}$ and the $n-2$ apex regions of l_1, l'_k and k is non-empty, for $1 \leqslant k \neq \tilde{k} \leqslant n-1$. (This means that there exists a regular n-gon whose center is c, one vertex is $p \in l_1$ and the vertex on l_2 is $\rho_c^{\tilde{k}}(p)$ and this n-gon stabs L.) To this end, we compute $A_{12}^{\tilde{k}}$ and we compute all apex regions A_{1j}^k for $1 \leqslant k \neq \tilde{k} \leqslant n-1$ and $3 \leqslant j \leqslant n$. We compute the intersections $A_{12}^{\tilde{k}} \cap A_{1j}^k$ (all these intersections lie on one plane, namely the plane defined by $A_{12}^{\tilde{k}}$). We know that $A_{12}^{\tilde{k}} \cap A_{1j}^k$ is either empty, a line segment, or a quadrangle $P \subset A_{12}^{\tilde{k}}$. Thus, the complexity of the resulting arrangement is $\mathcal{O}(n^4)$. Again, we have to decide whether there exists a feasible point $r = (p, c) \in A_{12}^{\tilde{k}}$ that means it belongs to $n-1$ apex regions, with each region being from a different segment with a different angle. We can consider the intersection points of the arrangement as possible candidates. For each point we construct the bipartite graph as before and decide whether it has a perfect matching. Since graphs of neighboring cells differ in only one edge and we know the structure of the arrangement, we can further improve the running time by using a dynamic matching algorithm. We traverse the cells in depth first search

order and build the bipartite graph for the representative of each cell. In each step we have to update the matching. An update in the dynamic matching algorithm can be done in $\mathcal{O}(n^2)$ time [13]. We have to check $\mathcal{O}(n^4)$ representatives. For each we build a bipartite graph and decide whether the graph has a perfect matching. For the first representative, this takes $\mathcal{O}(n^{2.5})$ time [70], for all other representative this needs only $\mathcal{O}(n^2)$ time.
Hence the running time is $\mathcal{O}(n^6)$. We have to repeat this procedure for every $A_{12}^{\tilde{k}}$, $\tilde{k} = \{1, ...n-1\}$, and thus the total running time is $\mathcal{O}(n^7)$.

Theorem 2.1.9. *Let L be a set of n line segments in the plane. We can decide in $\mathcal{O}(n^7)$ time whether there exists a regular n-gon whose vertices stab L. The n-gon can be computed in the same time, if it exists.*

2.1.2 Stabbing Disks

We want to analyze the running time of Algorithm 1 for the case where the objects are disks. Thus, we consider the following problem:

> Given a set of disks $\mathcal{D} = \{D_1, D_2, .., D_n\}$ in the plane, decide whether there exists a regular n-gon whose vertices stab $\mathcal{D}$ and compute the n-gon, if it exists.

We consider the apex region of two disks D_i, D_j and an angle $2k\pi/n$:

$$A_{ij}^k = \{(p, c) \mid c \in \mathbb{R}^2, p \in D_i, \rho_c^k(p) \in D_j\}$$

This is a set of two polynomial inequalities, one for $p = (p_1, p_2)$ and one for $c = (c_1, c_2)$: Let the center of D_i be $a = (a_1, a_2)$ and let the center of D_j be $b = (b_1, b_2)$. Thus, $D_i = \{(x, y) \mid (x - a_1)^2 + (y - a_2)^2 \leqslant r(D_i)^2\}$ and $D_j = \{(x, y) \mid (x - b_1)^2 + (y - b_2)^2 \leqslant r(D_j)^2\}$.

Then,

$$(p_1 - a_1)^2 + (p_2 - a_2)^2 \leqslant r(D_i)^2 \tag{2.5}$$

$$(c_1 + \cos(2\pi k/n)(p_1 - c_1) - \sin(2\pi k/n)(p_2 - c_2) - b_1)^2 + (c_2 + \sin(2\pi k/n)(p_1 - c_1) + \cos(2\pi k/n)(p_2 - c_2) - b_2)^2 \leqslant r(D_j)^2 \tag{2.6}$$

There are four variables, namely p_1, p_2, c_1, c_2, and, thus, $A_{ij}^k \subset \mathbb{R}^4$. Applying Theorem 2.1.4, we get the following result

Theorem 2.1.10. *Let $\mathcal{D}$ be a set of n disks in the plane. We can decide in $\mathcal{O}(n^{10.5}/\sqrt{\log n})$ time whether there exists a regular n-gon whose vertices stab $\mathcal{D}$. The n-gon can be computed in the same time, if it exists.*

2.1.3 Optimization Case for Disks

In this section we consider the following problem:

> Given a set P of n points in the plane, compute the minimum value δ such that for each point exists a translation by a vector of length $\leqslant \delta$ and the translated points lie in symmetric position.

Symmetric position means that the points are vertices of a regular n-gon. This problem is a simplified version of the so-called *approximate symmetry detection problem*: Given a set of n disks and an integer k, is it possible to place a point in each disk such that this set of points is invariant under rotations by $2\pi/k$. For any fixed $k \geqslant 3$ the problem is NP-hard [73]. For more details of the approximate symmetry detection problem see [43, 73, 74].

Our problem is equivalent to the approximate symmetry detection problem where $k = n$. This special case seems to be easier than the problem for general k, as an exact solution can be computed in polynomial time ($\mathcal{O}(n^{20})$) [22, 23]. We will present a fast and easy constant approximation algorithm and a $(1+\epsilon)$-approximation algorithm.

In the following, let δ^* be the minimum value such that for each point of P exists a translation by a vector of length $\leqslant \delta^*$ and the translated points are vertices of a regular n-gon. This n-gon will be denoted by Q^* and its center by c^*.

Constant factor approximation. The idea of this algorithm is quite simple. Take any point $p \in P$ and compute a regular n-gon that has p as a vertex and the centroid of P as center. This can be done in linear time. We denote the resulting regular n-gon by Q. Constructing Q does not yield an approximation of δ^*, since we do not know which point of P is moved to which vertex of Q and by what value. In order to get this value we compute the bottleneck distance between P and the vertices of Q. (The bottleneck distance between two point sets P and Q is defined as the minimum over all one-to-one correspondences $f : P \to Q$ of $\max_{p \in P} d(p, f(p))$.) The bottleneck distance can be computed in $\mathcal{O}(n^{1.5} \log n)$ time [51]. Hence, the total running time is $\mathcal{O}(n^{1.5} \log n)$. We still have to analyze the approximation factor. We will show that Q^* can be transformed to Q by translating each vertex by at most $3\delta^*$, hence each point of P has to be translated by at most $4\delta^*$ in order to be a vertex of Q. Let λ be be the bottleneck distance between P and the vertices of Q. We want to prove the following.

Lemma 2.1.11. $\lambda \leqslant 4\delta^*$

In order to prove Lemma 2.1.11 we need the following two lemmas.

Lemma 2.1.12. *Let Q be a regular n-gon with center c and let p be a point in the plane. Let q be any vertex of Q. There exists a regular n-gon Q' with center c and*

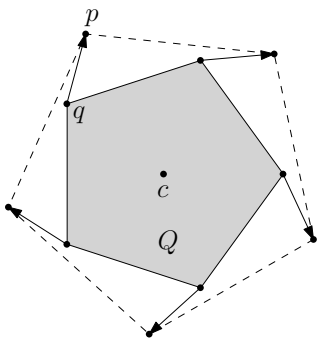


Figure 2.3: Q can be transformed into the dashed polygon which is still a regular n-gon with center c.

p as a vertex such that the corresponding vertices of Q and Q' have distance exactly $d(p,q)$.

Proof. Q is a regular n-gon and so it has rotational symmetry n. Let the vertices of Q be $q_1, q_2, \ldots, q_n$ in counterclockwise order and let $q = q_n$. We move q on p, this can be done by the translation vector $\vec{qp}$. Now we rotate this vector around c by an angle of $2\pi/n$ and translate q_2 by this new vector and so on. (More precisely, each vertex q_i of Q is translated by a vector that is defined by rotating $\vec{qp}$ around c by an angle of $2i\pi/n$.) Each vertex is translated by a distance of $d(p,q)$ and the resulting polygon is again a regular n-gon with center c and it has p as a vertex, see Figure 2.3. □

The following lemma was proven by Iwanowski [74].

Lemma 2.1.13 ([74])**.** *Let $P = \{p_1, p_2, .., p_n\}$ be a set of points in the plane and let g be the centroid of P. Let Q be a regular n-gon with center c such that the corresponding vertices of Q and points of P have distance at most δ^*. Then, $d(c,g) \leqslant \delta^*$.*

Now we can state the proof of Lemma 2.1.11.

Proof of Lemma 2.1.11. Let the centroid of P be denoted by g. We start by translating Q^* in such a way that the center of Q^* coincides with g. Hence Q^* is translated by $\vec{c^*g}$. By Lemma 2.1.13 we know that $|d(c^*g)| \leqslant \delta^*$ and so there is a vertex of the translated polygon Q^* that has a distance smaller than $2\delta^*$ to p (recall that p is a point of P). By Lemma 2.1.12, Q can be obtained from the translated version of Q^* by translating each vertex by at most $2\delta^*$. It follows that each point of P has to be translated by at most $4\delta^*$ to coincide with its corresponding vertex of Q. □

Lemma 2.1.11 proved the approximation factor of our algorithm and, hence, we have the following result.

Theorem 2.1.14. *Let P be a set of n points in the plane. A value δ such that for each point $p \in P$ exists a translation by a vector of length $\leqslant \delta$, the translated points lie in symmetric position, and $\delta \leqslant 4\delta^*$ (where δ^* is the optimal value) can be computed in $\mathcal{O}(n^{1.5} \log n)$ time.*

$(1+\epsilon)$-Approximation. We first compute a 4-approximation for the problem in $\mathcal{O}(n^{1.5} \log n)$ time using our algorithm from Theorem 2.1.14. Let δ^{apx} be the computed value. We lay out a $\frac{1}{\epsilon} \times \frac{1}{\epsilon}$-grid in the δ^{apx}-neighborhood of the centroid of P and denote this grid by G_1. We also lay out a $\frac{1}{\epsilon} \times \frac{1}{\epsilon}$-grid in the δ^{apx}-neighborhood of any point $p \in P$ and denote it by G_2. The grid size of these grids is $\delta^{\text{apx}}\epsilon$. Now we take all possible pairs (c', p') of points where c' is a grid point of G_1 and p' is a grid point of G_2. We construct the regular n-gon that has center c' and p' as a vertex, this takes linear time. Then we compute the bottleneck distance between the vertices of this polygon and P in $\mathcal{O}(n^{1.5} \log n)$ time. Hence, the total running time is $\mathcal{O}(\frac{1}{\epsilon^4} n^{1.5} \log n)$. We claim that the regular n-gon whose vertices have the smallest bottleneck distance to the points in P gives a $(1+\epsilon)$ approximation for the problem. Let λ be the value of this bottleneck distance. Recall that δ^* is the minimum value such that for each point of P exists a translation by a vector of length $\leqslant \delta^*$ and the translated points are vertices of a regular n-gon. This n-gon will be denoted by Q^* and has center c^*. We have the following:

Lemma 2.1.15. *$\lambda \leqslant (1+\epsilon')\delta^*$ for any $\epsilon' \geqslant 12\sqrt{2}\epsilon$.*

Proof. There is a grid point c in G_c that has distance at most $\sqrt{2}\delta^{\text{apx}}\epsilon \leqslant 4\sqrt{2}\epsilon\delta^*$ to c^* and there is a grid point p in G_p that has distance at most $4\sqrt{2}\epsilon\delta^*$ to p^* where p^* is a vertex of Q^*. Compute the regular n-gon Q that has c as center and p as a vertex. We can translate Q^* to Q by translating each vertex by at most $12\sqrt{2}\epsilon\delta^*$ in the following way. First we translate Q^* in such a way that the center of Q^* coincides c; Q^* is translated by at most $4\sqrt{2}\epsilon\delta^*$. Then there is a vertex of the translated polygon Q^* that has distance smaller than $8\sqrt{2}\epsilon\delta^*$ to p. By Lemma 2.1.12, Q can be obtained from the translated version of Q^* by translating each vertex by at most $8\sqrt{2}\epsilon\delta^*$. It follows that each point of P has to be translated by at most $\delta^* + 12\sqrt{2}\epsilon\delta^* = \delta^*(1 + 12\sqrt{2}\epsilon) \leqslant \delta^*(1+\epsilon')$ to coincide with its corresponding vertex of Q. □

We conclude with:

Theorem 2.1.16. *Let P be a set of n points in the plane. A value δ such that for each point $p \in P$ exists a translation by a vector of length $\leqslant \delta$, the translated points lie in symmetric position, and $\delta \leqslant (1+\epsilon)\delta^*$ (where δ^* is the optimal value) can be computed in $\mathcal{O}(\frac{1}{\epsilon^4} n^{1.5} \log n)$ time.*

2.2 Stabbing with the Vertices of a General Polygon

Last, we consider a generalized version of the problem studied in Section 2.1 where the restriction to regular polygons is removed. We first state the problem.

> Given a set of geometric objects $B = \{b_1, b_2, \ldots, b_n\}$ and a polygon P with n vertices in the plane, decide whether there exists a scaled, translated, and rotated copy of P whose vertices stab the objects.

(Notice that P can also be just an ordered sequence of points, it does not have to be a polygon.)

We start with showing that Algorithm 1 can be generalized by only increasing the running time by a linear factor. Later on, we consider in detail the cases where the objects are (i) line segments or (ii) disks. For the case of line segments we will also give a faster algorithm.

The basic idea of the algorithm in Section 2.1.1 was to compute the rotation center and a vertex of the stabbing regular n-gon, if it exists. We will use the same idea, however, a general polygon has no rotation center but we can use two specific reference points instead. The first reference point can be any fixed point in the plane. The second reference point is a vertex of P. For simplification we use the centroid of the vertices of P as the first reference point and we denote it by c. As second reference point, we fix any vertex of P, we call this vertex v. All other points of P are now determined by a specific rotation from v around c and a specific scaling of the vector from c to the image point of v. We compute these specific angles and scaling factors in advance. Let the angles be $\alpha_1, \alpha_2, ..., \alpha_n$ and their corresponding scaling factors $f_1, f_2, .., f_n$. Let the vertices of P be $v, v_1, v_2, \ldots, v_{n-1}$ in a given order. Then $\alpha_i = \angle v c v_i$ and the corresponding scaling factor is $f_i = |\overline{cv_i}|/|\overline{cv}|$ for all $i = 1, \ldots, n-1$.

Assume that the centroid c' and a vertex v' of some scaled, translated, and rotated copy of P are given. In order to compute the remaining vertices of the polygon, v' is rotated around c' by the angle α_i and then the vector from c' to this image point of v' is scaled by the factor f_i, $1 \leqslant i \leqslant n$, see also Figure 2.4. (For brevity, we will say that the point itself is scaled, but actually we mean that the corresponding vector from c' to this point is scaled.)

We start now with generalizing the definition of an apex point. For this we need some new notation. Let p, c be points in the plane, α an angle and $f \geqslant 0$ a value. The transformed image of p rotated around c by the angle α and then scaled by the factor f is denoted by $\theta_f^{c,\alpha}(p)$. Consider two points p, q in the plane, an angle α and a value $f \geqslant 0$. We call a point c the *apex point* of p, q, α and f if $\theta_f^{c,\alpha}(p) = q$. For two geometric objects b_i, b_j, an angle α and a value $f \geqslant 0$ we compute for every $p \in b_i$ all

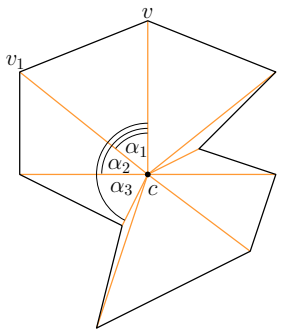


Figure 2.4: The image point of v rotated by an angle of α_1 around c and scaled by a factor of $|\overline{cv_1}|/|\overline{cv}|$ is v_1.

possible apex points c such that $\theta_f^{c,\alpha}(p) \in b_j$. We denote the apex region of b_i, b_j, α and f by $A_{ij}^{\alpha,f}$. Thus,

$$A_{ij}^{\alpha,f} = \{(p,c) \mid c \in \mathbb{R}^2, p \in b_i, \theta_f^{c,\alpha}(p) \in b_j\}$$

As in Section 2.1, we assume that all geometric objects have constant description complexity. Therefore, the apex regions have constant description complexity. In the following, we assume that the apex region can be described with a constant number of (in)equalities of degree a with d variables (note that $d \leqslant 4$ and a is a constant).

Algorithm. The pseudocode of the algorithm is given in Algorithm 2.

Our algorithm proceeds in the following way: Assume that we have already computed the specific angles and scaling factors of P. Let the angles be $\alpha_1, \alpha_2, ..., \alpha_n$ and their corresponding scaling factors $f_1, f_2, .., f_n$. We fix an object, say b_m, and we assume that the reference point v of the resulting polygon lies on b_m (as we do not have this information, we have to repeat the following for every $m = 1, \ldots, n$). Now we use a slightly generalized version of Algorithm 1 as a subroutine. It decides whether there exists a scaled, translated, and rotated copy of P whose vertices stab B and the reference point v lies on b_m. We repeat the modified version of Algorithm 1 for every $m = 1, \ldots, n$ and so we decide whether there exists a scaled, translated, and rotated copy of P whose vertices stab B.

We still have to analyze the running time. We repeat the following for every b_m, $1 \leqslant m \leqslant n$: We consider all possible apex regions $A_{mj}^{\alpha_i,f_i}$ for $1 \leqslant i \leqslant n-1$ and $1 \leqslant j \neq m \leqslant n$. Since there are $n-1$ possible values for i and j, there are $\mathcal{O}(n^2)$ apex regions in total. Each of these regions can be described with a constant number of hypersurfaces of degree a in $\mathbb{R}^d$ (a is a constant and $d \leqslant 4$, as mentioned before). Thus, the arrangement consists of $\mathcal{O}(n^{2d})$ cells (Theorem 2.1.2). We can compute a representative for each cell in $\mathcal{O}(n^{2(d+1)})$ time (Theorem 2.1.3). For each representative we construct the corresponding bipartite graph and decide whether this graph has a perfect matching. Thus, the total running time is $\mathcal{O}(n^{2d+2.5}/\sqrt{\log n})$.

Algorithm 2

Input: Set of geometric objects $B = \{b_1, \ldots, b_n\}$ and a polygon P with n vertices in the plane.

Output: TRUE if there exists a scaled, translated, and rotated copy of P stabbing B, and FALSE otherwise.

for all $m \in \{1, \ldots, n\}$ **do**
  $\mathcal{A} = \emptyset$
  for $j \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** $n-1$ **do**
      if $j \neq m$ **then**
        $\mathcal{A} = \mathcal{A} \cup \{A_{mj}^{\alpha_i, f_i}\}$
  Compute a representative $r = (v, c)$ for each cell of the arrangements $\mathcal{A}$.
  for all r **do**
    //Build corresponding graph $G_r = (V_1 \dot{\cup} V_2, E)$*:*
    $V_1 = \{v_1, v_2, \ldots, v_{n-1}\}$
    // v_i *represents* b'_i *where* $\{b'_1, b'_2, \ldots, b'_{n-1}\} = B \setminus \{b_m\}$
    $V_2 = \{u_1, u_2, \ldots, u_{n-1}\}$, $E = \emptyset$
    // u_i *represents* α_i *and* f_i
    for $i \leftarrow 1$ **to** $n-1$ **do**
      for $k \leftarrow 1$ **to** $n-1$ **do**
        if $\theta_{f_k}^{c, \alpha_k}(v) \in b'_i$ **then**
          $E = E \cup \{(v_i, u_j)\}$
    if $G_r = (V_1 \dot{\cup} V_2, E)$ has a perfect matching **then**
      return TRUE
return FALSE

Since we have to repeat this procedure for $m = 1, 2, \ldots, n$, the total running time is $\mathcal{O}(n^{2d+3.5}/\sqrt{\log n})$.

Recall that the algorithm also computes the reference point v and the centroid c of the scaled, translated, and rotated copy of P, if it exists. Hence, the copy of P can be computed in additional linear time.

Theorem 2.2.1. *Let P be a polygon with n vertices and B be a set of n geometric objects in the plane. Assuming the apex region of two objects in B and an angle can be described with a constant number of polynomial (in)equalities of constant degree with d variables, we can decide in $\mathcal{O}(n^{2d+3.5}/\sqrt{\log n})$ time whether there exists a scaled, translated, and rotated copy of P whose vertices stab B. The copy can be computed in the same time, if it exists.*

2.2.1 Stabbing Line Segments

We consider the problem when the objects are line segments.

> Given a set of line segments $L = \{l_1, l_2, .., l_n\}$ and a polygon P with n vertices in the plane, decide whether there exists a scaled, translated, and rotated copy of P whose vertices stab L.

We consider the apex region for two line segments $l_i, l_j \in L$, an angle α, and a value $f \geqslant 0$:

$$A_{ij}^{\alpha,f} = \{(p, c) \mid c \in \mathbb{R}^2, p \in l_i, \theta_f^{c,\alpha}(p) \in l_j\}$$

This set is characterized by polynomial (in)equalities for $p = (p_1, p_2)$ and $c = (c_1, c_2)$: Without loss of generality, assume that all segments in L are non-vertical. Let $l_i = \{ax + b \mid x \in [x_1, x_2]\}$ and $l_j = \{sx + t \mid x \in [x_3, x_4]\}$. The image point (x', y') of a point (x, y) rotated around c by an angle α and scaled by a factor f is given by $x' = c_1 + f\cos\alpha(x - c_1) - f\sin\alpha(y - c_2)$ and $y' = c_2 + f\sin\alpha(x - c_1) + f\cos\alpha(y - c_2)$ (see, e.g., [57]). Then,

$$p_2 = ap_1 + b \tag{2.7}$$

$$x_1 \leqslant p_1 \leqslant x_2 \tag{2.8}$$

$$c_2 + f\sin(2\pi k/n)(p_1 - c_1) + f\cos(2\pi k/n)(p_2 - c_2) = s(c_1 + f\cos(2\pi k/n)(p_1 - c_1) - f\sin(2\pi k/n)(p_2 - c_2)) + t \tag{2.9}$$

$$x_3 \leqslant c_1 + f\cos(2\pi k/n)(p_1 - c_1) - f\sin(2\pi k/n)(p_2 - c_2) \leqslant x_4 \tag{2.10}$$

We can eliminate p_2 by combining (2.7) and (2.9), and also combining (2.7) and (2.10). Thus, $A_{i,j}^{\alpha,f} \subset \mathbb{R}^3$; there are three variables p_1, c_1, and c_2. Applying Theorem 2.2.1, we get the following result:

Corollary 2.2.2. Let L be a set of n line segments and P a polygon with n vertices in the plane. We can decide in $\mathcal{O}(n^{9.5}/\sqrt{\log n})$ time whether there exists a scaled, translated, and rotated copy of P whose vertices stab L. The copy can be computed in the same time, if it exists.

By studying the geometric properties of these apex regions, we can improve the running time.

Geometric properties of the apex regions. The apex regions are parallelograms in $\mathbb{R}^3$. This can be concluded from the (in)equalities (2.7)–(2.10). We additionally prove it in a geometric way, like in Section 2.1.1.

Recall Lemma 2.1.7 and Lemma 2.1.8; we conclude the following lemmas.

Lemma 2.2.3. *Let l be a line segment and p a point in the plane, let α be an angle and $f \geqslant 0$ a value. The apex region of p, l, α and f is a line segment with length $|l|\sqrt{f^2 + 1 - 2f\cos\alpha}$.*

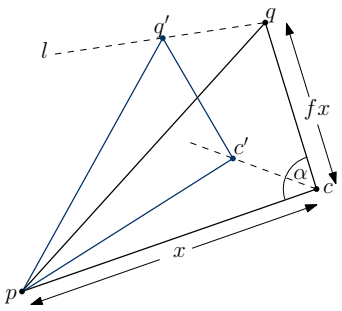


Figure 2.5: $\triangle pqq'$ and $\triangle pcc'$ are similar.

Proof. Let $q, q' \in l$. Let c be the apex point of p, q, α and f. Let c' be the apex point of p, q', α and f. Then $\triangle qcp$ and $\triangle q'c'p$ are similar (Figure 2.5). By exactly the same arguments as in the proof of Lemma 2.1.7 we can show that the apex region of p, l, α and f is a line segment.

The length of the segment can be computed in the following way, see also Figure 2.5: $\frac{|\overline{cc'}|}{|\overline{qq'}|} = \frac{|\overline{pq'}|}{|\overline{pc'}|} = \frac{|\overline{pq}|}{|\overline{pc}|}$. Thus, the length of the segment is $|l|\frac{|\overline{pq}|}{|\overline{pc}|}$. We use the cosine formula to show that $|\overline{pq}|^2 = f^2x^2 + x^2 - 2fx^2\cos\alpha$ and hence the length of the segment is $|l|\sqrt{f^2 + 1 - 2f\cos\alpha}$. This concludes the lemma. □

Lemma 2.2.4. *Let l_i, l_j be two line segments in the plane, α an angle and $f \geqslant 0$ a value. The apex region $A_{i,j}^{\alpha,f}$ is a parallelogram in $\mathbb{R}^3$ with side lengths $|l_i|\sqrt{f^2 + 1 - 2f\cos\alpha}$ and $|l_j|\sqrt{f^2 + 1 - 2f\cos\alpha}$.*

Proof. From the proof of Lemma 2.1.8 follows easily that $A_{i,j}^{\alpha,f}$ is a parallelogram. The side lengths can be proven in the following way: One side has the length of the apex region of a point $p \in l_i$, l_j, α, and f which is $|l_j|\sqrt{f^2 + 1 - 2f\cos\alpha}$. The other side has the length of the apex region of a point $q \in l_j$, l_i, α and f which is $|l_i|\sqrt{f^2 + 1 - 2f\cos\alpha}$. □

Now we can use the same idea as in Section 2.1.1 to improve the running time of the algorithm of Corollary 2.2.2.

Faster Algorithm. Assume that we have already computed the specific angles and scaling factors of P. Let the angles be $\alpha_1, \alpha_2, ..., \alpha_n$ and their corresponding scaling factors $f_1, f_2, .., f_n$. Again, we start by fixing a segment, say l_1, and we assume that $v \in l_1$. As we do not have this information, we repeat the following algorithm for each $l \in L$: The idea is that we also fix a second segment, say l_2 and an integer $\tilde{k} \in \{1, 2, ..., n\}$. We will repeat the following procedure for all possible values for $\tilde{k}$: We want to decide whether there exist a point $v \in l_1$ and a center c such that $\theta_{f_{\tilde{k}}}^{c,\alpha_{\tilde{k}}}(v) \in l_2$ and $\theta_{f_k}^{c,\alpha_k}(v) \in l'_k$ for $1 \leqslant k \neq \tilde{k} \leqslant n-1$ and $\bigcup\{l_i\} = L \setminus \{l_1, l_2\}$. Such a point $v \in l_1$ and a center c exist if the intersection of $A_{12}^{\alpha_{\tilde{k}}, f_{\tilde{k}}}$ and the $n-2$

apex regions of l_1, l'_k and k is non-empty, for $1 \leqslant k \neq \tilde{k} \leqslant n-1$. To this end, we compute $A_{12}^{\alpha_{\tilde{k}},f_{\tilde{k}}}$ and we compute all apex regions $A_{1j}^{\alpha_i,f_i}$ for $1 \leqslant i \neq \tilde{k} \leqslant n-1$ and $3 \leqslant j \leqslant n$. We compute the intersections $A_{12}^{\alpha_{\tilde{k}},f_{\tilde{k}}} \cap A_{1j}^{\alpha_i,f_i}$ and this intersection is either empty, a line segment, or a quadrangle $P \subset A_{12}^{\alpha_{\tilde{k}},f_{\tilde{k}}}$. Thus, the complexity of the resulting arrangement is $\mathcal{O}(n^4)$. Again, we have to decide whether there exists a feasible point $r = (v, c) \in A_{12}^{\alpha_{\tilde{k}},f_{\tilde{k}}}$ that means it belongs to $n-2$ apex regions, with each region being from a different segment with a different angle (and, hence, also a different scaling factor). We can consider the intersection points of the arrangement as possible candidates. For each point we construct the bipartite graph and decide whether it has a perfect matching. Since graphs of neighboring cells differ in only one edge and we know the structure of the arrangement, we can improve the running time by using a dynamic matching algorithm as in Section 2.1.1. We have to check $\mathcal{O}(n^4)$ representative. For each we build a bipartite graph and decide whether the graph has a perfect matching. For the first representative, this takes $\mathcal{O}(n^{2.5})$ time [70], for all other representative this needs only $\mathcal{O}(n^2)$ time. Hence, the running time is $\mathcal{O}(n^6)$. We have to repeat this procedure for every $A_{j2}^{\alpha_{\tilde{k}},f_{\tilde{k}}}, \tilde{k} = 1, \ldots, n-1$ and $j = 1 \ldots, n$ and thus the total running time is $\mathcal{O}(n^8)$.

Theorem 2.2.5. *Let L be a set of n line segments and P a polygon with n vertices in the plane. We can decide in $\mathcal{O}(n^8)$ time whether there exists a scaled, translated, and rotated copy of P whose vertices stab L. The copy can be computed in the same time, if it exists.*

2.2.2 Stabbing Disks

We analyze the running time of Algorithm 2 when the objects are disks. Thus, we consider the following problem:

> Given a set of disks $\mathcal{D} = \{D_1, D_2, .., D_n\}$ and a polygon P with n vertices in the plane, decide whether there exists a scaled, translated, and rotated copy of P whose vertices stab $\mathcal{D}$.

The apex region of D_i, D_j, angle α, and a value $f \geqslant 0$ is defined as

$$A_{i,j}^{\alpha,f} = \{(p, c) \mid c \in \mathbb{R}^2, p \in D_i, \theta_f^{c,\alpha}(p) \in D_j\}$$

This is a set of two polynomial inequalities, one for $p = (p_1, p_2)$ and one for $c = (c_1, c_2)$ as is shown in the following. Let the center of D_i be $a = (a_1, a_2)$ and let the center of D_j be $b = (b_1, b_2)$. Thus, $D_i : \{(x, y)|(x - a_1)^2 + (y - a_2)^2 \leqslant r(D_i)^2\}$ and $D_j : \{(x, y)|(x - b_1)^2 + (y - b_2)^2 \leqslant r(D_j)^2\}$.

Then,

$$\begin{aligned}
&(p_1 - a_1)^2 + (p_2 - a_2)^2 \leqslant r(D_i)^2 \\
&(c_1 + f\cos(2\pi k/n)(p_1 - c_1) - f\sin(2\pi k/n)(p_2 - c_2) - b_1)^2 + \\
&(c_2 + f\sin(2\pi k/n)(p_1 - c_1) + f\cos(2\pi k/n)(p_2 - c_2) - b_2)^2 \leqslant r(D_j)^2
\end{aligned}$$

These are two polynomial inequalities with four variables, thus $A_{i,j}^{\alpha,f} \subset \mathbb{R}^4$. Applying Theorem 2.2.1, we get the following:

Theorem 2.2.6. *Let $\mathcal{D}$ be a set of n disks and a polygon P with n vertices in the plane. We can decide in $\mathcal{O}(n^{11.5}/\sqrt{\log n})$ time whether there exists a scaled, translated, and rotated copy of P whose vertices stab $\mathcal{D}$. The polygon can be computed in the same time, if it exists.*

2.3 Conclusions and Open Problems

We presented algorithms for deciding whether a set of geometric objects can be stabbed with the vertices of a polygon. First we studied the problem of stabbing a set of geometric objects with the vertices of a regular polygon. We also considered an optimization version of the problem which is a restricted version of the approximating symmetry detection problem. Later on we considered the problem of stabbing objects with the vertices of a general polygon. As our running times are rather high, the main remaining question is how these running times can be improved.

Another interesting open problem is the problem of stabbing a set of geometric objects with the boundary of a given polygon. This problem seems to be much more involved than the problems studied here.

Chapter 3

Stabbing with Point Sequences

In this chapter we want to stab sequences of geometric objects by point sequences. We say that a sequence of objects $\mathcal{A} = A_1, \ldots, A_n$ is stabbed by a sequence of points $P = p_1, \ldots, p_n$ if $p_i \in A_i$ for all $1 \leqslant i \leqslant n$. In the following we study the problem of stabbing two sequences of geometric objects with two sequences of points under the condition that these point sequences have a distance as small as possible.

The distance of sequences of points can be measured with the *discrete Fréchet distance*. The Fréchet distance is a natural distance function for continuous shapes such as curves and surfaces [15, 18, 19, 102]. The discrete Fréchet distance is a variant of the Fréchet distance where only the vertices of the polygonal curves are considered, hence the input is a sequence of points. The discrete Fréchet distance between two point sequences $P = p_1, \ldots, p_n$ and $Q = q_1, \ldots, q_m$ is defined as follows: Let a coupling C between P and Q be a sequence of ordered pairs of indices

$$(\pi_1, \rho_1), \ldots, (\pi_l, \rho_l)$$

such that $\pi_1 = \rho_1 = 1$, $\pi_l = n$, $\rho_l = m$, and for all $i = 1, \ldots, l$ one of the following statements is true:

- $\pi_{i+1} = \pi_i$ and $\rho_{i+1} = \rho_i + 1$.
- $\pi_{i+1} = \pi_i + 1$ and $\rho_{i+1} = \rho_i$.
- $\pi_{i+1} = \pi_i + 1$ and $\rho_{i+1} = \rho_i + 1$.

The discrete Fréchet distance is then the minimum over all couplings of $\max_{1 \leqslant i \leqslant l} \operatorname{dist}(p_{\pi_i}, q_{\rho_i})$, in short

$$dF(P, Q) = \min_{C \in \mathcal{C}} \max_{1 \leqslant i \leqslant l} \operatorname{dist}(p_{\pi_i}, q_{\rho_i})$$

where $\mathcal{C}$ is the set of all couplings between P and Q and $\operatorname{dist}(p, q)$ denotes the Euclidean distance between the points p and q. See Figure 3.1.

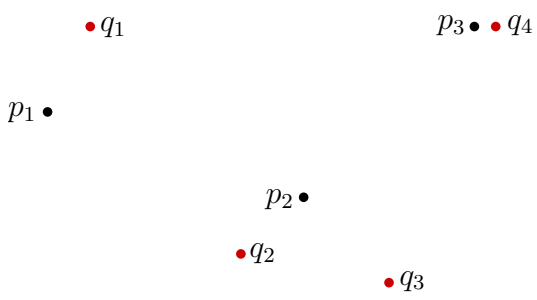


Figure 3.1: The discrete Fréchet distance between $P = p_1, p_2, p_3$ and $Q = q_1, q_2, q_3, q_4$ is achieved by the coupling $(p_1, q_1), (p_2, q_2), (p_2, q_3), (p_3, q_4)$ and so $dF(P, Q) = \text{dist}(p_2, q_3)$.

The discrete Fréchet distance for two point sequences with m and n points, respectively, can be computed in $\mathcal{O}(nm)$ time [52]. Recently, Agarwal et al. [2] presented an algorithm that runs in $\mathcal{O}(\frac{mn \log\log n}{\log n})$ time (assuming $m \leqslant n$).

In Section 3.1 we will consider the problem of stabbing two sequences of segments with two sequences of points under the condition that the discrete Fréchet distance between the point sequences is as small as possible. In Section 3.2 we consider the same problem but here the input is two sequences of disks.

Partial results of this chapter were obtained in collaboration with Ahn, Knauer, Scherfenberg and Vigneron and have been published in [10, 11]. In these papers the problems are studied in a different context, namely, shape matching with *imprecise input*. Most of the previous work on shape matching assumes that the input is given precisely. The input, however, could be only an approximation. In many cases, geometric data come from measurements of continuous real-world phenomenons, and the measuring devices have finite precision. Such impreciseness of geometric data has also been studied, see, e.g., [84] for an overview. Imprecise data can be modeled in different ways. A common way to model data that consist of points is to assign each point to a region, typically a disk or a square. The point itself is then known to lie anywhere in the region. These regions are called *imprecise points* and if we place a point inside a region we call it a *realization* of the imprecise point. Hence, our problem could be stated in the following way: Given two sequences of imprecise points, find two point sequences that are realizations of the imprecise point sequences such that the discrete Fréchet distance between these point sequences is as small as possible. In [10, 11] we solved this problem when the imprecise points are modeled as squares or as disks. (An earlier result of ours, where one of the sequences is precise, i.e., a point sequence, has been published in [12] and can also be found in [103].)

Notation. Let $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ be two sequences of geometric objects in the plane. Let $P = p_1, \ldots, p_n$ and $Q = q_1, \ldots, q_n$ be two point sequences in the plane. The subsequence $A_1, \ldots, A_i$ of $\mathcal{A}$ is denoted by $\mathcal{A}_i$ $(1 \leqslant i \leqslant n)$.

If P stabs $\mathcal{A}$, we denote this by $P \in_S \mathcal{A}$. If $P \in_S \mathcal{A}$ and $Q \in_S \mathcal{B}$, we denote this by $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$. By $(P, Q) \in_{(S,\delta)} (\mathcal{A}, \mathcal{B})$ we mean that $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$ and $dF(P, Q) \leqslant \delta$, for a value $\delta \geqslant 0$.

For an object $A \in \mathcal{A}$ and a value $\delta \geqslant 0$, we denote by $A(\delta)$ the set of points that are at distance at most δ from A.

Recall that $\text{dist}(p, q)$ denotes the Euclidean distance between the points p and q.

3.1 Stabbing Sequences of Line Segments

> Given two sequences of line segments $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ in the plane, compute the minimum value δ such that there exist two point sequences P, Q with $dF(P, Q) = \delta$ and $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$.

We first give an algorithm for the decision version of this problem. Later we show how this decision algorithm can be used in order to get an algorithm for the optimization problem. The decision version can be formulated as follows

> Given two sequences of line segments $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ in the plane and a value $\delta \geqslant 0$, decide whether there exist two point sequences $P = p_1, \ldots, p_n$ and $Q = q_1, \ldots, q_m$ such that $(P, Q) \in_{(S,\delta)} (\mathcal{A}, \mathcal{B})$.

Recall that $A_i(\delta)$ and $B_j(\delta)$ $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m)$ represent the set of points that are at distance at most δ from A_i and B_j, respectively. So, $A_i(\delta) = A_i \oplus C_\delta$ where $\oplus$ is the Minkowski sum and C_δ is a disk with radius δ.

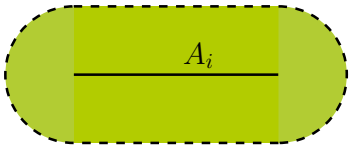


Figure 3.2: $A_i(\delta)$ is marked in green.

$A_i(\delta)$ consists of a rectangle that has two half disks glued at two opposite sides. In the following, we call this geometric object a *tube*.

Decision algorithm. Our decision algorithm is related to the algorithm of Eiter and Mannila to compute the discrete Fréchet distance [52]; it is based on dynamic programming. During the algorithm we build an array with n rows and m columns. In each cell of the array we store two *feasiblity regions*, which indicate where the current points (p_i, q_j) may lie.

While the algorithm proceeds, we fill the cells of the array in an iterative manner from the lower left to the upper right cell, i.e, from $(1,1)$ to (n,m), where the ith row represents the region A_i, and the jth column represents B_j. Each cell (i,j) is filled with two feasibility regions $\mathrm{FA}_\delta(i,j)$ and $\mathrm{FB}_\delta(i,j)$. As we will see in Lemma 3.1.1, the region $\mathrm{FA}_\delta(i,j)$ represents the possible locations of the point p_i under the condition that $(P_i,Q_j) \in_{(S,\delta)} (\mathcal{A}_i,\mathcal{B}_j)$ and there exists a coupling C whose last two pairs are not $(i-1,j),(i,j)$. Similarly, the region $\mathrm{FB}_\delta(i,j)$ represents the possible locations of the point q_j under the condition that $(P_i,Q_j) \in_{(S,\delta)} (\mathcal{A}_i,\mathcal{B}_j)$ and there exists a coupling C whose last two pairs are not $(i,j-1),(i,j)$. Thus, there exist two point sequences P and Q such that $(P,Q) \in_{(S,\delta)} (\mathcal{A},\mathcal{B})$ if and only if the feasibility region $\mathrm{FA}_\delta(n,m)$ or the feasibility region $\mathrm{FB}_\delta(n,m)$ of the upper right cell is non-empty.

We give our decision algorithm in Algorithm 3. The input are two sequences of line segments and a value $\delta \geqslant 0$. The algorithm returns TRUE if there are two point sequences that stab the input and their discrete Fréchet distance is smaller or equal to δ, otherwise the algorithm returns FALSE. The initialization is done in lines 1-8. We introduce an extra zeroth row and column in the array. It allows boundary cases when $i = 1$ and $j = 1$ to be handled correctly in the main loop. The main loop is from line 9 to 16. We give a brief description of how the feasible regions for the cell (i,j) are computed in this loop: The case distinction reflects the definition of the discrete Fréchet distance. Assume that we have already a coupling of ordered pairs $(\pi_1,\rho_1),\ldots,(\pi_k,\rho_k)$, then there are three possible pairs for the next pair in the coupling. First, the next pair could be $(\pi_{k+1},\rho_{k+1}) = (\pi_k+1,\rho_k+1)$. This case corresponds to a diagonal step in the array and the two feasible regions of the new cell are only determined by the location of its two corresponding line segments, i.e., $\mathrm{FA}_\delta(i,j) = A_i \cap B_j(\delta)$ and $\mathrm{FB}_\delta(i,j) = A_i(\delta) \cap B_j$ (line 15 and 16). The second possibility for the next pair is $(\pi_{k+1},\rho_{k+1}) = (\pi_k,\rho_k+1)$ which represents a horizontal step in the array. The third possibility for the next pair is $(\pi_{k+1},\rho_{k+1}) = (\pi_k+1,\rho_k)$ which represents a vertical step in the array. Clearly, for the horizontal step $\mathrm{FA}_\delta(i,j) \subset \mathrm{FA}_\delta(i-1,j)$ and the vertical step $\mathrm{FB}_\delta(i,j) \subset \mathrm{FB}_\delta(i,j-1)$ (line 12 and 13). (Notice that the algorithm first tries to perform a diagonal step. Only when both feasibility regions in the cell $(i-1,j-1)$ are empty, i.e., $\mathrm{FA}_\delta(i-1.j-1) = \emptyset$ and $\mathrm{FB}_\delta(i-1,j-1) = \emptyset$, the algorithm tries to reach the cell (i,j) by a vertical or a horizontal step.) See also Fig. 3.3 for an example of the algorithm.

The following lemma will lead to the correction of the algorithm.

Lemma 3.1.1. *For any $2 \leqslant i \leqslant n$, $2 \leqslant j \leqslant m$, there exist two point sequences P_i and Q_j such that $(P_i,Q_j) \in_{(S,\delta)} (\mathcal{A}_i,\mathcal{B}_j)$ if and only if $\mathrm{FA}_\delta(i,j) \neq \emptyset$ or $\mathrm{FB}_\delta(i,j) \neq \emptyset$. More precisely, for any $x,y \in \mathbb{R}^d$, we have:*

(a) $x \in \mathrm{FA}_\delta(i,j)$ if and only if there exist two point sequences P_i and Q_j such that $(P_i,Q_j) \in_S (\mathcal{A}_i,\mathcal{B}_j)$ and $p_i = x$ and there is a coupling achieving $dF(P_i,Q_j) \leqslant \delta$ whose last two pairs are not $(i-1,j),(i,j)$ (i.e., the last step is not vertical).

Algorithm 3

Input: Two sequences of geometric objects $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ in the plane, and a value $\delta \geqslant 0$.

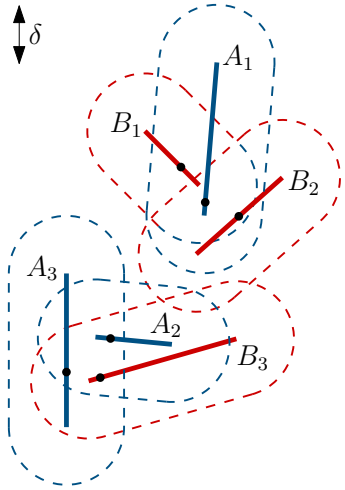
Output: TRUE when there exists two point sequences P and Q such that $(P, Q) \in_{(S,\delta)} (\mathcal{A}, \mathcal{B})$, and FALSE otherwise.

```
1: for i ← 1 to n do
2:     FA_δ(i, 0) ← ∅
3:     FB_δ(i, 0) ← ∅
4: for j ← 1 to m do
5:     FA_δ(0, j) ← ∅
6:     FB_δ(0, j) ← ∅
7: FA_δ(0, 0) ← ℝ²
8: FB_δ(0, 0) ← ℝ²
9: for i ← 1 to n do
10:    for j ← 1 to m do
11:        if FA_δ(i − 1, j − 1) = ∅ and FB_δ(i − 1, j − 1) = ∅ then
12:            FA_δ(i, j) ← FA_δ(i, j − 1) ∩ B_j(δ)
13:            FB_δ(i, j) ← FB_δ(i − 1, j) ∩ A_i(δ)
14:        else
15:            FA_δ(i, j) ← A_i ∩ B_j(δ)
16:            FB_δ(i, j) ← A_i(δ) ∩ B_j
17: if FA_δ(n, m) = ∅ and FB_δ(n, m) = ∅ then
18:    return FALSE
19: else
20:    return TRUE
```

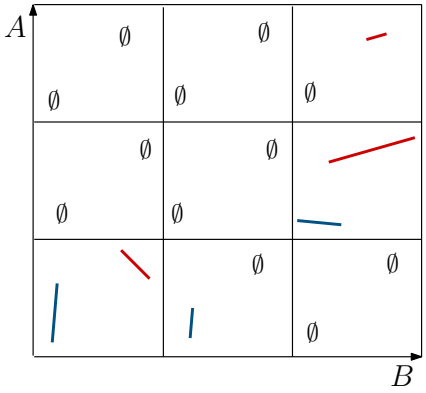
(b) $y \in \mathrm{FB}_\delta(i, j)$ *if and only if there exist two point sequences* P_i *and* Q_j *such that* $(P_i, Q_j) \in_S (\mathcal{A}_i, \mathcal{B}_j)$ *and* $q_j = y$ *and there is a coupling achieving* $dF(P_i, Q_j) \leqslant \delta$ *whose last two pairs are not* $(i, j-1), (i, j)$ *(i.e., the last step is not horizontal).*

Proof. We proof the lemma when $i, j \geqslant 3$. The boundary cases where $i = 2$ or $j = 2$ can be easily checked. We only prove Lemma 3.1.1(a); the proof of (b) is similar. Our proof is done by induction on (i, j). The induction hypothesis is that Lemma 3.1.1 is true for all cells (i', j') that have already been handled by the algorithm before cell (i, j). (Notice that this includes that the assumption is true for all cells (i', j') with $i' \leqslant i$ and $j' \leqslant j$, but not $i' = i$ and $j' = j$.) For the induction step we have to show both directions.

We first assume that $x \in \mathrm{FA}_\delta(i, j)$. We have to show that there exist two point sequences P_i and Q_j such that $(P_i, Q_j) \in_S (\mathcal{A}_i, \mathcal{B}_j)$ and $p_i = x$ and there is a coupling achieving $dF(P_i, Q_j) \leqslant \delta$ whose last two pairs are not $(i-1, j), (i, j)$. There are two



(a) The boundaries of the regions A_i^δ and B_i^δ are dashed. The points represent the stabbing sequences $P \in_R A$ and $Q \in_R B$ where $\mathrm{F}(P,Q) \leqslant \delta$.



(b) In each cell (i,j) $\mathrm{FA}_\delta(i,j)$, is shown in the lower-left corner and $\mathrm{FB}_\delta(i,j)$ in the upper-right corner.

Figure 3.3: Example for Algorithm 3 for sequences of line segments.

cases

- $\mathrm{FA}_\delta(i-1,j-1) \neq \emptyset$ or $\mathrm{FB}_\delta(i-1,j-1) \neq \emptyset$. Then, by induction hypothesis there exist two point sequences P_{i-1}, Q_{j-1} such that $(P_{i-1}, Q_{j-1}) \in_{(S,\delta)} (\mathcal{A}_{i-1}, \mathcal{B}_{j-1})$. The algorithm sets $\mathrm{FA}_\delta(i,j)$ to $A_i \cap B_j(\delta)$ and $\mathrm{FB}_\delta(i,j)$ to $A_i(\delta) \cap B_j$. Since $x \in \mathrm{FA}_\delta(i,j)$ and so $x \in B_j(\delta)$ it follows immediately that there exists $y' \in B_j$ such that $\mathrm{dist}(x,y') \leqslant \delta$. The sequences P_{i-1} and Q_{j-1} can be extended by $p_i = x$ and $q_j = y'$. Because $dF(P_{i-1}, Q_{j-1}) \leqslant \delta$ and $\mathrm{dist}(p_i, q_j) \leqslant \delta$, $dF(P_i, Q_j) \leqslant \delta$ and the last two pairs of the coupling achieving this distance is $(i-1, j-1), (i,j)$.

- $\mathrm{FA}_\delta(i-1,j-1) = \emptyset$ and $\mathrm{FB}_\delta(i-1,j-1) = \emptyset$. Then $\mathrm{FA}_\delta(i,j)$ is set to $\mathrm{FA}_\delta(i,j-1) \cap B_j(\delta)$ (line 12 of the algorithm). It holds that $x \in \mathrm{FA}_\delta(i,j-1)$ because $\mathrm{FA}_\delta(i,j) \subset \mathrm{FA}_\delta(i,j-1)$. By induction hypothesis there are two point sequences P_i and Q_{j-1} such that $(P_i, Q_{j-1}) \in_{(S,\delta)} (\mathcal{A}_i, \mathcal{B}_{j-1})$ and $a_i = x$. Since $x \in B_j(\delta)$ there is a point $y' \in B_j$ such that $\mathrm{dist}(x,y') \leqslant \delta$. The sequence Q_{j-1} can be extended by $q_j = y'$ and so we obtain two sequences P_i and Q_j such that $dF(P_i, Q_j) \leqslant \delta$ and the last two pairs of the coupling achieving this distance is $(i, j-1), (i,j)$.

Now we assume that there exist two point sequences P_i and Q_j such that $(P_i, Q_j) \in_S (\mathcal{A}, \mathcal{B})$ and there exists a coupling achieving $dF(P_i, Q_j) \leqslant \delta$ whose last

two pairs are not $(i-1,j)(i,j)$. We want to show that $p_i \in \mathrm{FA}_\delta(i,j)$. Again, we distinguish between two cases

- $\mathrm{FA}_\delta(i-1,j-1) \neq \emptyset$ or $\mathrm{FB}_\delta(i-1,j-1) \neq \emptyset$. This implies that the algorithm sets $\mathrm{FA}_\delta(i,j)$ to $A_i \cap B_j(\delta)$ (line 14). Since $P_i \in_S \mathcal{A}_i$, we know that $p_i \in A_i$ and since $Q_j \in_S \mathcal{B}_j$ we know that $q_j \in B_j$. We also know that $dF(P_i,Q_j) \leqslant \delta$ and this implies that $\mathrm{dist}(p_i,q_j) \leqslant \delta$. Hence, $p_i \in B_j(\delta)$ and so $p_i \in A_i \cap B_j(\delta) = \mathrm{FA}_\delta(i,j)$.

- $\mathrm{FA}_\delta(i-1,j-1) = \emptyset$ and $\mathrm{FB}_\delta(i-1,j-1) = \emptyset$. Then, $\mathrm{FA}_\delta(i,j)$ is set to $\mathrm{FA}_\delta(i,j-1) \cap B_j(\delta)$ by the algorithm (line 12) and, by induction hypothesis, we know that there do not exist two point sequences P_{i-1}, Q_{j-1} such that $(P_{i-1},Q_{j-1}) \in_{(S,\delta)} (\mathcal{A}_{i-1},\mathcal{B}_{j-1})$. This implies that $dF(P_{i-1},Q_{j-1}) > \delta$ and so the coupling achieving $dF(P_i,Q_j) \leqslant \delta$ cannot contain $(i-1,j-1)$. Hence, the last three pairs of the coupling can either be $(i-1,j-2)(i,j-1)(i,j)$ or $(i,j-2)(i,j-1)(i,j)$. So by induction we have $p_i \in \mathrm{FA}_\delta(i,j-1)$ and, since $dF(P_i,Q_j) \leqslant \delta$, we have that $p_i \in B_j(\delta)$. Thus, $p_i \in \mathrm{FA}_\delta(i,j-1) \cap B_j(\delta) = \mathrm{FA}_\delta(i,j)$.

This concludes the proof. □

The correctness of Algorithm 3 follows immediately from Lemma 3.1.1. It remains to investigate the running time. The initialization step can be done in linear time. The loop takes $\mathcal{O}(nm)$ time since lines 11-16 compute the intersections of line segments with tubes ($A_i(\delta)$ or $B_j(\delta)$). This can be done in constant time. Thus, we can decide in $\mathcal{O}(nm)$ time whether there exists two point sequences P and Q such that $(P,Q) \in_{(S,\delta)} (\mathcal{A},\mathcal{B})$. The point sequences P and Q themselves can be computed in the same running time by tracing back the array containing the feasibility regions (see Algorithm 4). More precisely, we first run Algorithm 3 in order to compute the feasibility regions FA_δ and FB_δ. If Algorihm 3 returns TRUE, we call Algorithm 4 with input $(\mathrm{FA}_\delta, \mathrm{FB}_\delta, n+1, m+1, \mathrm{NULL}, \mathrm{NULL})$.

We achieve the following

Theorem 3.1.2. *Let $\mathcal{A} = A_1,\ldots,A_n$ and $\mathcal{B} = B_1,\ldots,B_m$ be two sequences of line segments in the plane and let $\delta \geqslant 0$ be a value. Then we can decide in $\mathcal{O}(nm)$ time whether there exist two point sequences P and Q such that $(P,Q) \in_{(S,\delta)} (\mathcal{A},\mathcal{B})$. The point sequences P and Q can be computed in the same running time, if they exist.*

Optimization Algorithm. We use the parametrized search technique to achieve an algorithm for the optimization problem. This technique was first introduced by Megiddo [90]. The overall running time using the parametric search technique is $\mathcal{O}(p \cdot T_p + T_p \cdot T_d \log p)$, where p denotes the number of processors, T_d denotes the running time of a decision algorithm, and T_p denotes the running time of a parallel decision algorithm using p processors. Observe that the result of Algorithm 3

Algorithm 4

Input: FA_δ, FB_δ, i, j, p_i, q_j.
Output: The point sequences P_{i-1}, Q_{j-1}.

```
1: procedure printPoints(FA_δ, FB_δ, i, j, p_i, q_j)
2:     if i = 1 and j = 1 then
3:         print " "
4:         return
5:     if FA_δ(i − 1, j − 1) ≠ ∅ then
6:         choose p_{i−1} ∈ FA_δ(i − 1, j − 1) and q_{j−1} ∈ p_{i−1}(δ) ∩ B_{j−1}
7:         print "p_{i−1}" + "q_{j−1}" + printPoints(FA_δ, FB_δ, i − 1, j − 1, p_{i−1}, q_{j−1})
8:         return
9:     if FB_δ(i − 1, j − 1) ≠ ∅ then
10:        choose q_{j−1} ∈ FB_δ(i − 1, j − 1) and p_{i−1} ∈ q_{j−1}(δ) ∩ A_{i−1}
11:        print "p_{i−1}" + "q_{j−1}" + printPoints(FA_δ, FB_δ, i − 1, j − 1, p_{i−1}, q_{j−1})
12:        return
13:    if FA_δ(i, j − 1) ≠ ∅ then
14:        choose q_{j−1} ∈ p_i(δ) ∩ B_{j−1}
15:        print "q_{j−1}" + printPoints(FA_δ, FB_δ, i, j − 1, p_i, q_{j−1})
16:        return
17:    if FB_δ(i − 1, j) ≠ ∅ then
18:        choose p_{i−1} ∈ q_j(δ) ∩ A_{i−1}
19:        print "p_{i−1}" + printPoints(FA_δ, FB_δ, i − 1, j, p_{i−1}, q_j)
20:        return
```

only changes when there is a change in the combinatorial structure of the arrangement of the boundaries of the tubes and segments A_i, $A_i(\delta)$, B_j, $B_j(\delta)$ for all i, j. So, as a generic algorithm, we use an algorithm that computes the arrangement of these $2m+2n$ tubes/segments. There exists such an algorithm with running time $\mathcal{O}(\log nm)$ using $\mathcal{O}(n^2+m^2)$ processors. This algorithm works exactly in the same way as the one for disks [1]: The intersection points of every pair of tubes is computed in constant time with $\mathcal{O}(n^2+m^2)$ processors. Then, we sort around the boundary of each tube the points lying on it (obtaining the edges) in $\mathcal{O}(\log mn)$ using $\mathcal{O}(n+m)$ processors per tube. Finally the edges incident to each vertex of the arrangement are sorted around the vertex.

The decision algorithm is the algorithm from Theorem 3.1.2, which runs in $\mathcal{O}(nm)$ time. So we need a total running time of $\mathcal{O}((m^2+n^2)\log mn + nm\log^2 mn)$.

Theorem 3.1.3. *Let $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ be two sequences of line segments in the plane. Then we can compute the minimum value δ such that there exist two point sequences P, Q with $dF(P,Q) = \delta$ and $(P,Q) \in_S (\mathcal{A},\mathcal{B})$ in $\mathcal{O}((m^2+n^2)\log mn + nm\log^2 mn)$ time. The point sequences P and Q can be computed in the same running time, if they exists.*

3.2 Stabbing Sequences of Disks

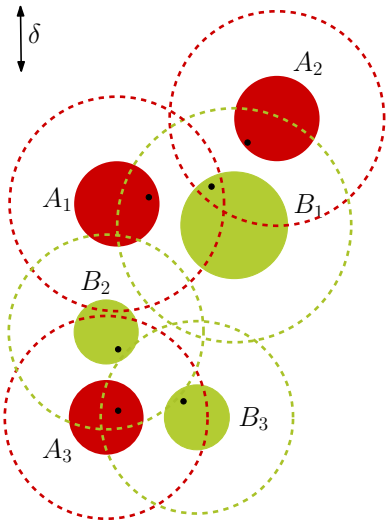
We study the following problem

> Given two sequences of disks $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ in the plane, compute the minimum value δ such that there exist two point sequences P, Q with $dF(P, Q)$ and $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$.

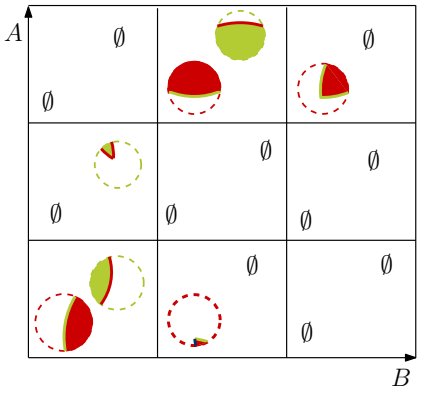
Like in the section before, we first give an algorithm that answers the decision version of this problem. Later on we show how this algorithm can be used in order to solve the optimization problem. The decision problem can be formulated as follows

> Given two sequences of disks $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ in the plane and a value $\delta \geqslant 0$, decide whether there exist two point sequences $P = p_1, \ldots, p_n$ and $Q = q_1, \ldots, q_m$ such that $(P, Q) \in_{(S,\delta)} (\mathcal{A}, \mathcal{B})$.

Decision algorithm. Our decision algorithm works exactly in the same way as in Section 3.1. Thus, the algorithm is given in Algorithm 3. Notice that now $A_i(\delta)$ and $B_i(\delta)$ are disks and, hence, the feasibility regions FA_δ and FB_δ are intersection regions of disks. Figure 3.4 shows an example of the algorithm.



(a) The boundaries of the regions A_i^δ and B_i^δ are dotted. The points represent the stabbing sequences $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$ where $\mathrm{F}(P, Q) \leqslant \delta$.



(b) In each cell (i, j) $\mathrm{FA}_\delta(i, j)$, is shown in the lower-left corner and $\mathrm{FB}_\delta(i, j)$ in the upper-right corner.

Figure 3.4: Example for Algorithm 3 for sequences of disks.

The correctness of the algorithm follows directly from Lemma 3.1.1. (Notice that the proof of Lemma 3.1.1 also works when FA_δ and FB_δ are disks.)

We still have to analyze the running time. The initialization step can be done in linear time. The analysis of the running time for the loop is more involved. A naïve way to implement the loop of Algorithm 3 for disks would require to construct the regions $\mathrm{FA}_\delta(i,j)$ and $\mathrm{FB}_\delta(i,j)$ explicitly for each cell (i,j). Since these regions may be intersections of $\Omega(n)$ disks, this would increase the running time enormously. We improve the running time of line 12-13 to amortized $\mathcal{O}(\log nm)$ time. We will show how line 12 can be implemented in amortized $\mathcal{O}(\log m)$ time, line 13 can be implemented similarly in amortized $\mathcal{O}(\log n)$ time. We fix the value i and we show how to build an incremental data structure that decides in amortized $\mathcal{O}(\log m)$ time whether $\mathrm{FA}_\delta(i,j) = \emptyset$. We do not maintain $\mathrm{FA}_\delta(i,j)$ explicitly, we only maintain a data structure that helps us to decide quickly whether $\mathrm{FA}_\delta(i,j)$ is empty or not. While Algorithm 3 proceeds, either $\mathrm{FA}_\delta(i,j)$ is reset to $A_i \cap B_j(\delta)$ or it is the intersection of $\mathrm{FA}_\delta(i,j-1)$ with $B_j(\delta)$. Hence, at any time $\mathrm{FA}_\delta(i,j) = B_{j'}(\delta) \cap B_{j'+1}(\delta) \cap \cdots \cap B_j(\delta) \cap A_i$ for some $1 \leqslant j' \leqslant j$.

Our data structure needs to perform three types of operations:

1. Set $\mathcal{S} = \emptyset$.
2. Insert the next disk into $\mathcal{S}$.
3. Decide whether the intersection of the disks in $\mathcal{S}$ is empty.

When we run Algorithm 3 on row i, the sequence of the m disks $B_1(\delta), \ldots, B_m(\delta)$ is known in advance, but not the sequence of operations. That is, we know the sequence of disks in advance, but the sequences of operations is given online. Since it can be decided in linear time whether m disks have a non empty intersection [101], a trivial implementation would lead to a running time of $\mathcal{O}(m)$ per operation. We will show now how these operations can be done in amortized $\mathcal{O}(\log m)$ time by using exponential and binary search [95].

Operation 1 is trivial to implement and needs only constant time. Operation 2 and 3 are implemented together. Suppose that the cardinality $|\mathcal{S}|$ of $\mathcal{S}$ is $k = 2^l$ before we perform operation 2. We check whether the intersection of $\mathcal{S}$ with the next k disks is empty or not. This can be done in linear time [101]. If this intersection is empty, we perform a binary search to find the first subsequence of disks, starting at the disks of $\mathcal{S}$, whose intersection is empty. This needs $\mathcal{O}(k \log k)$ time. Then we can perform the operations 2 and 3 in constant time until the next time operation 1 is performed. If the intersection is not empty, operation 2 and 3 can performed in constant time until $|S| = 2^{l+1}$ or the next time operation 1 is performed.

It follows that this data structure needs only amortized $\mathcal{O}(\log m)$ time per operation. Keeping one such data structure for each value of i, we can perform line 12 of Algorithm 3 in amortized $\mathcal{O}(\log m)$ time. Similarly, we can implement line 13 in amortized $\mathcal{O}(\log n)$ time.

Hence, the main loop of the algorithm takes $\mathcal{O}(nm \log nm)$ time.

The point sequences P and Q can be computed in the same running time by tracing back the array computed in Algorithm 3 as before. We then have to compute the feasibility regions explicitly, but only for the diagonal steps of the algorithm (see Algorithm 4). Since there are in total $n + m$ disks and at most $\min(m, n)$ diagonal steps, and computing the intersection of k disks takes $\mathcal{O}(k \log k)$ time [105], this does not increase the running time. Hence, we achieve the following theorem.

Theorem 3.2.1. *Let $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ be two sequences of disks in the plane and $\delta \geqslant 0$ a value. Then we can decide in $\mathcal{O}(nm \log nm)$ time whether there exist two point sequences P and Q such that $(P, Q) \in_{(S,\delta)} (\mathcal{A}, \mathcal{B})$. The point sequences P and Q can be computed in the same time, if they exist.*

Optimization case. We use parametric search in the same way as for Theorem 3.1.3. Observe that the result of Algorithm 3 only changes when there is a change in the combinatorial structure of the arrangement of the circles bounding the disks A_i, $A_i(\delta)$, B_j, $B_j(\delta)$ for all i, j. So, as a generic algorithm, we use an algorithm that computes the arrangement of these $2m+2n$ disks. There exists such an algorithm with running time $\mathcal{O}(\log nm)$ using $\mathcal{O}(n^2 + m^2)$ processors [1]. As a decision algorithm we take the algorithm of Theorem 3.2.1. Hence, we achieve the following result

Theorem 3.2.2. *Let $\mathcal{A} = A_1, \ldots, A_n$ and $\mathcal{B} = B_1, \ldots, B_m$ be two sequences of disks in the plane. Then we can compute the minimum value δ such that there exist two point sequences P, Q with $dF(P, Q) = \delta$ and $(P, Q) \in_S (\mathcal{A}, \mathcal{B})$ in $\mathcal{O}((m^2 + n^2) \log mn + nm \log^3 mn)$ time. The point sequences P and Q can be computed in the same running time, if they exists.*

Higher dimensions. This algorithm can also be implemented in higher dimensions. For d dimensions, the running time increases to $2^{\mathcal{O}(d^2)} m^2 n^2 log^3 mn$. Details can be found in [10, 11].

3.3 Conclusions and Open Problems

We considered the problem of stabbing two sequences of segments (resp. disks) with two point sequences where the discrete Fréchet distance between these point sequences is as small as possible. We gave efficient algorithms for both cases.

An interesting open problem is to find two point sequences that stab the input sequences and their discrete Fréchet distance is as large as possible. Our dynamic programming algorithm does not seem to apply to this case and the problem seems significantly more involved than the problem studied here.

Also, the problem could be studied under other similarity measures. Knauer et al. [81] studied the problem in the context of impreciseness for the *Hausdorff distance*. For the *bottleneck distance* the problem reduces to a simple perfect matching problem.

Part II

Covering Geometric Objects in the Plane

Introduction

This part of the thesis deals with covering geometric objects in the plane. Covering problems are a wide and intensively studied field in computational geometry. Famous examples are *set cover problems* in geometric settings. The original set cover problem is a combinatorial problem: Given a collection C of subsets of a finite set S, find the smallest cover for S in C, i.e., compute the smallest subset $C' \subseteq C$ such that each element of S is contained in $\bigcup C'$. Its decision version is one of Karp's 21 NP-complete problems [75]. Geometric settings of set cover problems have been studied intensively because they arise naturally in many applications, e.g., in sensor networks or facility location problems. For instance, let S be a set of points that represent the set of customers and let C be a set of disks representing the possible location of a base station of a wireless network. The question is to find the minimum number of base stations such that all customers are served by a station. Problems of this kind have been studied in many variations, see [67] and references therein.

In this part we will focus on specific covering problems that can also be considered as finding a good and simple approximation of geometric objects. The idea is that geometric objects are covered or substituted by simpler ones. There are two different ways to do this: The inner approximation and the outer approximation.

Outer approximation: We are given geometric objects and we want to fully cover them with simpler ones while maintaining a certain sense of similarity. A good example is the problem of computing the smallest bounding box [58]. Another example is the *two-center problem*: Given a set P of n points in the plane, find two smallest congruent disks that cover all points in P. The two-center problem is an extensively studied problem. The first subquadratic algorithm was stated by Sharir and has a running time of $\mathcal{O}(n \log^9 n)$ [106]. Currently, the best known deterministic algorithm runs in $\mathcal{O}(n \log^2 n \log^2 \log n)$ time [34] and there is a randomized algorithm with expected running time $\mathcal{O}(n \log^2 n)$ [53]. There has also been a fair amount of work on several variants of the two-center problem: Drezner [45] studied the two-center problem for weighted points, Shin et al. [108] considered the two-center problem for a convex polygon, and Bespamyatnikh et al. [28] studied the rectilinear two-center problem.

Inner approximation: Here, we are given a geometric object and we want to substitute it by a simpler one while still maintaining a certain sense of similarity. In

contrast to the outer approximation, the simple object has to be contained in the original object. Most contributions focus on objects that are polygons. De Pano et al. [42] presented algorithms that find, in $\mathcal{O}(n^2)$ time, the largest equilateral triangle and the largest square in a convex polygon with n vertices. Additionally, they gave an $\mathcal{O}(n^3)$ algorithm that finds the largest equilateral triangle in general polygons. McKenna et al. [89] studied the problem of finding the largest rectangle in a rectilinear polygon and gave an $\mathcal{O}(n \log^5 n)$ algorithm. A famous problem of this area is to compute the largest inscribed rectangle in a convex polygon. Alt et al. [17] presented an $\mathcal{O}(\log n)$ algorithm which computes the largest inscribed isothetic rectangle.

In Chapter 4 we study the two-center problem for disks where the input is a set of disks instead of a set of points. We consider two different versions: the *intersection problem* and the *covering problem*. In the intersection problem we want to compute two smallest congruent disks that together intersect every disk of the input set, whereas in the covering problem we want to compute two smallest congruent disks that together cover all disks of the input set. We present exact and approximation algorithms for these problems. Additionally, we consider a maximization version of this problem.

In Chapter 5 we consider the problem of computing the largest area inscribed rectangle in a convex polygon. We give efficient approximation algorithms and we also investigate the characterization of such a rectangle.

The work of this part is motivated by many applications. For instance, the two-center problem for disks could be considered in the context of impreciseness. The disks may model imprecise data points and our algorithm can be used to perform clustering on such imprecise data. As already mentioned in the first part of this thesis, the impreciseness of geometric data has been studied lately and quite a few geometric algorithms for imprecise points have been published. For example, Löffler and van Kreveld [87] studied the problem of computing the largest and the smallest possible disk containing a set of imprecise points. Other geometric algorithms on imprecise data are, e.g, computing the convex hull for imprecise points [86, 111] and computing the Delaunay triangulation for imprecise points [30, 85]. See also the thesis of Löffler [84] and the references therein for further work on impreciseness.

Another application for this problem is to consider the two-center problem as a facility location problem [45]. The two-center problem for disks models the case of mobile demand points. For instance, it could be used to place two base stations for a wireless network.

Chapter 4

Two Center Problems for Disks

In this chapter we consider the problem of covering a set of geometric objects by simpler ones: Here, the set of geometric objects is a set of n disks in the plane and the *simpler* objects are a pair of two congruent disks.

The problems studied in Section 4.1 and Section 4.2 are closely related to the well-known *two-center problem*: Given a set P of n points in the plane, find two smallest congruent disks that cover all points in P. As already mentioned in the introduction of this part, the two-center problem is well studied and the currently best known algorithm runs in $O(n \log^2 n \log^2 \log n)$ time [34].

The problem studied in this chapter can be considered as a new version of the two-center problem where the input is a set of disks instead of a set of points. We consider two different versions: the *intersection problem* and the *covering problem*. We are given a set $\mathcal{D}$ of disks in the plane. In the intersection problem we want to compute two smallest congruent disks C_1 and C_2 such that each disk in $\mathcal{D}$ intersects C_1 or C_2. In the covering problem we want to compute two smallest congruent disks C_1 and C_2 such that all disks in $\mathcal{D}$ are contained in the union of C_1 and C_2. In the latter we have to distinguish between two cases: Either a disk $D \in \mathcal{D}$ has to be fully covered by C_1 or C_2, we call this the *restricted covering case*, or D can be covered by $C_1 \cup C_2$, which we call the *general covering case*.

Both intersection and covering problems have been studied in the past for the special case where only one smallest disk that intersects, respectively covers, the input disks has to be found. Both problems can be solved in linear time [87] (intersection case), [56, 91] (covering case).

In Section 4.3, we study an optimization version of the intersection and the covering problem. The input is again a set of disks in the plane, but additionally a value $\delta \geqslant 0$ is given. We consider two different versions: The *maximum intersection problem* and the *maximum covering problem*. In the maximum intersection problem we want to compute two disks with radius δ such that these disks intersect the maximum number of the input disks, whereas in the maximum covering problem, these two disks should cover the maximum number of input disks.

To the best of our knowledge, the maximum covering problem has not been studied so far. The maximum intersection problem has been considered by de Berg et al. [39] in the context of continuous facility problems, but under the restriction that the input set is a set of *unit disks*.

The results of Section 4.1 and Section 4.2 are joint work with Hee-Kap Ahn, Sang-Sub Kim, Christian Knauer, Chan-Su Shin, and Antoine Vigneron and have been published in [8, 9].

Notation. The radius of a disk D is denoted by $r(D)$ and its center by $c(D)$. The circle that forms the boundary of D is denoted by ∂D. The Euclidean distance between two points p, q is denoted by $d(p, q)$.

In Section 4.1 and Section 4.2 we assume, without loss of generality, that no disk in $\mathcal{D}$ contains another disk in $\mathcal{D}$. (This is no restriction because if a disk is contained in another disk, we either delete the containing disk for the intersection case or the contained disk for the covering case.) In Section 4.3 and Section 4.4 we drop this assumption.

4.1 Intersection Problem

We start with the intersection problem:

> Given a set $\mathcal{D}$ of n disks in the plane, we want to find two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ has a nonempty intersection with C_1 or C_2.

We can state a simple $\mathcal{O}(n^3)$ algorithm. It is based on the following observation (see also Figure 4.1).

Observation 4.1.1. *Let C_1 and C_2 be a pair of smallest congruent disks intersecting $\mathcal{D}$. Let ℓ be the bisector of the segment connecting $c(C_1)$ and $c(C_2)$. Then, $D \cap C_i \neq \emptyset$ for every $D \in \mathcal{D}$ whose center lies on the same side of ℓ as the center of C_i, for $i = \{1, 2\}$.*

Proof. We prove the property by contradiction. Suppose there is a disk $D \in \mathcal{D}$ whose center lies on the same side of ℓ as the center of C_1 but $D \cap C_1 = \emptyset$. Then $D \cap C_2 \neq \emptyset$ because C_1 and C_2 are a pair of optimal covering disks. But this immediately implies that $D \cap C_1 \neq \emptyset$ since the center of D lies on the same side of ℓ as the center of C_1, and hence $d(c(C_1), c(D)) \leqslant d(c(C_2), c(D))$. □

A simple algorithm, based on this property, works as follows. For every bipartition of the set of centers of the disks in $\mathcal{D}$ by a line ℓ, compute the smallest disks intersecting all disks on each side of ℓ. (If ℓ contains the center of a disk, this disk can be mapped to any side.) There are $\mathcal{O}(n^2)$ possible partitions and the smallest

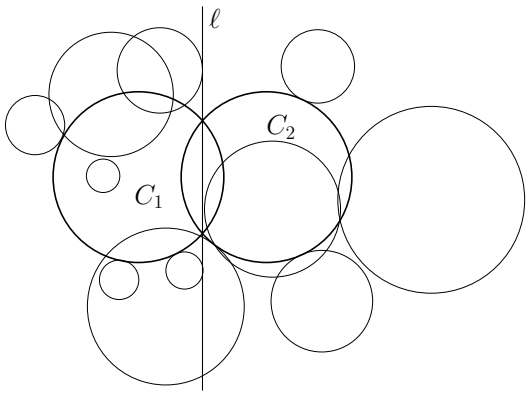


Figure 4.1: A pair of solution disks (C_1, C_2) and ℓ.

disk intersecting a set of disks can be computed in linear time [87]. So in total, this algorithm needs $\mathcal{O}(n^3)$ time.

In order to improve the running time, we start with solving the decision version of the intersection problem. First, we formulate the problem in a different way: For a real number $\delta \geqslant 0$ and a disk D, let the *δ-inflated* disk $D(\delta)$ be a disk with radius $r(D) + \delta$ concentric to D. We can now state the problem in the following way:

> Given a set $\mathcal{D}$ of n disks in the plane and a value $\delta \geqslant 0$, are there two points p_1 and p_2 such that $D(\delta) \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$?

Clearly, there exist two points p_1, p_2 such that $D(\delta) \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$ if and only if the two disks centered at p_1 and p_2 with radius δ intersect all disks $D \in \mathcal{D}$.

Therefore, if an optimal solution of the intersection problem consists of two disks with radius δ^*, then δ^* is the smallest value for which there exist two points such that every disk in $\{D(\delta^*) \mid D \in \mathcal{D}\}$ contains at least one of these points.

4.1.1 Decision Algorithm

We present an efficient algorithm to solve the decision problem stated above. We first state a naïve approach, later we show how this approach can be improved.

Naïve Approach. We are given a set of disks $\{D_1, D_2, ..., D_n\}$ in the plane and a value $\delta \geqslant 0$. We first construct the arrangement of the δ-inflated disks $D_i(\delta)$, $i = 1, 2, .., n$. This arrangement consists of $\mathcal{O}(n^2)$ cells and can be computed in $\mathcal{O}(n^2)$ time [21]. The idea is to traverse all cells in the arrangement in a depth-first manner. To this end, we first construct the dual graph of the arrangement. This graph contains a node for each cell in the arrangement, and two nodes are connected with an edge if the cells have a common boundary. We now perform a depth-first search on the graph and traverse the cells in this manner. While traversing the cells

we do the following: We place one center point, say p_1, in the current cell. Then we decide whether all disks that do not contain p_1 have a nonempty intersection. If they have a nonempty intersection the algorithm returns a positive answer. (The algorithm can additionally return a pair of solution points, this would be p_1 and any point inside the intersection of all disks that do not contain p_1.) If the disks that do not contain p_1 have an empty intersection, we move p_1 to the next cell and repeat the test until we have visited every cell. This approach leads again to a running time of $\mathcal{O}(n^3)$: The arrangement consists of $\mathcal{O}(n^2)$ cells, hence we traverse $\mathcal{O}(n^2)$ cells. For each cell we test whether $\mathcal{O}(n)$ disks have a nonempty intersection, this can be done in linear time [101].

Faster Approach. We show now that the running time of the naïve algorithm can be improved by almost a linear factor. We are given a set of disks $\{D_1, D_2, ..., D_n\}$ in the plane and a value $\delta \geqslant 0$. Again, we start with constructing the arrangement of the δ-inflated disks $D_i(\delta), i = 1, 2, .., n$, and the dual graph of this arrangement. The set of inflated disks is then denoted by $\mathcal{D}$. This time, we consider a traversal of the arrangement of the δ-inflated disks by a path π that visits each cell at least once and crosses only $\mathcal{O}(n^2)$ cells, that is, some cells might be crossed several times but on average each cell is crossed $\mathcal{O}(1)$ times. This can be achieved by choosing the path π to be the Eulerian tour of the depth-first search tree from the naïve approach above, see also Figure 4.2 (by Eulerian tour of a tree, we mean that we double every edge of the tree and take the Eulerian tour of the resulting graph). While we move the point p_1 along π and traverse the arrangement, we want to decide whether the disks that do not contain p_1 have a nonempty intersection. We denote this set of disks by $\mathcal{D}_{\bar{p_1}}$. Notice that while moving p_1 from one cell to the next cell, the set $\mathcal{D}_{\bar{p_1}}$ varies only by one disk. Either a new disk is inserted in $\mathcal{D}_{\bar{p_1}}$ or a disk is removed from $\mathcal{D}_{\bar{p_1}}$.

Next, we use a segment tree [40] in order to decide efficiently whether $\mathcal{D}_{\bar{p_1}}$ has a nonempty intersection. Each disk of $\mathcal{D}$ may appear or disappear several times during the traversal of π: each time we cross the boundary of a cell, one disk is inserted or deleted from $\mathcal{D}_{\bar{p_1}}$. So each disk appears in $\mathcal{D}_{\bar{p_1}}$ along one or several segments of π. We store these segments in a segment tree. (Figure 4.2 and Figure 4.3 show an example of a disk arrangement, the path π and the resulting segment tree.) As there are only $\mathcal{O}(n^2)$ crossings with cell boundaries along π, this segment tree is built over a total of $\mathcal{O}(n^2)$ endpoints and thus has total size $\mathcal{O}(n^2 \log n)$: Each segment of along which a given disk of $\mathcal{D}$ is in $\mathcal{D}_{\bar{p_1}}$ is inserted in $\mathcal{O}(\log n)$ nodes of the segment tree. Each node u of the segment tree stores a set $\mathcal{D}_u \subseteq \mathcal{D}$; from the discussion above, they represent the disks that do not contain p_1 during the whole segment of π that is represented by u and that are not stored at the ancestors of u. In addition, we store at each node u the intersection $I_u = \bigcap \mathcal{D}_u$ of the disks stored at u. Each such intersection I_u is a convex set bounded by $\mathcal{O}(n)$ convex arcs and we store them as an array of circular arcs sorted along the boundary of I_u. In total it takes $\mathcal{O}(n^2 \log^2 n)$

time to compute the intersections I_u for all nodes u in the segment tree, since there are in total $\mathcal{O}(n^2 \log n)$ disks stored at the nodes and the intersection of k disks can be computed in $\mathcal{O}(k \log k)$ time [105].

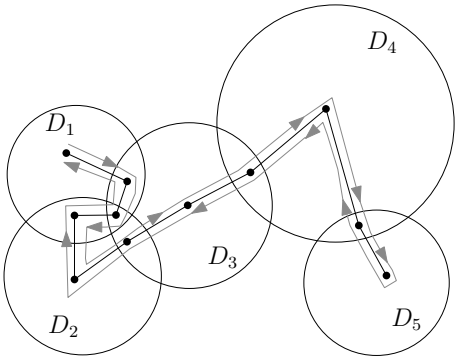


Figure 4.2: The depth-first search tree and the path π. For brevity, we denote the δ-inflated disks $D_i(\delta)$ by D_i.

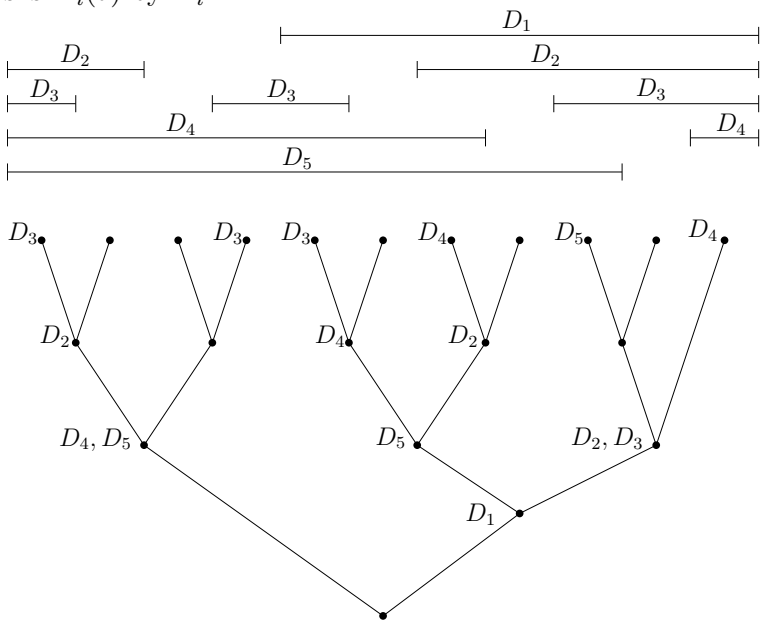


Figure 4.3: The segment tree is depicted for only half of the path π. Since π results from a depth-first search tree, which is a path, the other part of π is identical.

We now need to decide whether at some point, when p_1 moves along π, the intersection of the disks in $\mathcal{D}_{\bar{p_1}}$ (that is, the disks that do not contain p_1) is nonempty. To do this, we use the segment tree. Each cell of the arrangement is represented by a leaf of this tree: All disks that are not stabbed by any point inside this cell are stored at this leaf and all its ancestors. So if p_1 is moved to a cell, all disks stored at the corresponding leaf and its ancestors represent $\mathcal{D}_{\bar{p_1}}$. Hence, we consider each leaf of the segment tree separately. At each leaf, we test whether the intersection of

the disks stored at this leaf and all its ancestors is non-empty. If this intersection is non-empty, the algorithm returns a positive answer. The algorithm can additionally return a pair of solution points, this would be p_1 and any point inside the intersection of all disks $\mathcal{D}_{\bar{p_1}}$.

Recall that we stored at each node u the intersection $I_u = \bigcap \mathcal{D}_u$ of the disks stored at u; each of these intersection is a convex circular polygon with $\mathcal{O}(n)$ circular arcs. So deciding whether the intersection of the disks stored at this leaf and all its ancestors is non-empty reduces to emptiness testing for a collection of $\mathcal{O}(\log n)$ circular polygons with $\mathcal{O}(n)$ circular arcs each.

This problem can also be stated in the following way: Given a collection $\mathcal{C}$ (a subset of $\bigcup I_i$) of $\mathcal{O}(\log n)$ convex circular polygons, decide whether this collection has a nonempty intersection; more precisely, decide whether there exists a point $p \in \bigcap \mathcal{C}$.

We can solve this problem by using convex programming, either randomized or deterministic:

Using the known methods for randomized convex programming [36, 107], we can solve the problem using $\mathcal{O}(\log n)$ expected number of the following primitive operations:

(i) Given I_i, I_j and vector $a \in \mathbb{R}^2$, find the point $v \in I_i \cap I_j$ that minimizes $a \cdot v$.

(ii) Given I_i and a point p, decide whether $p \in I_i$.

Alternatively, we can solve the problem using deterministic convex programming [33]. Then we use $\mathcal{O}(\log n \log\log n)$ of the following primitive operations.

(iii) Given I_i, I_j and vector $a \in \mathbb{R}^2$, find two points $v_1, v_2 \in I_i \cap I_j$ that minimize $a \cdot v_1$ and $-a \cdot v_2$.

(iv) Given I_i and a line ℓ, compute the line segment $I_i \cap \ell$.

(For more details on convex programming, see, e.g, [33].)
Each of the operations (i), (ii), (iii), (iv) needs $\mathcal{O}(\log n)$ time [25, 35, 44, 101]. So we can decide in $\mathcal{O}(\log^2 n)$ expected time or in $\mathcal{O}(n^2 \log^2 n \log\log n)$ deterministic time whether the convex circular polygons have a nonempty intersection.

Summarizing, we have the following:

Lemma 4.1.2. *Let $\mathcal{D}$ be a set of n disks in the plane. We can decide in $\mathcal{O}(n^2 \log^2 n)$ expected time or in $\mathcal{O}(n^2 \log^2 n \log\log n)$ deterministic time whether there exist two points p_1, p_2 such that $D \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$. The points p_1, p_2 can be computed in the same running time, if they exist.*

4.1.2 Optimization Algorithm

In this section, we present an optimization algorithm based on the decision algorithm from the section above. We start by showing that there are only $\mathcal{O}(n^3)$ possible values for the optimal radius. Let δ^* be the radius of an optimal solution for the intersection problem. Recall that δ^* is the smallest value for which there exist two points such that each δ^*-inflated disk contains at least one of them. The following lemma shows that δ^* can be found in a set of $\mathcal{O}(n^3)$ possible values.

Lemma 4.1.3. *Let δ^* be the smallest value for which there exist two points p_1, p_2 such that each δ^*-inflated disk contains at least one of them. Then, either p_1 or p_2 is the tangent point of two δ^*-inflated disks, the common boundary point of three δ^*-inflated disks or $\delta^* = 0$.*

Proof. We prove the lemma by contradiction. Suppose this is not the case. Then the common intersection of the disks containing p_1 has a nonempty interior. Let p_1' be a point in this interior. Similarly, the common intersection of the disks containing p_2 has a nonempty interior and let p_2' be a point in the interior. Then there is a value $\delta' < \delta$ satisfying $D(\delta') \cap \{p_1', p_2'\} \neq \emptyset$ for every $D \in \mathcal{D}$. Since we also assumed that $\delta^* \neq \emptyset$, δ^* could be decreased which leads to a contradiction. □

Due to Lemma 4.1.3 we consider only a set of size $\mathcal{O}(n^3)$ of discrete values in order to find δ^*. The set consists of all values for which one of the events defined in Lemma 4.1.3 occurs. Whether $\delta^* = 0$ can be tested with the decision algorithm of Lemma 4.1.2 in $\mathcal{O}(n^2 \log^2 n)$ expected time or in $\mathcal{O}(n^2 \log^2 n \log\log n)$ worst-case time. Thus, first we check whether $\delta^* = 0$, and then we can assume that p_1 or p_2 is a common boundary point of three δ-inflated disks or a tangent point of two δ-inflated disks.

In order to compute all possible values for δ, we construct a frustum $F_i \in \mathbb{R}^3$ for each disk $D_i \in \mathcal{D}$. The bottom base of F_i is D_i and lies in the plane $z = 0$. The intersection of F_i with the plane $z = \delta$ is $D_i(\delta)$, see also Figure 4.4. The top base is $D_i(\delta_{\max})$ where $\delta_{\max}$ is the radius of the smallest disk intersecting all disks in $\mathcal{D}$. This disk can be computed in linear time [87]. Clearly, $\delta^* \in [0, \delta_{\max}]$.

There is an obvious correspondence between the event points defined in Lemma 4.1.3 and the intersections of the frustums: Consider the case where $p_1 = (x, y)$ is the common boundary point of the disks $D_i(\delta)$, $D_j(\delta)$, and $D_k(\delta)$ in the plane. Then the point $p_1' = (x, y, \delta)$ is the common boundary point of three frustums F_i, F_j, and F_k. Note that the corresponding value for δ is also obtained as it equals to the height of p_1'. Consider now the case that $p_1 = (x, y)$ is the tangent point of $D_i(\delta)$ and $D_j(\delta)$. Then the point $p_1' = (x, y, \delta)$ is the point with the smallest z-value on the intersection curve of F_i and F_j. We call such a point the *tangent point* of two frustums. Again, the corresponding δ value is also obtained, namely the height of p_1'. Hence, in order to find the points p_1 and p_2, all the tangent points and the common boundary points of the frustums have to be considered. There are $\mathcal{O}(n^2)$ tangent points and

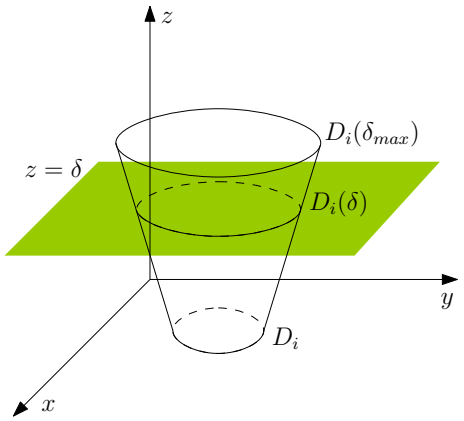


Figure 4.4: The frustum F_i.

$\mathcal{O}(n^3)$ common boundary points, therefore there are $\mathcal{O}(n^3)$ candidates for the point p_1 (resp. p_2) in total.

Now we can already state a naïve algorithm to compute the optimal δ value. First, we compute all possible δ values in $\mathcal{O}(n^3)$ time. Then we check for each candidate value whether there exist two points such that each δ-inflated disk contains at least one of them. This can be done by using the decision algorithm above. The solution is the smallest value for which the decision algorithm returns "yes".

This leads to a running time of $\mathcal{O}(n^5 \log^2 n)$ expected time or $\mathcal{O}(n^5 \log^2 n \log\log n)$ deterministic time. If we perform a binary search on the candidate values for δ the running time can be decreased to $\mathcal{O}(n^3 \log n)$ in both cases, as the running time is dominated by sorting the candidate values.

As computing the candidate points explicitly takes too much time, we use an implicit binary search in order to improve the running time for the randomized algorithm. For the deterministic version we use parametric search.

Implicit Binary Search. We perform an implicit binary search on all possible δ values. As argued before, the possible δ values correspond to the common boundary points of the frustums, more precisely one of the solution points p_1, p_2 is the projection of a point p', which is a tangent point of two frustums or the common boundary point of three frustums. Hence, the point p' is a vertex of the arrangement $\mathcal{A}$ of the n frustums $F_1, F_2, ..., F_n$. The complexity of $\mathcal{A}$ is $\mathcal{O}(n^3)$. We now descibe how to perform the binary search over the vertices of $\mathcal{A}$ in an implicit way:

Binary Search on a Coarse List of Events. We consider $\mathcal{O}(n^2)$ pairs of frustums and compute the tangent points of each pair. We also randomly select $\mathcal{O}(n^2 \log n)$ triples of frustums and compute the common boundary point of each triple. This takes $\mathcal{O}(n^2 \log n)$ time. Since the optimal value δ^* for δ lies in $[0, \delta_{\max}]$ we only consider points whose z-value lies in this interval. These points are clearly vertices of $\mathcal{A}$

and thus we randomly select $\mathcal{O}(n^2 \log n)$ vertices of $\mathcal{A}$. We sort their radii associated with them in $\mathcal{O}(n^2 \log^2 n)$ time. Then we perform a binary search on these values with the decision algorithm of Lemma 4.1.2 and we determine two consecutive radii δ_i and δ_{i+1} such that δ^* is between δ_i and δ_{i+1}. This takes $\mathcal{O}(n^2 \log^3 n)$ time. Since the vertices were picked randomly, the strip $S[\delta_i, \delta_{i+1}]$ bounded by the two planes $z := \delta_i$ and $z := \delta_{i+1}$ contains only $k \in \mathcal{O}(n)$ vertices of $\mathcal{A}$ with probability larger than $1/2$ [96, Theorem 5.1.2].

Zooming into the Interval. We have to compute all k vertices of $\mathcal{A}$ in $S[\delta_i, \delta_{i+1}]$. This can be done in $\mathcal{O}(k \log n + n^2 \log n)$ time by using a standard sweep-plane algorithm: First, we compute the intersection of the frustums $F_1, F_2, .., F_n$ with the sweep-plane $z = \delta_i$. This intersection forms an arrangement of $\mathcal{O}(n)$ disks in $\mathbb{R}^2$. The complexity of the arrangement is $\mathcal{O}(n^2)$ time and it can be computed by simply computing the arrangement of the δ_i-inflated disks $D_j(\delta_i), j = 1, 2, .., n$. We construct the portion of the arrangement $\mathcal{A}$ in $S[\delta_i, \delta_{i+1}]$ incrementally by sweeping a plane orthogonal to the z-axis from the intersection at $z = \delta_i$ towards $z = \delta_{i+1}$. As a result, we can compute the $k \in \mathcal{O}(n)$ vertices (and their corresponding radii) in $S[\delta_i, \delta_{i+1}]$ in $\mathcal{O}(n \log n)$ time. If the number of the k vertices inside the strip becomes too large, we abort the sweep and restart the algorithm with a new random sample. This only happens with small probability.

In order to find the minimum value δ^*, we perform a binary search on these $\mathcal{O}(n)$ radii we just computed, using the decision algorithm in Lemma 4.1.2. This takes $\mathcal{O}(n^2 \log^3 n)$ expected time. The solution points p_1 and p_2 can also be computed by the decision algorithm.

Parametric search. To get a deterministic algorithm, we use the parametric search technique of Megiddo [90]. The overall running time using the parametric search technique is $\mathcal{O}(p \cdot T_p + T_p \cdot T_d \log p)$, where p denotes the number of processors, T_d denotes the running time of a decision algorithm, and T_p denotes the running time of the parallel decision algorithm using p processors. As generic algorithm, we use a parallel algorithm that computes in $\mathcal{O}(\log n)$ time the arrangement of the inflated disks using $\mathcal{O}(n^2)$ processors [1]. And so we need to run the decision algorithm $\mathcal{O}(\log^2 n)$ times, and the total running time becomes $\mathcal{O}(n^2 \log^4 n \log \log n)$.

Thus, we have proved the following theorem:

Theorem 4.1.4. *Given a set $\mathcal{D}$ of n disks in the plane, we can compute two smallest congruent disks whose union intersects every disk in $\mathcal{D}$ in $\mathcal{O}(n^2 \log^3 n)$ expected time, and in $\mathcal{O}(n^2 \log^4 n \log \log n)$ deterministic time.*

4.2 Covering Problem

In this section we consider the covering problem:

> Given a set $\mathcal{D}$ of n disks in the plane, compute two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ is covered by C_1 or C_2.

We distinguish between two cases:

- In the *general case*, each disk $D \in \mathcal{D}$ must be covered by $C_1 \cup C_2$.
- In the *restricted case*, each disk $D \in \mathcal{D}$ has to be fully covered by C_1 or by C_2.

Figure 4.5 shows examples for both cases.

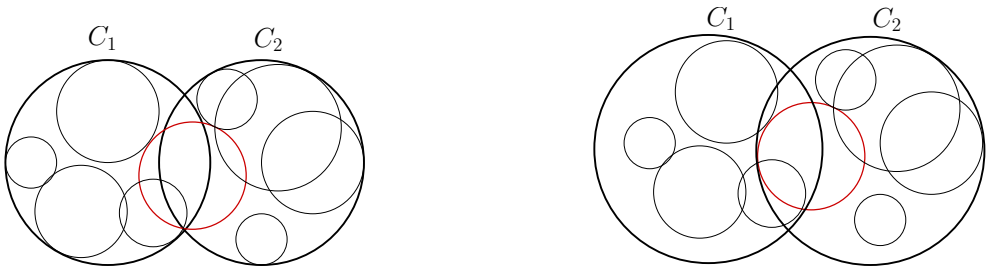


Figure 4.5: From left to right: A pair of solution disks for the *general* covering problem. Notice that the red disk is covered by $C_1 \cup C_2$ but not fully covered by only one solution disk. On the right: A pair of solution disks for the *restricted* covering problem for the same set of input disks.

4.2.1 The Restricted Case

We start with the restricted covering problem.

> Given a set $\mathcal{D}$ of n disks in the plane, compute two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ is covered by C_1 or C_2.

Observation 4.1.1 can be adapted to the restricted covering case.

Observation 4.2.1. *Let C_1 and C_2 be a pair of smallest congruent disks covering $\mathcal{D}$. Let ℓ be the bisector of the segment connecting $c(C_1)$ and $c(C_2)$. Then, $D \subset C_i$ for every $D \in \mathcal{D}$ whose center lies on the same side of ℓ as the center of C_i, for $i = \{1, 2\}$.*

The proof is a straightforward adaption of the proof of Observation 4.1.1.

It follows immediately that the restricted covering problem can be solved in $\mathcal{O}(n^3)$ time: For every bipartion of the set of centers of the disks in $\mathcal{D}$ by a line ℓ, compute the smallest disk covering all disks on each side of ℓ. This takes $\mathcal{O}(n^3)$ time, since

there are $\mathcal{O}(n^2)$ possible partitions and the smallest disk covering a set of disks can be found in linear time [91].

Now we show that the algorithm from Section 4.1 can also be adapted to solve the restricted covering problem. We consider the decision version of the restricted covering problem:

> Given a set $\mathcal{D}$ of n disks in the plane and a value δ, decide whether there exist two disks C_1 and C_2 of radius δ such that each disk $D \in \mathcal{D}$ is covered by either C_1 or C_2.

This implies that for each disk $D \in \mathcal{D}$ covered by C_i, the following holds: $d(c(D), c(C_i)) + r(D_j) \leqslant \delta$, for $i = \{1, 2\}$. Let $r_{\max}$ be the maximum of radii of all disks in $\mathcal{D}$. It holds that $\delta \geqslant r_{\max}$, since if $\delta < r_{\max}$ there clearly exist no two disks with radius δ which cover $\mathcal{D}$. Hence, we can formulated the decision problem in a different way.

> Given a value δ, do there exist two points, p_1 and p_2, in the plane such that $D^*(\delta) \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$, where $D^*(\delta)$ is the disk concentric to D with radius $\delta - r(D) \geqslant 0$?

Recall the definition of δ-*inflated* disks from Section 4.1. Every disk $D \in \mathcal{D}$ was replaced by a disk concentric to D with radius $r(D)+\delta$. Here we need to replace each disk D by a disk with radius $\delta - r(D)$ concentric to D. Since we know that $\delta \geqslant r_{\max}$, we add an initialization step, in which every disk D is replaced by a disk concentric to D with radius $r_{\max} - r(D)$. Then we can use exactly the same decision algorithms as in Section 4.1 and, hence, also the same optimization algorithms in order to compute a solution for the restricted covering problem: Let δ^* be the solution value computed by this algorithm. Then $\delta^* + r_{\max}$ is a solution for the covering problem. We summarize this result in the following theorem.

Theorem 4.2.2. *Given a set $\mathcal{D}$ of n disks in the plane, we can compute two smallest congruent disks such that each disk in $\mathcal{D}$ is covered by at least one of them in $\mathcal{O}(n^2 \log^3 n)$ expected time or in $\mathcal{O}(n^2 \log^4 n \log\log n)$ worst-case time.*

4.2.1.1 Constant Factor Approximation

In this section we develop efficient constant factor approximation algorithms. We first present a 2-approximation algorithm, which runs in $\mathcal{O}(n)$ time. The algorithm is based on the well known greedy k-center approximation algorithm by Gonzalez [61]. We give the algorithm in Algorithm 5.

The distance between two disks D_1 and D_2 is denoted by $d'(D_1, D_2)$ and is defined as $d'(D_1, D_2) = d(c(D_1), c(D_2)) + r(D_1) + r(D_2)$.

The algorithm OneCover($\mathcal{U}$), which is used as a subroutine, computes the smallest disk covering a set of disks $\mathcal{U}$.

Algorithm 5

Input: A set of disks $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ in the plane.

Output: Two disks C_1, C_2 such that each disk in $\mathcal{D}$ is covered by C_1 or C_2 and $\max(r(C_1), r(C_2)) \leqslant 2$ times the radius of the optimal solution.

1: $\mathcal{U}_1 = \emptyset$ and $\mathcal{U}_2 = \emptyset$.
2: Compute the disk $D_l \in \mathcal{D}$ such that $d'(D_1, D_l) \geqslant d'(D_1, D_j)$, $\forall D_j \in \mathcal{D}$.
3: $\mathcal{U}_1 = \mathcal{U}_1 \cup \{D_1\}$ and $\mathcal{U}_2 = \mathcal{U}_2 \cup \{D_l\}$.
4: **for all** $D_j \in \mathcal{D}$ **do**
5: **if** $d'(D_j, D_1) < d'(D_j, D_l)$ **then**
6: $\mathcal{U}_1 = \mathcal{U}_1 \cup \{D_j\}$
7: **else**
8: $\mathcal{U}_2 = \mathcal{U}_2 \cup \{D_j\}$
9: Compute $C_1 = \text{OneCover}(\mathcal{U}_1)$ and $C_2 = \text{OneCover}(\mathcal{U}_2)$
10: **return** C_1 and C_2

Note that Algorithm 5 returns two disks with different radii. If we want the disks to be congruent, we have to add a step to decide which disk has the maximum radius and enlarge the smaller disk accordingly.

Since computing the disk D_l which has the largest distance to D_1 takes linear time, and OneCover($\mathcal{U}$) takes $\mathcal{O}(n)$ time [91], the total running time of the algorithm is $\mathcal{O}(n)$. The approximation factor of 2 can be easily proven: Assume that $r(C_1) \geqslant r(C_2)$ and let D_j be the disk in $\mathcal{U}_1$ that has the largest distance $d_{\max}$ to D_1, $d'(D_1, D_j) = d_{\max}$, and so $d'(D_j, D_l) \geqslant d_{\max}$. Then, $r(C_1) \leqslant d_{\max}$ but, since $d'(D_1, D_l) \geqslant d_{\max}$, the optimal solution disks have a radius $\geqslant d_{\max}/2$ and so our algorithm computes a 2-approximation.

Theorem 4.2.3. *Given a set $\mathcal{D}$ of n disks in the plane, for the restricted covering problem a 2-approximation can be computed in $\mathcal{O}(n)$ time.*

Notice that if the solution disks C_1^*, C_2^* are far apart, i.e., $d'(C_1^*, C_2^*) > 6r(C_1^*)$, this algorithm computes an optimal solution. All disks of $\mathcal{D}$ covered by C_1^* have a distance larger than $2r(C_1^*)$ to any point in C_2^*. The analog holds for the disks of $\mathcal{D}$ covered by C_2^*. Thus, the partition of the disks in $\mathcal{D}$ then leads to the partition of the optimal solution.

(Notice that this algorithm improves the approximation algorithm in [9, Theorem 13]. Both algorithms have the same running time, but here the approximation factor is 2 while in [9, Theorem 13] it is 6.)

4.2.1.2 $(1+\epsilon)$-Approximation

Recall Observation 4.2.1. Let C_1^* and C_2^* be a pair of smallest congruent disks covering $\mathcal{D}$. Let ℓ^* be the bisector of the segment connecting the centers of C_1^* and

C_2^*. Then each disk $D \in \mathcal{D}$ is covered by the disk C_i^* whose center lies on the same side of ℓ^* as the center of C_i^*, for $i = \{1, 2\}$.

If we know this bisector ℓ^* we know the bipartition of the disks and, hence, we can compute the optimal covering disks. We now start with computing an optimal pair of covering disks under the assumption that the direction of the bisector ℓ^* is known. We present an algorithm with running time $\mathcal{O}(n \log n)$. Later on we explain how to obtain a $(1+\epsilon)$-approximation by using this algorithm.

Fixed Orientation. Assume the direction of the bisector ℓ^* is given and without loss of generality it is vertical. We first sort the centers of all $D \in \mathcal{D}$ by their x-coordinates. If there are disks having the same x-coordinate, we sort them lexicographically, first by their x- and then by their y-coordinate. Then we sweep a vertical line ℓ from left to right and we maintain two set of disks: $\mathcal{D}_1$ is the set of disks whose centers lie to the left of ℓ and $\mathcal{D}_2 = \mathcal{D} \setminus \mathcal{D}_1$.

Let C_1 be the smallest disk covering $\mathcal{D}_1$ and C_2 the smallest disk covering $\mathcal{D}_2$. The crucial property of C_1 and C_2 is the following: While sweeping ℓ from left to right, the radius of C_1 is nondecreasing and the radius of C_2 nonincreasing. We want to compute $\min \max(r(C_1), r(C_2))$. This can be done by performing a binary search on the sorted list of the centers of the disks in $\mathcal{D}$. In each step we compute the radius of C_1 and C_2, which takes $\mathcal{O}(n)$ time [91]. Thus, the total running time is $\mathcal{O}(n \log n)$.

In the following we explain how this algorithm can be used in order to get a $(1+\epsilon)$-approximation for the restricted covering problem. The main idea is to find a set of good sample orientations.

Sampling. We use $2\pi/\epsilon$ sample orientations chosen regularly over 2π, and compute, for each orientation, the solution in $\mathcal{O}(n \log n)$ time. The approximation factor can be proven by showing that there is a sample orientation that has an angle at most ϵ with the optimal bisector. First notice that we can assume that $\epsilon < 1$, otherwise we can use the approximation algorithm above which leads to a better solution. Let b be the bisector of a pair of optimal solution disks C_1^* and C_2^*. Without loss of generality we assume that b is vertical as in Figure 4.6. Let q denote the midpoint of $c(C_1^*)$ and $c(C_2^*)$.

Let ℓ be a line which passes through q and that makes an angle at most ϵ with b in counterclockwise direction as in Figure 4.6. (To simplify the presentation, the angle in the calculation is set to exactly ϵ). Let p_1 be the intersection point of ℓ with the upper circular arc of ∂C_1^*. Let p_2 be the point symmetric to p along b. Clearly p_2 lies on the boundary of C_2^*. We will show that there exist two disks C_1 and C_2 where C_1 covers all disks whose center lie to the left of ℓ and C_2 covers all disks whose center lie to the right of ℓ and $r(C_1) = r(C_2) \leqslant (1+\epsilon) r(C_1^*)$.

We will explain the construction of C_2 and prove that C_2 covers all disks whose centers lie to the right of ℓ. C_1 can be constructed analogously. The center of C_2

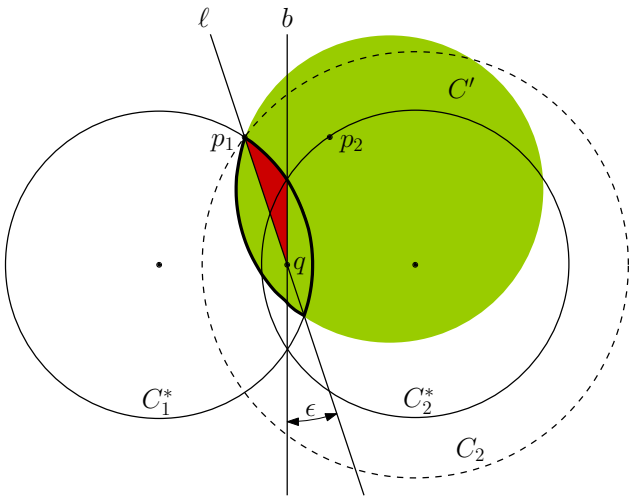


Figure 4.6: $r(C_2) \leqslant (1+\epsilon')r(C_2^*)$ for any $\epsilon' \geqslant 6\epsilon$.

is set to $c(C_2^*)$ and its radius is set to $|\overline{c(C_2^*)p_1}| \leqslant |\overline{c(C_2^*)p_2}| + |\overline{p_2p_1}|$. It holds that $|\overline{c(C_2^*)p_2}| + |\overline{p_2p_1}| \leqslant r(C_2^*) + 6r(C_2^*)\sin\epsilon$, since $|\overline{qp_1}| = |\overline{qp_2}| \leqslant 3r(C_1^*)$ and so the distance between p_2 and b is at most $3r(C_2^*)\sin\epsilon$. Clearly C_2 covers all disks that were covered by C_2^*. In addition, it must cover all disks whose center lie in the region of C_1^* that is bounded by ℓ and b and has q as its lowest point (this region is shown in red in Figure 4.6). Note that the disks whose centers lie in this region are fully covered by C_1^*, but not necessarily by C_2^*.

It remains to prove that all disks having their center in the red region are fully covered by C_2. Let C' be the disk symmetric to C_1^* along ℓ. Then all disks whose center lie in the red region are covered by $C_1^* \cap C'$, because this region is symmetric along ℓ and they are fully covered by C_1^*. Since C_2 contains the intersection $C_1^* \cap C'$, we conclude that all disks whose centers lie on the right side of ℓ are covered by C_2. We can prove the analog for C_1.

Hence,

$$r(C_1) = r(C_2) \leqslant (1+6\sin\epsilon)r(C_1^*) \leqslant (1+\epsilon')r(C_1^*) = (1+\epsilon')r(C_2^*)$$

as $\sin\epsilon \leqslant \epsilon$ for $\epsilon \leqslant 1$ (this can be shown by using the Taylor series: $\sin x = \sum_{n=0}^{\infty}(-1)^n x^{2n+1}/(2n+1)! = x - x^3/3! + x^5/5! - ...$) and for any $\epsilon' \geqslant 6\epsilon$.

Since any solution whose bisector is parallel to ℓ has a radius of at most $r(C_1)$, this solution has radius at most $(1+\epsilon')$ times the optimal radius.

Hence, we have the following result.

Theorem 4.2.4. *Given a set $\mathcal{D}$ of n disks in the plane, a $(1+\epsilon)$-approximation for the restricted covering problem for $\mathcal{D}$ can be computed in $\mathcal{O}((n/\epsilon)\log n)$ time.*

We can improve the running time to $\mathcal{O}(n + 1/\epsilon^3 \log 1/\epsilon)$ by using an adaption of the algorithm by Agarwal and Procopiuc [4].

We start by computing a 2-approximation in $\mathcal{O}(n)$ time using our algorithm from Theorem 4.2.3. Let C_1^{apx}, C_2^{apx} be the disks computed by this approximation algorithm and let r^{apx} be their radius. We consider a grid of size $\delta = \lambda\epsilon r^{\max} \leqslant 2\lambda\epsilon r^*$ over the plane, where λ is a small enough constant. That is, we consider the points with coordinates $(i\delta, j\delta)$ for some integers i, j. Observe that there are only $\mathcal{O}(1/\epsilon^2)$ grid points in $C_1^{\text{apx}} \cup C_2^{\text{apx}}$. The center of each disk D is moved to a nearby grid point. That is, a center (x, y) is replaced by $(\delta\lceil x/\delta\rceil, \delta\lceil y/\delta\rceil)$. If two or more centers are moved to the same grid point, we only keep the disk with the largest radius. We denote this new set by $\mathcal{D}_g$ and this set consists of $\mathcal{O}(1/\epsilon^2)$ disks; all centers of disks in $\mathcal{D}_g$ are grid points inside $C_1^{\text{apx}} \cup C_2^{\text{apx}}$ or at distance at most $\sqrt{2}\delta$ from the boundary of this union.

We compute a $(1+\epsilon')$-approximation for the set $\mathcal{D}_g$ in $\mathcal{O}((1/\epsilon^3)\log 1/\epsilon)$ time by applying our algorithm from Theorem 4.2.4. These computed disks have a radius $\leqslant (1+\epsilon')(r^* + \sqrt{2}\delta)$ and we inflate these disks by a factor of $\sqrt{2}\delta$. So the radii of these inflated disks are $\leqslant (1+\epsilon')(r^* + \sqrt{2}\delta) + \sqrt{2}\delta \leqslant (1+\epsilon)r^*$ (if we choose $\lambda \leqslant 1/(8\sqrt{2})$, $\epsilon' = 1/4\epsilon$, and $\epsilon \leqslant 1$). Hence, these disks are a $(1+\epsilon)$-approximation for the optimal solution for $\mathcal{D}$.

Theorem 4.2.5. *Given a set $\mathcal{D}$ of n disks in the plane, a $(1+\epsilon)$ approximation for the restricted covering problem for $\mathcal{D}$ can be computed in $\mathcal{O}(n + (1/\epsilon^3)\log 1/\epsilon)$ time, for any $0 < \epsilon \leqslant 1$.*

4.2.2 The General Case

In this section we study the general covering case.

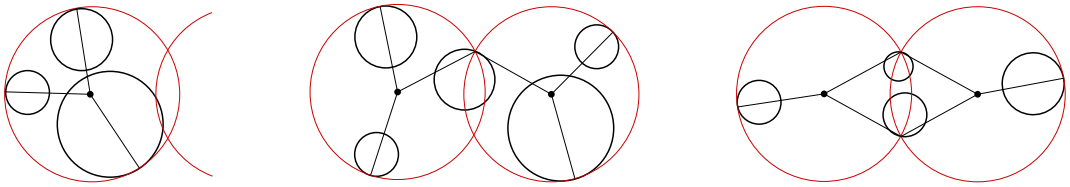
> Given a set $\mathcal{D}$ of n disks in the plane, compute two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ is covered by $C_1 \cup C_2$.

We start with giving a characterization of the optimal covering. The optimal covering of a set $\mathcal{D}'$ of disks with one disk is determined by at most three disks of $\mathcal{D}'$ touching the covering disk such that the convex hull of the contact points contains the center of the covering disk. (See Figure 4.7(a).)

When covering by two disks, a similar argument applies, and thus the optimal covering disks (C_1^*, C_2^*) are determined by at most five input disks:

Lemma 4.2.6. *The optimal covering by two disks C_1^*, C_2^* satisfies one of the following conditions.*

1. *For some $i \in \{1, 2\}$, the disk C_i^* is the optimal one-covering of the disks contained in C_i^*, as in Figure 4.7(a).*



(a) (b) (c)

Figure 4.7: The three configurations for the optimal 2-center covering of disks.

2. *There is an input disk that is neither fully contained in C_1^* nor in C_2^*, but contains one point of $\partial C_1^* \cap \partial C_2^*$ on its boundary as in Figure 4.7(b).*

3. *There are two input disks D_i, D_j (possible $i = j$) none of them being fully covered by C_1^* or C_2^*, such that D_i contains one point of $\partial C_1^* \cap \partial C_2^*$ on its boundary and D_j contains the other point of $\partial C_1^* \cap \partial C_2^*$ on its boundary as in Figure 4.7(c).*

In all cases, both covering disks are determined by at most three disks; for one covering disk C^*, it holds that the contact points of the determining disks contain the center $c(C^*)$ in their convex hull.

Proof. The optimal solution is a pair of congruent disks that achieves a local minimum in radius, that is, we cannot reduce the radius of the covering disks by translating them locally. If one covering disk is completely determined by the input disks contained in it, then case 1 applies. Otherwise, there always exists at least one input disk D such that D is not contained in C_i^* for all $i \in \{1, 2\}$. Moreover such input disks always touch $C_1^* \cup C_2^*$ from inside at the intersection points of ∂C_1^* and ∂C_2^*, otherwise we can always get a pair of smaller congruent covering disks. If only one point of $\partial C_1^* \cap \partial C_2^*$ is touched by an input disk D, both covering disks are determined by at most two additional disks touching from inside together with D because the covering disks are congruent. If both intersection points of $\partial C_1^* \cap \partial C_2^*$ are touched by input disks D_i and D_j, possible $i = j$, one covering disk is determined by one additional disk and the other covering disk by at most one additional disk touching from inside together with D_i and D_j because the covering disks are congruent. It is not difficult to see that for one covering disk C^*, it holds that the contact points of the determining disks contain the center $c(C^*)$ in their convex hull; otherwise we can get a pair of smaller congruent covering disks. □

Using a decision algorithm and the parametric search technique, we can construct an exact algorithm for the general covering problem. The main part of this algorithm was invented by my coauthors Hee-Kap Ahn, Chan-Su Shin, and Antoine Vigneron. We still state the basic ideas in this thesis since they nicely complement the results of this section.

Let r^* be the radius of an optimal solution for the general case of covering by two disks. The decision algorithm is based on the following lemma that, for a given $r > 0$, returns "yes" if $r \geqslant r^*$, and "no" otherwise. (See also Figure 4.8).

Lemma 4.2.7 ([9, Lemma 6]). *Assume that $r \geqslant r^*$. Then there exists a pair of congruent disks C_1, C_2 of radius r such that their union contains the input disks, an input disk D touches C_1 from inside, and one of the following property holds.*

(a) C_1 is identical to D.

(b) There is another input disk touching C_1 from inside.

(c) There is another input disk D' not contained in C_2, but it touches a common intersection t of ∂C_1 and ∂C_2 that is at distance $2r$ from the touching point of D. If this is the case, we say that D and t are aligned with respect to C_1.

(d) There are two disks D_i and D_j, possibly $i = j$, such that D_i touches a common intersection of ∂C_1 and ∂C_2, and D_j touches the other common intersection of ∂C_1 and ∂C_2.

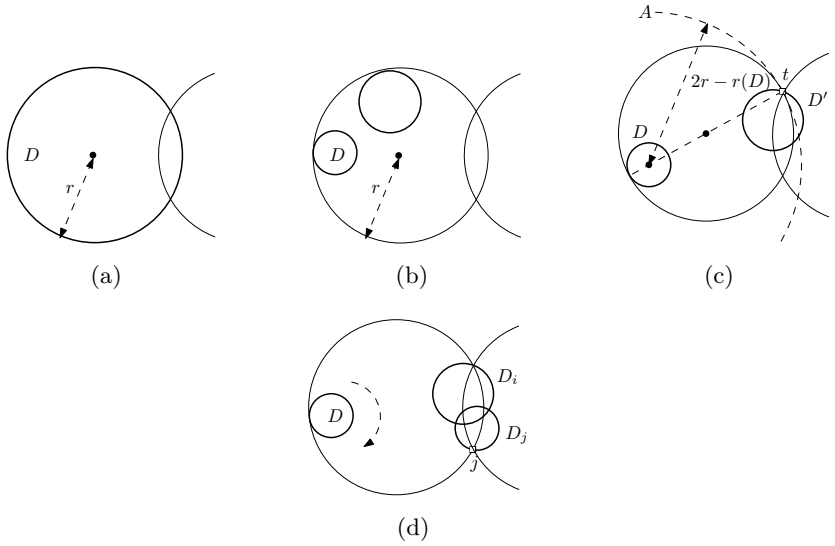


Figure 4.8: Four cases for $r \geqslant r^*$.

Decision Algorithm. Based on the cases discovered in Lemma 4.2.7, we do the following:
Case (a). Choose an input disk D. C_1 has radius r and covers only D. Then C_2 is the smallest disk containing $\mathcal{D} \setminus D$. If the radius of C_2 is $\leqslant r$, we return "yes".
Case (b). We simply choose a pair of input disks D and D'. There are two candidates

for C_1, as C_1 has radius r and touches D and D'. So we consider separately each of the two candidates for C_1. Then C_2 is chosen to be the smallest disk containing the input disks, or the portions of input disks (crescents), that are not covered by C_1, which can be computed in $\mathcal{O}(n)$ time. If for one of the two choices of C_1, the corresponding disk C_2 has radius $\leqslant r$, we return "yes".
Case (c). For each input disk D, we do the following.

1. For the circle A with center $c(D)$ and radius $2r - r(D)$, compute $A \cap D'$ for every other disk D'. Let t be such an intersection point.

2. For each t,

   (a) remove (part of) the input disks covered by the covering disk determined by D and t, and compute the smallest disk covering the remaining input.

   (b) If this algorithm returns a covering disk with radius $\leqslant r$, return "yes".

Case (d). For each input disk D that touches C_1 from inside, we do the following. Let i be the index of the first input disk that the circular arc of C_1 from the touching point hits in clockwise orientation. Let j be the index of the last input disk that the circular arc leaves. For each pair (i, j), we still have some freedom of rotating C_1 around D within some interval (C_2 changes accordingly.) During the rotation, an input disk not covered by the union of C_1 and C_2 may become fully covered by the union, or vice versa. We call such an event an *I/O event*. Note that an I/O event occurs only when an input disk touches C_1 or C_2 from inside (otherwise, it is another pair, (i', j) or (i, j').)

We compute all I/O events and sort them. At the beginning of the rotation of C_1 around D, we compute the number of input disks that are not fully covered, and set the variable *counter* to this number. Then we handle I/O events one by one and update the counter. If the counter becomes 0, we return "yes".

The running time of the algorithm is $\mathcal{O}(n^3 \log n)$; its analysis can be found in [9]. Hence, we get the following lemma.

Lemma 4.2.8 ([9, Theorem 8])**.** *Given a set $\mathcal{D}$ of n disks in the plane and a value $r > 0$, we can decide in $\mathcal{O}(n^3 \log n)$ time whether there exist two disks with radius r that cover all disks in $\mathcal{D}$.*

Optimization Algorithm. For the optimization algorithm we use parametric search. To use the parametric search technique, we design a parallel version of the decision algorithm with running time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3)$ processors. This parallel algorithm can be found in [9]. Then the overall algorithm runs in time $\mathcal{O}(n^3 \log^4 n)$.

Theorem 4.2.9. *Given a set $\mathcal{D}$ of n disks in the plane, we can compute two smallest congruent disks whose union covers all disks in $\mathcal{D}$ in $\mathcal{O}(n^3 \log^4 n)$ time.*

4.2.2.1 Constant Factor Approximation.

We apply again the well known greedy k-center approximation algorithm by Gonzalez [61] to our general covering problem. The idea is as follows: We start by picking an arbitrary point p_1 inside the union of all disks in $\mathcal{D}$. For instance, we could pick the center of D_1 to be p_1. Then we compute the point p_2 in $\bigcup \mathcal{D}$ that is farthest away from p_1. This can be done in linear time by computing the point farthest away from p_1 for each disk $D \in \mathcal{D}$. We take these two points p_1, p_2 as center points for our two covering disks and the radius is the maximal distance from any point in $\bigcup \mathcal{D}$ to its closest point in $\{p_1, p_2\}$. This distance can be computed in linear time: The bisector of p_1 and p_2 divides the points of $\bigcup \mathcal{D}$ into two sets. The points lying on the same side of the bisector as p_1 form one set $\mathcal{S}_1$, while the points that lie on the same side of the bisector as p_2 form the other set $\mathcal{S}_2$. The points lying on the bisector can be assigned to any set. In order to compute this maximal distance, we only have to compute the maximal distance from any point in $\mathcal{S}_1$ to p_1, and the maximal distance from any point in $\mathcal{S}_2$ to p_2.

This algorithm leads to a 2-approximation. This can be shown in the same way as it is done for the k-center problem [61]: Suppose q is the point in $\bigcup \mathcal{D}$ that has maximal distance to $\{p_1, p_2\}$. We denote this distance by η. The disks computed by the algorithm above have radius $\leqslant \eta$. Note that $|p_1 p_2| \geqslant \eta$. It is easy to see that the radius of the disks in an optimal solution is $\geqslant 1/2\eta$ and, hence, the algorithm computes a 2-approximation.

Theorem 4.2.10. *Given a set $\mathcal{D}$ of n disks in the plane, we can compute a 2-approximation for the general covering problem for $\mathcal{D}$ in $\mathcal{O}(n)$ time.*

4.2.2.2 $(1+\epsilon)$-Approximation.

We can improve the approximation factors above significantly. We now present an $(1+\epsilon)$-approximation algorithm: The algorithm is again an adaptation of the algorithm by Agarwal and Procopiuc [4] as in Section 4.2.1.2. We start by computing a 2-approximation for the general covering case in $\mathcal{O}(n)$ time using our algorithm from Theorem 4.2.10. Let $C_1^{\max}$, $C_2^{\max}$ be the disks computed by this approximation algorithm and let $r^{\max}$ be their radius. As in the proof of Theorem 4.2.5, we consider a grid of size $\delta = \lambda \epsilon r^{\max} \leqslant 2\lambda\epsilon r^*$ over the plane where λ is a small enough constant. Each center is moved to a nearby grid point, so center (x, y) is replaced by $(\delta\lceil x/\delta \rceil, \delta\lceil y/\delta \rceil)$. If two or more centers are moved to the same grid point, we only keep the disk with the largest radius. The centers are now grid points and these grid points lie either inside $C_1^{\max} \cup C_2^{\max}$ or have a distance at most $\sqrt{2}\delta$ from the boundary of this union, and so we are left with a set of $\mathcal{O}(1/\epsilon^2)$ disks. We now replace this set of disks by grid points: Each disk is replaced by the grid points which are closest to the boundary of the disk and lie inside the disk, see Figure 4.9.

In order to compute these points we consider each column and each row of the

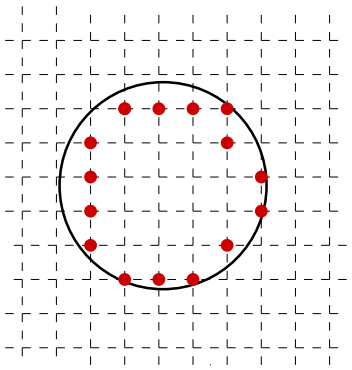


Figure 4.9: The disk is replaced by the marked points.

grid separately: The intersection of each disk with this column is an interval, and we replace the interval by the lowest and the highest grid point lying inside this interval. Since the set of disks has size $\mathcal{O}(1/\epsilon^2)$ and the number of columns is $\mathcal{O}(1/\epsilon)$, it takes in total $\mathcal{O}(1/\epsilon^3)$ time. We do the equivalent for each row of the grid. The set of grid points we obtain is denoted by P_g and its size is $\mathcal{O}(1/\epsilon^2)$. We compute two smallest disks E_1, E_2 that cover P_g in $\mathcal{O}(\frac{1}{\epsilon^2}\log^2\frac{1}{\epsilon}\log^2\log\frac{1}{\epsilon})$ time using the algorithm from Chan [34]. The radii of E_1, E_2 are at most $r^* + \sqrt{2}\delta$, and if these radii are increased by $2\sqrt{2}\delta$, these two disks cover $\mathcal{D}$. The radii of these inflated versions of E_1, E_2 is $r^* + 3\sqrt{2}\delta \leqslant (1+\epsilon)r^*$ if we choose $\lambda \leqslant 1/(6\sqrt{2})$.

Theorem 4.2.11. *Given a set $\mathcal{D}$ of n disks in the plane, a $(1+\epsilon)$-approximation for the general covering case for $\mathcal{D}$ can be computed in $\mathcal{O}(n + 1/\epsilon^3)$ time.*

4.3 Maximum Intersection Problem

In this section we consider an optimization version of the intersection problem presented in Section 4.1.

> Given a set $\mathcal{D}$ of n disks in the plane and a value $\delta \geqslant 0$, compute two disks with radius δ that intersect the maximum number of disks in $\mathcal{D}$.

De Berg et al. studied a simplified version of this problem where all disks have the same radii [39]. They presented an $\mathcal{O}(n^3 \log n)$ time algorithm and a $(1-\epsilon)$-approximation algorithm with running time $\mathcal{O}(n\log n + n\epsilon^{-6}\log(1/\epsilon))$. If the input is a set of unit disks and the pair of solution disks has to be disjoint, the problem can even be solved in $\mathcal{O}(n^{8/3}\log^2 n)$ time [32].

We study now the problem for a set of disks with different radii. We state two straightforward algorithms and later on we present an efficient algorithm that is based on the data structures used in the algorithm for unit disks by de Berg et al. [39].

We first formulate the problem in a different way, following the approach of Section 4.1. Recall the definition of δ-inflated disks from Section 4.1: For a real number $\delta \geqslant 0$ and a disk D, let the *δ-inflated* disk $D(\delta)$ be a disk with radius $r(D) + \delta$ concentric to D. Let $\mathcal{D}(\delta)$ be the set of all δ-inflated disks $D \in \mathcal{D}$. We can now state the problem in the following way:

> Given a value $\delta \geqslant 0$, compute two points p_1, p_2 such that they stab the maximum number of disks in $\mathcal{D}(\delta)$ (that is $|\{D(\delta) \cap \{p_1, p_2\} \mid D \in \mathcal{D}\}|$ is as large as possible).

Clearly, there exist two points p_1, p_2 that stab k disks of $\mathcal{D}(\delta)$ if and only is there exist two disks centered at p_1 and p_2 with radius δ that intersect k disks of $\mathcal{D}$. Hence, if we compute a pair of points p_1, p_2 that stabs the maximum number of disks in $\mathcal{D}(\delta)$, the disks centered at p_1, p_2 with radius δ intersect the maximum number of disks in $\mathcal{D}$.

A straightforward way to solve this problem is the following: Compute the arrangement of the disks $\mathcal{D}(\delta)$, this can be done in $\mathcal{O}(n^2)$ time and this arrangement consists of $\mathcal{O}(n^2)$ cells [21]. Then we take any possible pair of two cells, place a point in each of the cells and compute the number of disks in $\mathcal{D}(\delta)$ that are stabbed by these points. Since there are $\mathcal{O}(n^2)$ cells, there are $\mathcal{O}(n^4)$ such pairs. For each pair, the maximum number of disks stabbed by the corresponding points can be computed in $\mathcal{O}(n)$ time. So in total we need $\mathcal{O}(n^5)$ time.

3SUM-Hardness. The problem stated above is 3SUM-hard. *3SUM-hardness* was introduced by Gajentaan and Overmars [59]. 3SUM is the following problem: Given a set of integers S, are there three integers $a, b, c \in S$ with $a + b + c = 0$. This problem can be solved in $\Theta(n^2)$ time but no subquadratic-time algorithm is known. A problem is 3SUM-hard if it is at least as difficult as 3SUM, meaning a subquadratic-time algorithm for this problem gives a subquadratic-time algorithm for 3SUM. More formally, a problem P is 3SUM-hard if and only if every instance of 3SUM of size n can be solved using a constant number of instances of P of at most linear size and $o(n^2)$ additional time (see also [59, Lemma 2.1]). Proving a problem to be 3SUM-hard does not prove any lower bound for this problem, however, it implies that there is not so much hope for a subquadratic-time algorithm.

We will present now the 3SUM-hardness proof for our problem. The proof is very simple. It follows directly from the fact that the problem of computing the *deepest cell* in an arrangement of disks is 3SUM-hard [24] (the *depth* of a cell is defined as the number of disks that contain this cell). Thus, computing a point that stabs the maximum number of disks is 3SUM-hard. This problem reduces to our problem in the following way: Duplicate the given set of disks and put it somewhere such that the original set and the duplicated set do not intersect. Computing two points that stab the maximum number of disks in this new set corresponds to computing one point that stabs the maximum number of disks in the original set.

We present now an efficient algorithm to solve the problem stated above. We first state a naïve approach, later on we show how this approach can be improved.

Naïve Approach. The basic idea is quite similar to the idea of the algorithm in Section 4.1.1. We are given a set of disks $\{D_1, D_2, ..., D_n\}$ in the plane and a value $\delta \geqslant 0$. We first construct the arrangement of the δ-inflated disks $D_i(\delta), i = 1, 2, .., n$. This arrangement consists of $\mathcal{O}(n^2)$ cells and can be computed in $\mathcal{O}(n^2)$ time [21]. The idea is to traverse the cells of the arrangement in a way similar to Section 4.1.1. To this end, we traverse the cells in a depth-first manner. While traversing the cells we do the following: We place one point, say p_1, in the current cell, then we compute the point that stabs the maximum number of disks that are not stabbed by p_1. We move p_1 to the next cell and repeat this until we have visit every cell, then we return the pair of points that stabs the maximum number of disks. This idea leads to an algorithm with running time $\mathcal{O}(n^4)$: There are $\mathcal{O}(n^2)$ cells and computing the maximum depth of a set of n disks can be done in $\mathcal{O}(n^2)$ time [31].

Faster Approach. We show now that the running time of the naïve algorithm can be improved by almost a linear factor. We use a similar approach as in [39]. Again, we start with a set of disks $\{D_1, D_2, ..., D_n\}$ in the plane and a value $\delta \geqslant 0$ and we compute the corresponding arrangement of the δ-inflated disks. The set of these inflated disks is then denoted by $\mathcal{D}$. Before we explain how the arrangement is traversed, we observe an interesting fact: We do not have to consider all cells in the arrangement, it is enough to consider only the maximal cells (this idea can also be found in [77]). A maximal cell has the property that no other cell is defined by a superset of the disks defining this cell. Hence, if there exist two points that together stab k disks of the arrangement, there exist two points that both lie in maximal cells and together stab at least k disks. In order to determine the set of maximal cells we use a important property of maximal cells: A maximal cell is *convex* (and all convex cells are maximal). Note that the number of convex cells can be $\Omega(n^2)$ (see Figure 4.10).

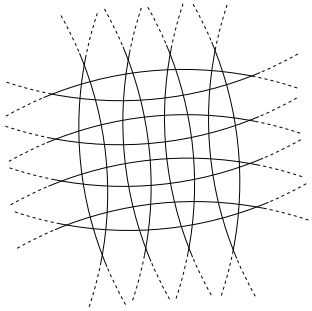


Figure 4.10: The disks build a grid and all cells with depth 4 are maximal. Clearly there are $\Omega(n^2)$ maximal cells.

Now we explain how the arrangement is traversed. The crucial part is that we do not consider all cells, instead we consider a superset of all convex cells. Our traversal is defined as follows: We traverse the boundary of each disk $D_i(\delta)$ $(i = 1, 2, \ldots, n)$; that means we traverse all cells that are incident to the boundary of $D_i(\delta)$ and lie in $D_i(\delta)$, see Figure 4.11. While traversing the boundary of this disk, we visit at most $\mathcal{O}(n)$ cells; note that during the traversal of the boundary of a disk a cell can be visited more than once, see Figure 4.12. We count each visit individually; the number of visited cells per disk is still $\mathcal{O}(n)$.

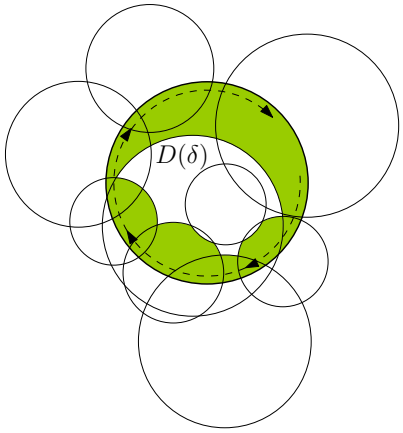


Figure 4.11: The traversal traverses the marked cells for the disk $D(\delta)$.

We start with traversing the boundary of $D_1(\delta)$, then the boundary of $D_2(\delta)$ and so on. The path traversing the boundary of $D_i(\delta)$ is denoted by γ_i. Hence, our traversal consists of n connected paths, namely γ_i for all $i = 1, 2, \ldots, n$. We denote the concatenation of all paths by $\gamma = \gamma_1\gamma_2 \ldots \gamma_n$.

We traverse now the arrangement by γ, hence we traverse a set of cells that contains all convex cells as a subset. Also, we visit cells more than once. However, per disk we visit at most $\mathcal{O}(n)$ cells and, in total, we visit $\mathcal{O}(n^2)$ cells.

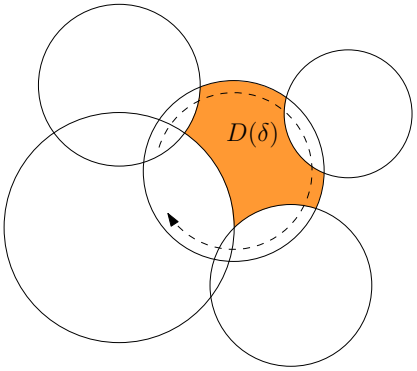


Figure 4.12: The orange cell is visited more than once during the traversal of $D(\delta)$.

We place one point, say p_1, in a cell and we will move p_1 along γ and traverse the arrangement. Let $\mathcal{D}_{p_1}$ be the set of disks that are stabbed by p_1. Then, while we move p_1 along γ, we want to compute the maximum subset of disks in $\mathcal{D} \setminus \mathcal{D}_{p_1}$ that have a nonempty intersection. To do this efficiently, we use a segment tree. Each disk of $\mathcal{D}$ may appear or disappear several times during the traversal of γ: Each time we cross the boundary of a cell, one disk is inserted or deleted from $\mathcal{D}_{p_1}$. So each disk appears in $\mathcal{D}_{p_1}$ along one or several segments of the traversal. We store these segments in a segment tree. Consider a connected part of γ, say γ_1. The first cell, which is traversed by γ_1, consists of $\mathcal{O}(n)$ disks because each cell is formed by the intersection of $\mathcal{O}(n)$ disks. During the traversal of the boundary of $D_1(\delta)$ each disk $D' \in \mathcal{D} \setminus \{D_1(\delta)\}$ is inserted and deleted at most once from $\mathcal{D}_{p_1}$, since the boundaries of two disks intersect at most twice. Hence, each disk is represented by $\mathcal{O}(n)$ segments and so the segment tree is built over a total of $\mathcal{O}(n^2)$ segments. Thus the tree has total size of $\mathcal{O}(n^2 \log n)$: Each segment is stored at $\mathcal{O}(\log n)$ nodes of the tree. Additionally, we store the following two values at each node u (the idea of storing these values, and also the notation, is taken from [39]):

- We store the total number of segments stored at the node u. We denote the value by **Cover**(u).
- Each node stores a value **maxDepth**(u). If u is a leaf then maxDepth(u) = Cover(u). Else maxDepth(u) is the sum of Cover(u) and the maximum value of maxDepth of the two children of u. Hence,

  $$\text{maxDepth}(u) = \text{Cover}(u) + \max(\text{maxDepth}(c_1), \text{maxDepth}(c_2)),$$

  where c_1 and c_2 are the two children of u.

Notice that maxDepth($root$) stores the value of the deepest cell of the arrangement where $root$ is the root of the tree and, hence, it gives the maximum number of disks that can be stabbed by a point. The idea is now to update the segment tree in such a way that it represents the arrangement of disks in $\mathcal{D} \setminus \mathcal{D}_{p_1}$. So in order to find the maximum number of disks in $\mathcal{D} \setminus \mathcal{D}_{p_1}$ that have a nonempty intersection, we return maxDepth($root$). During the traversal disks are removed or added to $\mathcal{D}_{p_1}$. A disk can be removed or added to the segment tree in $\mathcal{O}(n \log n)$ time, see [39]. For completeness we still state the ideas here:
Each disk contributes to $\mathcal{O}(n)$ segments in total. Each segment is stored in $\mathcal{O}(\log n)$ nodes. Assume that a disk D is added to $\mathcal{D}_{p_1}$. Then we have to update the values Cover(v) and maxDepth(v) for the nodes that store D and their ancestors. First we have to find the set of nodes N_D that store disk D. This set consists of $\mathcal{O}(n \log n)$ nodes and can be found in the same time. The values of the nodes in N_D can be updated in $\mathcal{O}(1)$ time, we only have to decrease Cover(u) and maxDepth(u) by 1 for each node $u \in N_D$. But we also have to update the ancestors of the nodes in N_D. Since maxDepth(u)=Cover(u)+max(maxDepth(c_1),maxDepth(c_2)), the values

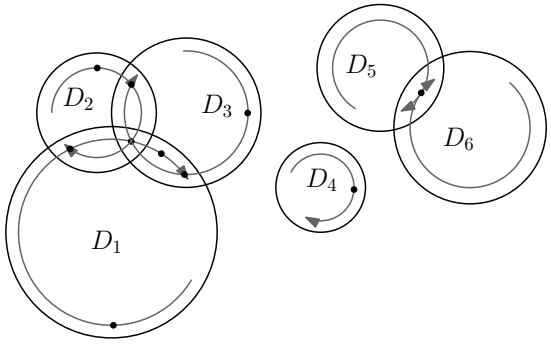


Figure 4.13: The traversal of the arrangement. For brevity, we denote the δ-inflated disks $D_i(\delta)$ by D_i.

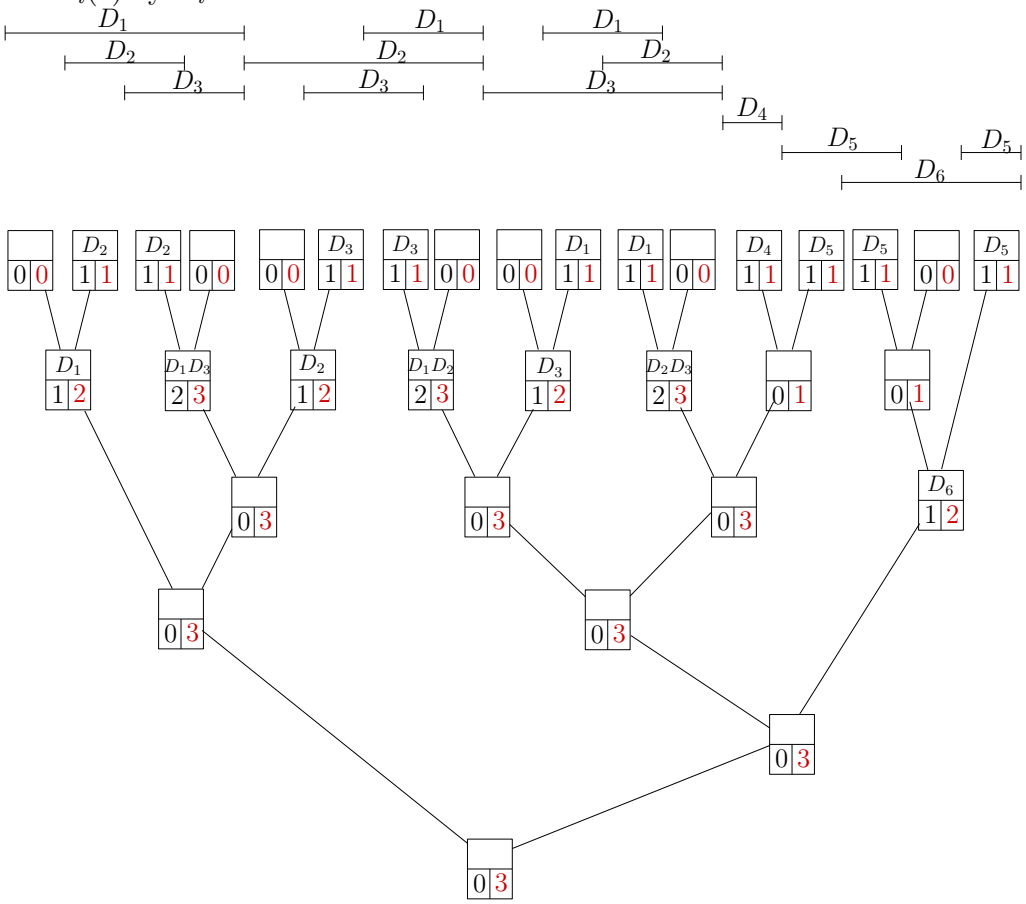


Figure 4.14: The segment tree is depicted. The value Cover(u) is the left value and maxDepth(u) the right value.

of the ancestors can be updated in a bottom-up fashion in $\mathcal{O}(1)$ time: Consider only one segment representing D. This segment is stored in $\mathcal{O}(\log n)$ nodes and at most two of these nodes have the same depth. All nodes and their ancestors can be

updated in $\mathcal{O}(\log n)$ time in total. This is based on the properties of a segment tree. Thus, we need $\mathcal{O}(n \log n)$ time to update the tree when a disk is removed or added to $\mathcal{D}_{p_1}$.
While traversing γ, a disk appears or disappears in $\mathcal{D}_{p_1}$ and hence we have to update the tree. The path γ traverses $\mathcal{O}(n^2)$ cells. For each connected part γ_i of γ, these cells differ in exactly one disk and, hence, the updating of the tree takes $\mathcal{O}(n \log n)$ time per cell. The last cell of γ_i and the first cell of γ_{i+1} differ in $\mathcal{O}(n)$ cells and so the updating of the tree takes $\mathcal{O}(n^2 \log n)$ time. But since γ consists of n connected paths, in total, traversing γ and all updates take $\mathcal{O}(n^3 \log n)$ time.

At the end, the algorithm returns the maximum number of disks that can be stabbed by two points. Notice that the algorithm can additionally return a pair of solution points.

Thus, we have showed the following

Theorem 4.3.1. *Given a set $\mathcal{D}$ of n disks in the plane and a value $\delta \geqslant 0$, we can compute two disks with radius δ whose union intersect the maximum number of disks in $\mathcal{D}$ in $\mathcal{O}(n^3 \log n)$ time.*

4.3.1 Constant Factor Approximation

We apply the well-known greedy approximation algorithm for the maximum coverage problem [69, Section 3.9]. The maximum coverage problem is formulated as follows: Given a collection of sets and a natural number k, find k sets such that the total number of elements covered by these sets is maximized (note that sets may contain the same elements). The greedy approximation algorithm is very simple: Add the set with the maximum number of elements into the solution set, in the next step add the set with the maximum number of elements that are not contained in the solution sets and so on. This algorithms yields an approximation factor of $(1 - (1 - 1/k)^k) > (1 - 1/e)$ [69, Theorem 3.8].

We can adapt this idea to our problem. First we compute the disk C_1 with radius δ that intersects the maximum number of disks in $\mathcal{D}$. In the second step we compute the disk C_2 with radius δ that intersect the maximum number of disks in $\mathcal{D}$ that are not already intersect by C_1. This algorithm leads to an approximation factor of $(1-(1-1/2)^2) = 3/4$. This follows immediately from the proof of the approximation factor of the maximum coverage problem, see [69, Theorem 3.8].

We still need to analyze the running time. As we explained above, we consider the set of the δ-inflated disks of $\mathcal{D}$. The deepest cell of this arrangement of disks can be computed in $\mathcal{O}(n^2)$ time [31]. Let p_1 be a point inside this cell. Then, all δ-inflated disks that contain p_1 are deleted and the deepest cell in this new arrangement is computed; let p_2 be a point inside this cell. The two disks with radius δ and centers p_1 and p_2, respectively, give a $3/4$-approximation for the maximum intersection problem and the total running time is $\mathcal{O}(n^2)$.

Theorem 4.3.2. *Let $\mathcal{D}$ be a set of n disks in the plane and $\delta \geqslant 0$ a value. We can compute a $3/4$-approximation for the maximum intersection problem for $\mathcal{D}$ in $\mathcal{O}(n^2)$ time.*

The running time can be improved by almost a linear factor at the expense of slightly decreasing the approximation factor. We use the same algorithm as in Theorem 4.3.2, but instead of computing the deepest cell exactly we approximate it within a factor of $(1-\epsilon)$ by using the algorithm of Aronov and Har-Peled [24]. Their algorithm approximates the deepest cell of a set of n disks by a factor of $(1-\epsilon)$ in $\mathcal{O}(n\epsilon^{-2}\log n)$ expected running time.

We show that our algorithm has an approximation factor of $(3/4-\epsilon)$. The quality of the approximation factor can be shown in the same way as for the maximum coverage problem [69, Theorem 3.8]. For completeness we state the proof here. Let C_1 be the disk that the algorithm computes in the first step, and let C_2 be the disk that is computed in the second step. Let $nb(C_1)$ and $nb(C_2)$ be the number of disks that are intersected by C_1 and C_2. Let $nb(\mathrm{OPT})$ be the number of disks intersected by the optimal solution. The disk that intersects the maximum number of disks in $\mathcal{D}$ intersects at least $\frac{nb(\mathrm{OPT})}{2}$ disks and so $nb(C_1) \geqslant \frac{(1-\epsilon)nb(\mathrm{OPT})}{2}$. Now consider the set $\mathcal{D}' = \mathcal{D}\backslash\{D \mid D \in \mathcal{D} \text{ and } D \cap C_1 \neq \emptyset\}$. The disk that intersects the maximum number of disks in $\mathcal{D}'$ intersects at least $\frac{nb(\mathrm{OPT})-nb(C_1)}{2}$, so $nb(C_2) \geqslant \frac{(1-\epsilon)(nb(\mathrm{OPT})-nb(C_1))}{2}$. Then,

$$\begin{aligned}
nb(C_1)+nb(C_2) &\geqslant nb(C_1)+\frac{(1-\epsilon)(nb(\mathrm{OPT})-nb(C_1))}{2}\\
&= nb(C_1)(1-\frac{(1-\epsilon)}{2})+\frac{(1-\epsilon)nb(\mathrm{OPT})}{2}\\
&\geqslant \frac{(1-\epsilon)nb(OPT)}{2}(1-\frac{(1-\epsilon)}{2})+\frac{(1-\epsilon)nb(\mathrm{OPT})}{2}\\
&= ((1-\epsilon)-\frac{(1-\epsilon)^2}{4})nb(\mathrm{OPT})\\
&= (\frac{3}{4}-\frac{\epsilon}{2}-\frac{\epsilon^2}{4})nb(\mathrm{OPT})\\
&\geqslant (\frac{3}{4}-\epsilon)nb(\mathrm{OPT}) \text{ for } \epsilon \leqslant 1
\end{aligned}$$

Theorem 4.3.3. *Let $\mathcal{D}$ be a set of n disks in the plane and $\delta \geqslant 0$ a value. We can compute a $(3/4-\epsilon)$-approximation for the maximum intersection problem for $\mathcal{D}$ in $\mathcal{O}(n\epsilon^{-2}\log n)$ time, for $0 < \epsilon \leqslant 1$.*

4.4 Maximum Covering Problem

We consider the following problem.

> Given a set $\mathcal{D}$ of n disks in the plane and a value $\delta \geqslant 0$, compute two disks C_1, C_2 with radius r that together cover the maximum number of

> disks in $\mathcal{D}$. (A disk is covered by C_1 and C_2 if it is completely contained in C_1 or C_2.)

Again, we follow the approach in Section 4.2.1 and formulated the problem in a different way. Repeat the definition of $D^*(\delta)$: $D^*(\delta)$ is the disk concentric to D with radius $\delta - r(D)$; if $\delta - r(D) < 0$, the disk is not defined. Notice that a disk with radius larger than δ can never be fully covered by a disk with radius δ. Let $\mathcal{D}^* = \{D^*(\delta) \mid D \in \mathcal{D} \text{ and } r(D) \leqslant \delta\}$.

> Given a value δ, compute two points, p_1 and p_2, such that these points are contained in the maximum number of disks in $\mathcal{D}^*(\delta)$.

This problem can be solved with the same algorithm as in Theorem 4.3.1. We only have to add an initialization step in which each disk $D \in \mathcal{D}$ is replaced by a disk concentric to D with radius $\delta - r(D)$. If $\delta - r(D) < 0$, the disk D is deleted. After this initialization step the same algorithm as in Section 4.3 can be used.

Theorem 4.4.1. *Let $\mathcal{D}$ be a set of n disks in the plane and $r \geqslant 0$ a value. We can compute two disks with radius r that cover the maximum number of disks in $\mathcal{D}$ in $\mathcal{O}(n^3 \log n)$ time.*

4.4.1 Constant Factor Approximation

By the same argumentation as above, we can use the approximation algorithms of Section 4.3.1. We only have to add the same initialization step where each disk $D \in \mathcal{D}$ is replaced by $D^*(\delta)$. Thus, we achieve immediately the following results.

Theorem 4.4.2. *Let $\mathcal{D}$ be a set of n disks in the plane and $\delta \geqslant 0$ a value. We can compute a*

- *3/4-approximation for the maximum covering problem for $\mathcal{D}$ in $\mathcal{O}(n^2)$ time.*
- *$(3/4-\epsilon)$-approximation for the maximum coverage problem for $\mathcal{D}$ in $\mathcal{O}(n\epsilon^{-2} \log n)$ time.*

4.5 Conclusions and Open Problems

In this chapter we considered new versions of the two-center problem where the input is a set of disks which we called the intersection and the covering problem. We also studied optimization versions of these problems. We gave exact and approximation algorithms; all our results are summarized in the following table.

	Exact algorithm	$(1+\epsilon)$-approximation
Intersection problem	$\mathcal{O}(n^2 \log^4 n \log\log n)$ $\mathcal{O}(n^2 \log^3 n)$ expected time	–
General covering problem	$\mathcal{O}(n^3 \log^4 n)$	$\mathcal{O}(n + 1/\epsilon^3)$
Restricted covering problem	$\mathcal{O}(n^2 \log^4 n \log\log n)$ $\mathcal{O}(n^2 \log^3 n)$ expected time	$\mathcal{O}(n + (1/\epsilon^3) \log 1/\epsilon)$
	Exact algorithm	$(3/4-\epsilon)$-approximation
Max intersection problem	$\mathcal{O}(n^3 \log n)$	$\mathcal{O}(n\epsilon^{-2} \log n)$
Max covering problem	$\mathcal{O}(n^3 \log n)$	$\mathcal{O}(n\epsilon^{-2} \log n)$

There are still a lot of open problems. Our algorithms for the intersection and the restricted covering problem are superquadratic. Since the best-known algorithm for computing the two-center problem of a point set is near-linear [34, 53, 106] an obvious question is whether our problems can be solved in subquadratic time. These results for the subquadratic algorithms for the two-center problem are based on observations and data structures that only work for unit disks, we do not see how these ideas could help to improve our time bounds.

The decision versions of the intersection problem and the restricted covering problem are equivalent to deciding whether a set of disks is 2-piercable. We could neither find any algorithm that solves the latter in subquadratic time nor prove it to be 3SUM-hard. If someone could prove 3SUM-hardness, this would imply that there is not much hope for a subquadratic algorithm for the intersection problem and the restricted covering case. It would also explain why we could not find any efficient approximation algorithm for the intersection problem. Any constant-factor approximation algorithm has to decide whether two disks of size 0 intersect the input set and so it answers the 2-piercing problem for disks.

Another interesting problem is to improve the exact algorithm for the general covering case.

Chapter 5

Largest Inscribed Rectangle

This chapter deals with the problem of substituting a complex geometric object by a simpler one that lies completely inside this object while still maintaining a certain sense of similarity. One famous problem of this type is to compute the largest rectangle inscribed in a polygon. Alt et al. [17] presented an $\mathcal{O}(\log n)$ algorithm which computes the largest inscribed *axis-aligned* rectangle in a convex polygon with n vertices under the restriction that the vertices are given in counterclockwise direction, stored in an array. Daniels et al. [37] showed that the largest inscribed axis-aligned rectangle in a *general* polygon (allowing holes) with n vertices can be computed in $\mathcal{O}(n \log^2 n)$ time. Ahn et al. [7] described how to approximate axially symmetric polygons inside the more general class of convex sets, however the computed polygon is not necessarily a rectangle. Hall-Holt et al. [66] restrict the problem to *fat* rectangles, i. e., rectangles with an aspect ratio that is bounded by a constant. Under the assumption that a largest inscribed rectangle is fat, they $(1-\epsilon)$-approximate the largest fat rectangle in simple polygons in time $\mathcal{O}(n)$; in polygons with holes, their approximation algorithm runs in $\mathcal{O}(n \log n)$ time.

In Section 5.1 we consider the problem of computing the largest area inscribed rectangle in a convex polygon P on n vertices. To the best of our knowledge, there is no exact algorithm published so far, but there is a straightforward way in $\mathcal{O}(n^4)$ time. We first present this straightforward algorithm and then we focus on approximation algorithms. We present approximation algorithms with running times that are only logarithmically dependent on n, if the vertices are given in counterclockwise direction, stored in an array. The assumption on the vertex ordering is common when handling polygons. (As mentioned before, Alt et. al [17] used the same assumption for their algorithm.) If the ordering is not given in advance, it can be computed using standard convex hull algorithms in $\mathcal{O}(n \log n)$ time. We will assume in the remainder of this chapter that the ordering is given. In comparison to Hall et al. [66], our results show that fatness is not required for approximating a largest inscribed rectangle.

In Section 5.2 we show that our approximation algorithms can also be generalized to simple polygons with or without holes.

The results of Section 5.1.2 and Section 5.2 were obtained in collaboration with Christian Knauer, Jens M. Schmidt, and Hans Raj Tiwary and have been published in [82].

Notation. The area of a polygon P is denoted by $|P|$. A line segment connecting two points a and b is denoted by $\overline{ab}$ and its length by $|\overline{ab}|$. For a given convex polygon P, let R_{opt} be a largest area inscribed rectangle. Note that in general the largest area inscribed rectangle is not unique, see Figure 5.1; we will use R_{opt} to denote any one of the largest area inscribed rectangles.

For brevity, a largest area rectangle will be referred to as largest rectangle.

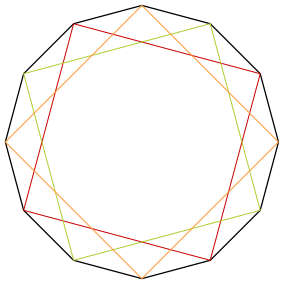


Figure 5.1: The largest inscribed rectangle is not unique.

5.1 Largest Inscribed Rectangle in a Convex Polygon

We consider the following problem:

> Given a convex polygon P with n vertices given in counterclockwise order, compute a largest area inscribed rectangle in P.

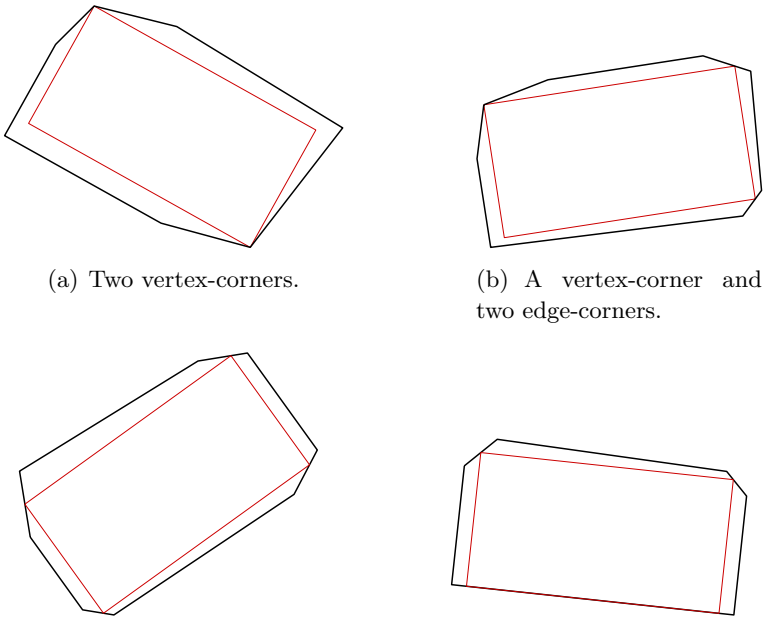
We first give a slow but exact algorithm that computes an optimal solution. Later on we present efficient approximation algorithms.

5.1.1 A Slow but Exact Algorithm

To our knowledge, there is no exact algorithm published for computing a largest inscribed rectangle in a convex polygon. But there is a straightforward way to compute such a rectangle in $\mathcal{O}(n^4)$ time. We will give such a straightforward algorithm in this section.

We first investigate the characterization of a largest rectangle. We start with introducing some new notation: If a corner of R_{opt} is coincident with a vertex of

P we call it a *vertex-corner*; if it is contained in an edge of the boundary of P we call it an *edge-corner*. (Notice that a vertex-corner is also an edge-corner.) The largest *axis-aligned* rectangle in a convex polygon either has only two diagonally opposite edge-corners or at least three edge-corners [17]. We will prove a similar characterization for R_{opt}, namely R_{opt} has either (a) two diagonally opposite vertex-corners, (b) a vertex-corner and two edge-corners, (c) four edge-corners, or (d) two edge-corners and a side of R_{opt} is aligned to a boundary edge of P, see Figure 5.2. (Recall that a vertex-corner is also an edge-corner and, hence, cases like a rectangle has four vertex-corners are also included in this characterization.)



(a) Two vertex-corners.

(b) A vertex-corner and two edge-corners.

(c) Four edge-corners.

(d) Two edge-corners and a side of R_{opt} is aligned to a boundary edge of P.

Figure 5.2: The four different cases for R_{opt}.

This implies that R_{opt} is either aligned to an edge of P or it is determined by four edges (since a vertex-corner is determined by two edges). Although this characterization is quite intuitive, it seems that there is no formal proof published. Therefore, we state a proof for this characterization in the following.

We start with showing that if R_{opt} contains only two vertex-corners, they have to be diagonally opposite and R_{opt} has to be a square.

Lemma 5.1.1. *If R_{opt} has only two vertex-corners (and no other edge-corner) then they are diagonally opposite and R_{opt} is a square.*

Proof. The proof is by contradiction. If R_{opt} has only two vertex-corners that are

adjacent, then R_{opt} can trivially be increased.

Thus, we consider the case where R_{opt} has two diagonally opposite vertex-corners and R_{opt} is not a square, see Figure 5.3. Let the vertices of R_{opt} be a, b, c, and d

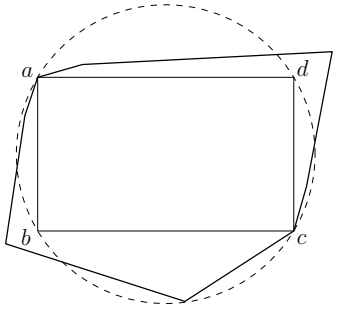


Figure 5.3: This rectangle is not a largest one.

in counterclockwise order, and let a and c be the vertex-corners. The vertices b and d do not touch the boundary of P. Consider the circle C that is determined by a and c. Now we move b on C in counterclockwise direction, d moves counterclockwise on C while keeping b and d antipodal (a and c stay fixed). Then the area of the rectangle defined by a, b, c, d is a concave function which obtains is maximum when a, b, c, and d form a square. Either this square is contained in P and we are done. Otherwise the largest rectangle with opposite vertices a and c is obtained when b or d are on the boundary of P and, hence, the rectangle has at least two vertex-corners and an edge-corner. Thus, R_{opt} was not a largest rectangle. □

The following lemma shows that an inscribed rectangle that has only 3 edge-corners (none of them being a vertex-corner) can never be a largest one.

Lemma 5.1.2. *An inscribed rectangle with* 3 *edge-corners is not an optimal one.*

Proof. The proof will be by contradiction. Assume that R_{opt} has exactly 3 edge-corners (none of them being a vertex-corner). Without loss of generality, let R_{opt} be axis-aligned and let the free corner (the corner that is not an edge-corner) be the upper right corner. Let the lengths of the sides be a and b. Let α be the angle between the upper edge of the rectangle and the edge of P on which the upper left corner lies. Let β be the angle between the right edge of the rectangle and the edge of P on which the lower right corner lies. (See Figure 5.4 left.)

We fix the lower left corner of the rectangle and move its adjacent corners along the edges of P, while maintaining a rectangle. We will show that the area of this rectangle increases by moving these corners in counterclockwise or in clockwise direction; hence the rectangle can be increased.

Consider Figure 5.4: We first notice that $0 < \alpha, \beta < \pi/2$. Let x_1, y_1 and x_2, y_2 be the lengths of the sides of the orange and the green rectangle, respectively. We would like to show that either $x_1 y_1 \geqslant ab$ or $x_2 y_2 \geqslant ab$.

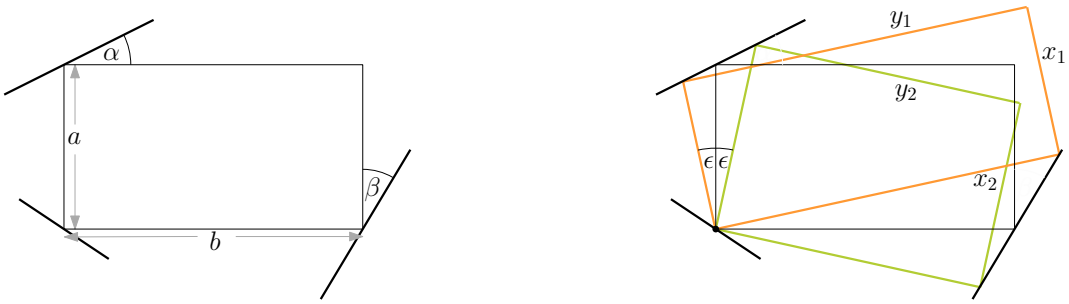


Figure 5.4: From left to right: The rectangle has 3 edge-corners. On the right: Either the area of the orange or the green rectangle is larger than the area of the black one.

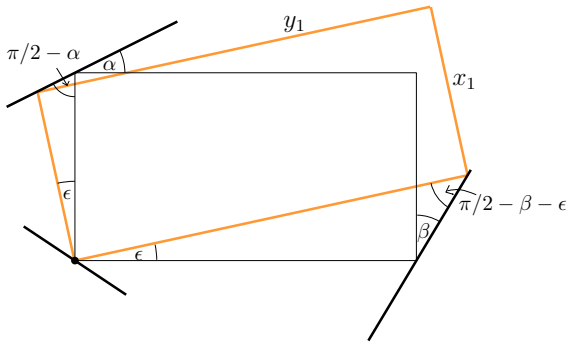


Figure 5.5: The area of the orange rectangle depends on a, b, α, β, and ϵ.

By using elementary trigonometry, we have that (see also Figure 5.5)

$$\begin{aligned} x_1 &= \frac{a\cos(\alpha)}{\cos(\alpha-\epsilon)} \\ y_1 &= \frac{b\cos(\beta)}{\cos(\beta+\epsilon)} \end{aligned}$$

Similarly, for the green rectangle we have:

$$\begin{aligned} x_2 &= \frac{a\cos(\alpha)}{\cos(\alpha+\epsilon)} \\ y_2 &= \frac{b\cos(\beta)}{\cos(\beta+\epsilon)} \end{aligned}$$

and so

$$\begin{aligned} x_1y_1 &= \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha-\epsilon)\cos(\beta+\epsilon)} \\ &= \frac{ab\cos(\alpha)\cos(\beta)}{\cos^2\epsilon\cos(\alpha)\cos(\beta)+\sin(\epsilon)\cos(\epsilon)\big[\sin(\alpha)\cos(\beta)-\sin(\beta)\cos(\alpha)\big]-\sin^2(\epsilon)\sin(\alpha)\sin(\beta)} \\ &= \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha)\cos(\beta)-\sin^2(\epsilon)\big[\cos(\alpha)\cos(\beta)+\sin(\alpha)\sin(\beta)\big]+\sin(\epsilon)\cos(\epsilon)\sin(\alpha-\beta)} \\ &= \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha)\cos(\beta)-\sin^2(\epsilon)\cos(\alpha-\beta)+\sin(\epsilon)\cos(\epsilon)\sin(\alpha-\beta)} \end{aligned}$$

Similarly,

$$\begin{aligned} x_2y_2 &= \frac{a\cos(\alpha)b\cos(\beta)}{\cos(\alpha+\epsilon)\cos(\beta-\epsilon)} \\ &= \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha)\cos(\beta)-\sin^2(\epsilon)\cos(\alpha-\beta)+\sin(\epsilon)\cos(\epsilon)\sin(\beta-\alpha)} \end{aligned}$$

(Notice that we choose ϵ sufficiently small such that $\cos(\alpha+\epsilon)$, $\cos(\alpha-\epsilon)$, $\cos(\beta+\epsilon)$, $\cos(\beta+\epsilon)$ are all larger than zero.)

If $\beta \geqslant \alpha$, then since $\cos(\alpha-\beta) \geqslant 0$ for $0 < \alpha, \beta < \pi/2$, it holds that

$$x_1y_1 = \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha)\cos(\beta)\underbrace{-\sin^2(\epsilon)\cos(\alpha-\beta)}_{\leqslant 0}+\underbrace{\sin(\epsilon)\cos(\epsilon)\sin(\alpha-\beta)}_{\leqslant 0}} \geqslant ab$$

If $\alpha \geqslant \beta$ it holds that

$$x_2y_2 = \frac{ab\cos(\alpha)\cos(\beta)}{\cos(\alpha)\cos(\beta)\underbrace{-\sin^2(\epsilon)\cos(\alpha-\beta)}_{\leqslant 0}+\underbrace{\sin(\epsilon)\cos(\epsilon)\sin(\beta-\alpha)}_{\leqslant 0}} \geqslant ab$$

This proves that a rectangle with 3 edge-corners is never a largest one. A rectangle with a larger area can always be constructed by fixing the lower left corner and moving the adjacent corners along the edges of P, either in clockwise or counterclockwise direction. The area of this rectangle increases until one of the following happens: The rectangle is either (i) aligned to a boundary edge of P, (ii) has a vertex-corner and two edge-corners, or (iii) has four edge-corners. This concludes the proof. □

It is easy to see that R_{opt} has not only one vertex-corner (and no edge-corners); it is also trivial to see that R_{opt} has more than two edge-corners or a vertex-corner and an edge-corner. Lemma 5.1.1 shows that if R_{opt} has only two vertex-corners then R_{opt} is a square. From the proof of Lemma 5.1.2 follows that R_{opt} is either

aligned to a boundary edge of P, has a vertex-corner and two edge-corners, or has four edge-corners. Thus, we can conclude that a largest inscribed rectangle either (a) has two opposite vertex-corners (and, thus, is a square), (b) has a vertex-corner and two edge-corners, (c) four edges-corners, or (d) a side is aligned to an edge of the polygon (Figure 5.2). And so our characterization for a largest rectangle is proven.

Algorithm. Our algorithm proceeds in the following way: Given is a convex polygon P with n vertices, first we compute the largest square in P; this takes $\mathcal{O}(n^2)$ time [42]. Then we compute, for each edge of the polygon, the largest inscribed rectangle aligned to this edge. In total this takes $\mathcal{O}(n \log n)$ time: The largest inscribed rectangle aligned to a given direction can be computed in $\mathcal{O}(\log n)$ time [17] and the polygon has n edges.

It remains to compute the largest inscribed rectangle that has either four edge-corners or a vertex-corner and two edge-corners. To keep the algorithm simple, we handle these cases together. Let e_1, e_2, e_3 be the boundary edges of P containing the edge-corners. (For simplification, we consider the vertex-corner as an edge-corner). If the fourth corner is also an edge-corner, let e_4 be the edge of P containing it; otherwise, let e_4 be a boundary edge that is intersected by extending the longer side of R_{opt}.

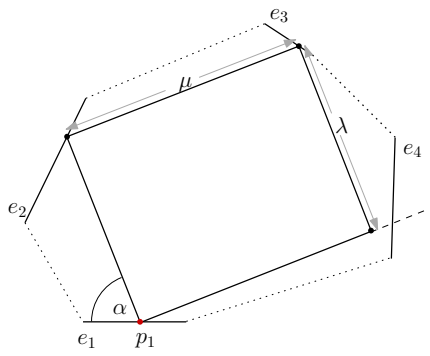


Figure 5.6: If the position of one corner of the rectangle and the orientation is known, the rectangle is determined.

Two parameters are necessary to determine the rectangle: The position of a corner of the rectangle on one of the edges, say e_1, and the angle α between e_1 and the closest side of R_{opt} in clockwise direction. Without loss of generality we assume that e_1 is parallel to the x-axis. The remaining corners are then given by

the following equations (see Figure 5.6):

$$\begin{aligned} p_2 &= p_1 + \lambda \frac{\vec{v}}{||\vec{v}||} \in e_2 \\ p_3 &= p_2 + \mu \frac{\vec{w}}{||\vec{w}||} \in e_3 \\ p_4 &= p_1 + \mu \frac{\vec{w}}{||\vec{w}||} \end{aligned}$$

where

$$\vec{v} = \begin{pmatrix} -\cos\alpha \\ \sin\alpha \end{pmatrix} \text{ and } \vec{w} = \begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix}$$

It has to be ensured that the line

$$g = p_1 + \delta \frac{\vec{w}}{||\vec{w}||}, \delta \in \mathbb{R}$$

intersects e_4 at a point with value $\delta \geqslant \mu$. Our goal is to maximize $\lambda\mu$. This is an optimization problem with an objective function with two parameters and a constant number of polynomial inequalities. Solving this problem takes constant time. Since there are $\mathcal{O}(n^4)$ different possibilities for the edges e_1, e_2, e_3, e_4, we need $\mathcal{O}(n^4)$ time in total.

At the end, the algorithm decides which of the computed rectangles has the largest area. This takes linear time. Thus, the total running time is $\mathcal{O}(n^4)$.

5.1.2 Approximation Algorithms

If we know the direction of one of the sides of R_{opt}, we can compute the largest rectangle R_{opt} itself in $\mathcal{O}(\log n)$ time by applying the algorithm of Alt et al. [17]. The general idea of our algorithm is to approximate the direction of alignment of a largest inscribed rectangle and to prove that the area of the largest inscribed rectangle aligned along this direction also approximates $|R_{\text{opt}}|$. For the computation, we construct a set of candidate directions and find the largest inscribed rectangle along each of these directions using the algorithm of Alt et al. [17]. The number of candidate directions will be $\mathcal{O}(\frac{1}{\epsilon})$ for the randomized version of our algorithm, and $\mathcal{O}(\frac{1}{\epsilon^2})$ or $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ for the deterministic one.

In the following let d_{opt} be a direction of R_{opt}.

5.1.2.1 Approximating the Direction of R_{opt}

The main idea of our algorithm is to find a direction close enough to the direction of any side of R_{opt}. Such a direction will be called an *ϵ-close direction*, for a fixed $\epsilon > 0$. To define what ϵ-close actually means, we first suppose that we know R_{opt}. The intersection of the diagonals of R_{opt} is denoted as its *center* s. Let $\overline{ab}$ be one of the two shortest sides of R_{opt} and let m be the midpoint of the segment $\overline{ab}$, see

Figure 5.7. Then, $\angle(asb) \leqslant \frac{\pi}{2}$, and we define the triangles T_1 and T_2 as the two triangles with vertices s, m and the third vertex being either $e := m - \epsilon(b - m)$ or $f := m + \epsilon(b - m)$. Analogously, choosing the side of R_{opt} opposite of $\overline{ab}$ gives the two triangles T_3 and T_4 having the same area. The area for each triangle T_i is $\epsilon \mid \overline{mb} \mid\mid \overline{sm} \mid /2$ and therefore an $\epsilon/8$-fraction of $|R_{\text{opt}}| = 4 \mid \overline{mb} \mid\mid \overline{sm} \mid$. We define a direction to be *ϵ-close* if the line containing s with that direction intersects $\overline{ef}$.

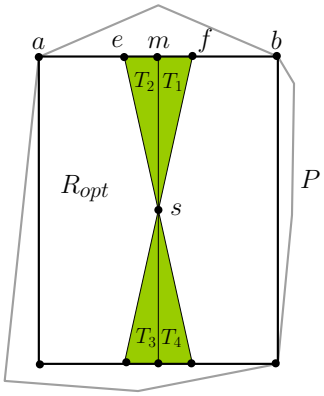


Figure 5.7: A largest rectangle R_{opt} in a convex polygon P. The area for each T_i, $1 \leqslant i \leqslant 4$, is an $\epsilon/8$-fraction of $|R_{\text{opt}}|$.

Now we show that an ϵ-close direction gives us a rectangle R_{apx} with $|R_{\text{apx}}| \geqslant (1 - c\epsilon)|R_{\text{opt}}|$ for a constant c. (This will lead to an $(1 - \epsilon)$ approximation by replacing $c\epsilon$ with ϵ' at the expense of an additional (but small) constant factor in the running time.) For this we have to prove the following lemma.

Lemma 5.1.3. *A largest inscribed rectangle R_{apx} that is aligned to an ϵ-close direction contains an area of at least $(1 - 6.5\epsilon)|R_{opt}|$.*

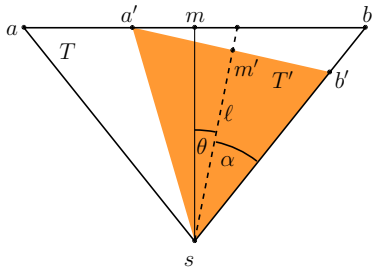


Figure 5.8: The triangle $T = asb$.

Consider the triangle $T = asb$ in R_{opt} (see Figure 5.8) and an ϵ-close direction d_{apx}. Let ℓ be the line with direction d_{apx} that contains s; we assume w. l. o. g. that ℓ intersects $\overline{mf}$ in Figure 5.7. (The case where ℓ intersects $\overline{me}$ is symmetric.) We denote the angle between $\overline{sm}$ and ℓ by θ and the angle between ℓ and $\overline{sb}$ by α. Let

T' be the isosceles triangle of maximum area that is contained in T and symmetric along ℓ (i. e., with ℓ as perpendicular bisector of the base side). Note that T' must contain s as a vertex. Let a' and b' be the two remaining vertices of T' and consider the midpoint m' of the segment $\overline{a'b'}$.

Instead of comparing $|R_{\text{apx}}|$ with $|R_{\text{opt}}|$ directly, we now compare the triangles T and T'. If we can show that $|T'| \geqslant (1 - c\epsilon)|T|$ for some constant c, then the largest rectangle aligned to d_{apx} has at least an area of $(1 - c\epsilon)|R_{\text{opt}}|$. The reduction to triangles does not matter for the approximation, as $|R_{\text{opt}}| = 4|T|$ and $|R_{\text{apx}}| \geqslant 4|T'|$.

Recalling elementary trigonometry we see that

$$\begin{aligned}
\epsilon &= \frac{|mf|}{|mb|} \geqslant \frac{\tan\theta}{\tan(\theta+\alpha)},\\
|T| &= |\overline{sa}|^2 \sin(\alpha+\theta)\cos(\alpha+\theta), \text{ and}\\
|T'| &= |\overline{sa'}|^2 \sin(\alpha)\cos(\alpha) = \frac{|\overline{sm}|^2}{\cos^2(\alpha-\theta)}\sin(\alpha)\cos(\alpha)\\
&= |\overline{sa}|^2 \frac{\cos^2(\alpha+\theta)}{\cos^2(\alpha-\theta)}\sin(\alpha)\cos(\alpha).
\end{aligned}$$

On the other hand, we want to show that

$$\begin{aligned}
&|T'| \geqslant (1-c\epsilon)|T|\\
\Leftrightarrow &\frac{|T'|}{|T|} \geqslant 1 - c\epsilon\\
\Leftrightarrow &\frac{|\overline{sa}|^2 \frac{\cos^2(\alpha+\theta)}{\cos^2(\alpha-\theta)}\sin(\alpha)\cos(\alpha)}{|\overline{sa}|^2\sin(\alpha+\theta)\cos(\alpha+\theta)} \geqslant 1 - c\epsilon\\
\Leftrightarrow &\frac{\sin(\alpha)\cos(\alpha)}{\cos^2(\alpha-\theta)\tan(\alpha+\theta)} \geqslant 1 - c\epsilon
\end{aligned}$$

for a constant c. To prove this inequality, we use the following lemma.

Lemma 5.1.4. *The function* $f(\alpha,\theta) = \frac{\sin(\alpha)\cos(\alpha)}{\cos^2(\alpha-\theta)} + c\tan(\theta) - \tan(\theta+\alpha)$ *is positive for* $0 \leqslant \alpha \leqslant \frac{\pi}{4}, 0 \leqslant \theta \leqslant \frac{\pi}{8}$ *and any constant* $c \geqslant 6.5$.

To prove this lemma, we need the following two propositions.

Proposition 5.1.5. $\frac{1}{4}\tan(x) \leqslant \tan(\frac{x}{3})$ *for* $0 \leqslant x \leqslant \frac{\pi}{4}$.

Proof. Consider the function $f(x) = \frac{1}{4}\tan(x) - \tan(\frac{x}{3})$. We have to show that $f(x) \leqslant 0$ for $0 \leqslant x \leqslant \frac{\pi}{4}$. The first and second derivatives of $f(x)$ with respect to x are:

$$f'(x) = \frac{1}{4}\sec^2(x) - \frac{1}{3}\sec^2(\frac{x}{3}),$$

$$f''(x) = \frac{1}{2}\sec^2(x)\tan(x) - \frac{2}{9}\sec^2(\frac{x}{3})\tan(\frac{x}{3})$$

Since $\tan(x) \geqslant \tan(\frac{x}{3})$, we have

$$f''(x) \geqslant 2\tan(\frac{x}{3})(\frac{1}{4}\sec^2(x) - \frac{1}{9}\sec^2(\frac{x}{3})).$$

Let x' be the root of $f'(x) = 0$. That is, let $x' \in [0, \frac{\pi}{4}]$ be such that

$$\frac{1}{4}\sec^2(x') - \frac{1}{3}\sec^2(\frac{x'}{3}) = 0$$

Since $\tan(x) \geqslant 0$ in the domain $0 \leqslant x \leqslant \frac{\pi}{4}$, we have

$$f''(x') \geqslant 2\tan(\frac{x'}{3})\frac{2}{9}\sec^2(\frac{x'}{3}) \geqslant 0$$

Therefore in the range $[0, \frac{\pi}{4}]$, $f(x)$ attains a minimum whenever $f'(x) = 0$ and the maxima are attained only at the boundary. Since $f(0) = 0$ and $f(\frac{\pi}{4}) < 0$, $f(x) \leqslant 0$ for $0 \leqslant x \leqslant \frac{\pi}{4}$. □

Proposition 5.1.6. *If we choose $\epsilon \leqslant \frac{1}{4}$, then $\theta \leqslant \frac{\alpha}{2}$.*

Proof. This proof uses Proposition 5.1.5 for the inequality marked with $(\times)$.

$$\frac{\tan(\theta)}{\tan(\alpha+\theta)} \leqslant \epsilon \leqslant \frac{1}{4}$$

$$\tan(\theta) \leqslant \frac{1}{4}\tan(\alpha+\theta) \overset{(\times)}{\leqslant} \tan(\frac{\theta+\alpha}{3})$$

$$\theta \leqslant \frac{\theta+\alpha}{3} \text{ for } [0, \frac{\pi}{4}]$$

$$\theta \leqslant \frac{\alpha}{2}$$

□

Now we can continue with the proof of Lemma 5.1.4.

Proof of Lemma 5.1.4.

$$\begin{aligned}
& \frac{\sin(\alpha)\cos(\alpha)}{\cos^2(\alpha-\theta)} + c\tan(\theta) - \tan(\theta+\alpha) \\
= \quad & \tan(\alpha)\frac{\cos^2(\alpha)}{\cos^2(\alpha-\theta)} + c\tan(\theta) - \frac{\tan(\theta)+\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)} \\
= \quad & \tan(\alpha)\frac{\cos^2(\alpha)}{\cos^2(\alpha-\theta)} + c\tan(\theta) - \frac{\tan(\theta)}{1-\tan(\theta)\tan(\alpha)} - \frac{\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)} \\
= \quad & \tan(\alpha)\frac{\cos^2(\alpha)}{\cos^2(\alpha-\theta)} + \tan(\theta)(c - \frac{1}{1-\tan(\theta)\tan(\alpha)}) - \frac{\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)}
\end{aligned}$$

Recall that $0 \leqslant \alpha \leqslant \frac{\pi}{4}$ and $0 \leqslant \theta \leqslant \frac{\pi}{8}$ and thus $\tan(\alpha) \leqslant 1$. Hence $\frac{1}{1-\tan(\theta)\tan(\alpha)} \leqslant \frac{1}{1-\tan(\frac{\pi}{8})} < 1.71$, so it suffices to show that

$$\tan(\alpha)\frac{\cos^2(\alpha)}{\cos^2(\alpha-\theta)} + c'\tan(\theta) - \frac{\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)} \geqslant 0$$

for some constant $c' = c - 1.71 > 0$.

$$\begin{aligned}
& \tan(\alpha)\tfrac{\cos^2(\alpha)}{\cos^2(\alpha-\theta)} + c'\tan(\theta) - \tfrac{\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)} \\
= \quad & \tan(\alpha)\tfrac{1+\tan^2(\alpha-\theta)}{1+\tan^2(\alpha)} + c'\tan(\theta) - \tfrac{\tan(\alpha)}{1-\tan(\theta)\tan(\alpha)} \\
= \quad & \tan(\alpha)\left(\tfrac{1+\tan^2(\alpha-\theta)}{1+\tan^2(\alpha)} + c'\tfrac{\tan(\theta)}{\tan(\alpha)} - \tfrac{1}{1-\tan(\theta)\tan(\alpha)}\right) \\
= \quad & \tan(\alpha)\left(\tfrac{1+\frac{(\tan(\alpha)-\tan(\theta))^2}{(1+\tan(\theta)\tan(\alpha))^2}}{1+\tan^2(\alpha)} + c'\tfrac{\tan(\theta)}{\tan(\alpha)} - \tfrac{1}{1-\tan(\theta)\tan(\alpha)}\right)
\end{aligned}$$

Replacing $\tan(\theta)$ by x and $\tan(\alpha)$ by y, we want to show that

$$y\left(\frac{1+\frac{(y-x)^2}{(1+xy)^2}}{1+y^2} + c'\frac{x}{y} - \frac{1}{1-xy}\right) > 0$$

$$\begin{aligned}
& y\left(\tfrac{1+\frac{(y-x)^2}{(1+xy)^2}}{1+y^2} + c'\tfrac{x}{y} - \tfrac{1}{1-xy}\right) \\
= \quad & y\left(\frac{(1+xy)^2+(y-x)^2}{(1+y^2)(1+xy)^2} + c'\frac{x}{y} - \frac{1}{1-xy}\right) \\
= \quad & y\left(\frac{y(1-xy)\left((1+xy)^2+(y-x)^2\right) + c'x(1+y^2)(1+xy)^2(1-xy) - y(1+y^2)(1+xy)^2}{y(1+y^2)(1+xy)^2(1-xy)}\right) \\
= \quad & y\left(\frac{y(1-xy)(1+x^2y^2+x^2+y^2) + c'x(1+y^2)(1+xy)^2(1-xy) - y(1+y^2)(1+xy)^2}{y(1+y^2)(1+xy)^2(1-xy)}\right) \\
= \quad & y\left(\frac{y(1-xy)(1+x^2)(1+y^2) + c'x(1+y^2)(1+xy)^2(1-xy) - y(1+y^2)(1+xy)^2}{y(1+y^2)(1+xy)^2(1-xy)}\right) \\
= \quad & \frac{y(1-xy)(1+x^2) + c'x(1+xy)^2(1-xy) - y(1+xy)^2}{(1+xy)^2(1-xy)}
\end{aligned}$$

Since the denominator is always larger than 0, it suffices to show that

$$y(1-xy)(1+x^2) + c'x(1+xy)^2(1-xy) - y(1+xy)^2 > 0$$

$$\begin{aligned} & y(1-xy)(1+x^2)+c'x(1+xy)^2(1-xy)-y(1+xy)^2 \\ = \quad & y(1+x^2-xy-x^3y)+(c'x-c'x^2y)(1+2xy+x^2y^2)-y(1+2xy+x^2y^2) \\ = \quad & x^2y-xy^2-x^3y^2+c'x+c'x^3y^2+2c'x^2y-c'x^2y-c'x^4y^3-2c'x^3y^2-x^2y^3-2xy^2 \\ = \quad & x\left((c'+1)xy-3y^2-x^2y^2+c'-c'x^2y^2-c'x^3y^3-xy^3\right) \end{aligned}$$

Recall that $y=\tan(\alpha)$. Since $\tan(\alpha)\leqslant 1$, it suffices to show that $(-3-x^2+c'-c'x^2-c'x^3-x)>0$. That is $(c'-3-x-(c'+1)x^2-c'x^3)>0$. Since $x=\tan(\theta)\leqslant\tan(\frac{\pi}{8})$ it suffices to pick c' such that $c'-3-\tan(\frac{\pi}{8})-(c'+1)\tan^2(\frac{\pi}{8})-c'\tan^3(\frac{\pi}{8})>0$. Thus $c'>\frac{3+\tan(\frac{\pi}{8})+\tan^2(\frac{\pi}{8})}{1-\tan^2(\frac{\pi}{8})-\tan^3(\frac{\pi}{8})}\approx 4.734$. This implies that the function $\frac{\sin(\alpha)\cos(\alpha)}{\cos^2(\alpha-\theta)}+c\tan(\theta)-\tan(\theta+\alpha)$ is positive for $0\leqslant\alpha\leqslant\frac{\pi}{4}, 0\leqslant\theta\leqslant\frac{\pi}{8}$ and for any $c\geqslant 6.5$.

Using Lemma 5.1.4 and the property $\epsilon\geqslant\frac{\tan\theta}{\tan(\theta+\alpha)}$, we obtain the following corollary.

Corollary 5.1.7. Let $0\leqslant\alpha\leqslant\frac{\pi}{4}, 0\leqslant\theta\leqslant\frac{\pi}{8}$ and let $c\geqslant 6.5$. Then

$$\frac{|T'|}{|T|}=\frac{\sin(\alpha)\cos(\alpha)}{\cos^2(\alpha-\theta)\tan(\alpha+\theta)}\geqslant 1-c\frac{\tan\theta}{\tan(\theta+\alpha)}\geqslant 1-c\epsilon.$$

From Corollary 5.1.7 follows immediately that $|T'|\geqslant(1-6.5\epsilon)|T|$. This implies that $|R_{\text{apx}}|\geqslant(1-6.5\epsilon)|R_{\text{opt}}|$ since $4|T|=|R_{\text{opt}}|$ and $4|T'|=|R_{\text{apx}}|$.

5.1.2.2 How to Get an ϵ-close Direction

It remains to show how to compute a direction ϵ-close to d_{opt} efficiently. Assume first that we know the center s of R_{opt}. If we choose $\Theta(\frac{1}{\epsilon})$ random points uniformly distributed inside P, at least one of them lies with high probability in one of the triangles T_1, T_2, T_3 and T_4 since the area of each triangle is an $\epsilon/8$ fraction of $|R_{\text{opt}}|$. Thus, taking the direction from s to this point gives us immediately an ϵ-close direction (see Figure 5.7). As we do not have the information about the location of s, we assume that any other point p inside R_{opt} is given. Then at least one triangle T_i, $1\leqslant i\leqslant 4$, has a translated copy T_i', where the translation maps s to p.

The triangle T_i' is contained in R_{opt} and therefore also contained in P. Picking a point q' in T_i' and taking the direction $\overline{pq'}$ has the same effect as picking a point q in T_i and taking the direction $\overline{sq}$. Thus, we do not have to compute s explicitly. Instead, it is sufficient to find a point inside R_{opt} (see also Figure 5.9).

Even though we do not know R_{opt}, picking points from it essentially amounts to picking points from the input polygon because the area of the largest inscribed rectangle in a convex polygon is at least half of the area of the polygon. This was proven by Radziszewski in 1952 [99]. More formally,

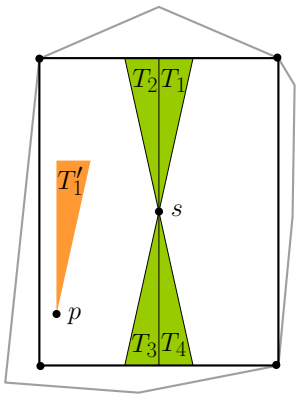


Figure 5.9: T_1' is a translated version of T_1 where s maps p.

Lemma 5.1.8 ([99, Theorem 6]). *Let P be a convex polygon and R_{opt} be a largest inscribed rectangle in P, then $|R_{opt}| \geqslant |P|/2$.*

Randomized algorithm. It follows from Lemma 5.1.8 that if we pick k points sampled uniformly at random from a convex polygon P, the expected number of points inside R_{opt} is $\frac{k}{2}$. All these points are distributed uniformly at random inside R_{opt}. Moreover, if we pick $\Theta(\frac{1}{\epsilon})$ points uniformly at random, the expected number of points inside the triangle T_i' is $\Theta(1)$. This obtains a constant success probability. More formally, let p, q be two points sampled uniformly at random in P then $\mathbf{Pr}[p \in R_{\text{opt}}] = \frac{1}{2}$ and $\mathbf{Pr}[q \in T_i'] = \frac{\epsilon}{16}$. Let k and k' be constants. We pick k random points as possible candidates for a point in R_{opt} and $\frac{k'}{16\epsilon}$ random points as possible candidates for a point in T_i'. Taking all possible pair of points where each pair consists of one point from each candidate set and computing the lines defined by these pairs gives a probability of $(1 - \frac{1}{2^k})(1 - (1 - \frac{\epsilon}{16})^{k'/16\epsilon})$ that one line has an ϵ-close direction. This is called the *probability of success*. On the other hand the *probability of failure* is $\frac{1}{2^k} + (1 - \frac{\epsilon}{16})^{k'/16\epsilon} - \frac{1}{2^k}(1 - \frac{\epsilon}{16})^{k'/16\epsilon}$. By using the following inequalities we can show that these probabilities are constant:

- $(1 + x)^n \geqslant 1 + nx$ (*Bernoulli's inequality*)
- $1 - x = \frac{1-x^2}{1+x} \leqslant \frac{1}{1+x}$, for $x > -1$

It follows that the probability of failure is constant and, hence, so is the probability of success:

$$\begin{aligned}
\frac{1}{2^k} + (1 - \frac{\epsilon}{16})^{k'/16\epsilon} - \frac{1}{2^k}(1 - \frac{\epsilon}{16})^{k'/16\epsilon} &= \frac{1}{2^k} + (1 - \frac{\epsilon}{16})^{k'/16\epsilon}(1 - \frac{1}{2^k}) \\
&\leqslant \frac{1}{2^k} + \frac{1}{(1 + \frac{\epsilon}{16})^{k'/16\epsilon}}(1 - \frac{1}{2^k}) \\
&\leqslant \frac{1}{2^k} + \frac{1}{1 + \frac{k'}{256}}(1 - \frac{1}{2^k}) = \mathcal{O}(1).
\end{aligned}$$

The probability of failure can be decreased to an arbitrary low constant $t' > 0$ by using probability amplification, that is, the algorithm is repeated several times. To achieve a probability of any constant $t' > 0$ the algorithm has to be repeated $\mathcal{O}(\log \frac{1}{t'})$ times. Expressed in a different way, to achieve a success probability of any $t < 1$ the algorithm has to be repeated $\mathcal{O}(\log \frac{1}{1-t})$ times.

Thus, we have the following lemma.

Lemma 5.1.9. *Let P be a convex polygon with n vertices given in cyclic order and let also a source of random points in P be given. Then we can compute a $(1-\epsilon)$-approximation R_{apx} for the largest inscribed rectangle in P with probability t in time $\mathcal{O}(\frac{1}{\epsilon} \log n \log \frac{1}{1-t})$ for any $0 < t < 1$.*

We can achieve the same running time without random points in P being given. It is easy to see that with a preprocessing time of $\mathcal{O}(n \log n)$ we can create a data structure for a (not necessarily convex) polygon P that returns a point distributed uniformly at random inside P in $\mathcal{O}(\log n)$ time per sample. This can be achieved by first computing a triangulation of the point set and then creating a balanced binary tree with the triangles as leaves, where the weight of any node is the sum of areas of all triangles contained in the subtree rooted at that node. Sampling a random point from P then amounts to traversing this tree from root to a leaf and following the left or the right child at any node with the probability proportional to their weights.

Since the ordering of the vertices of P is given and we want to avoid any preprocessing for P, we will not sample points from P uniformly at random. Instead, we take a uniform distribution over a square and fit these points inside the polygon. Thus, the sampling from P will simulate the sampling of random points from a square. Recall that we assume that the ordering of the vertices of P is known in advance. Let v_t, v_b be the topmost and bottommost vertices of P respectively; their computation takes $\mathcal{O}(\log n)$ time. We pick a height h between the two vertices uniformly at random and take the longest horizontal segment that fits inside P at this height. Again, this segment can be computed in $\mathcal{O}(\log n)$ time. We pick a point uniformly at random on this segment. This will be our sample point in P. We can repeat this process as many times as desired to get a large set of sample points that are in P. Each of these sample points can be generated in $\mathcal{O}(\log n)$ time.

We show that such a sampling works for our algorithm. Recall that two points p and q from P are needed such that p lies in a largest inscribed rectangle R_{opt} and q lies in a triangle of area $\Omega(\epsilon)$ that is a translated copy of one of the T_i's (see Figure 5.7). With our sampling method, the probability that a sample point is contained in any convex region Q of area $\epsilon|P|$ will be at least $\frac{\epsilon}{2}$.

Let L_h be the length of the largest horizontal segment inside P at height h, and l_h be the length of the largest horizontal segment inside Q at height h. Also, assume that the bottommost and topmost points in P are at heights 0 and 1 respectively (see Figure 5.10). Then $\frac{|Q|}{|P|} = \frac{\int_0^1 l_h \, dh}{\int_0^1 L_h \, dh}$. The probability that a sample point us-

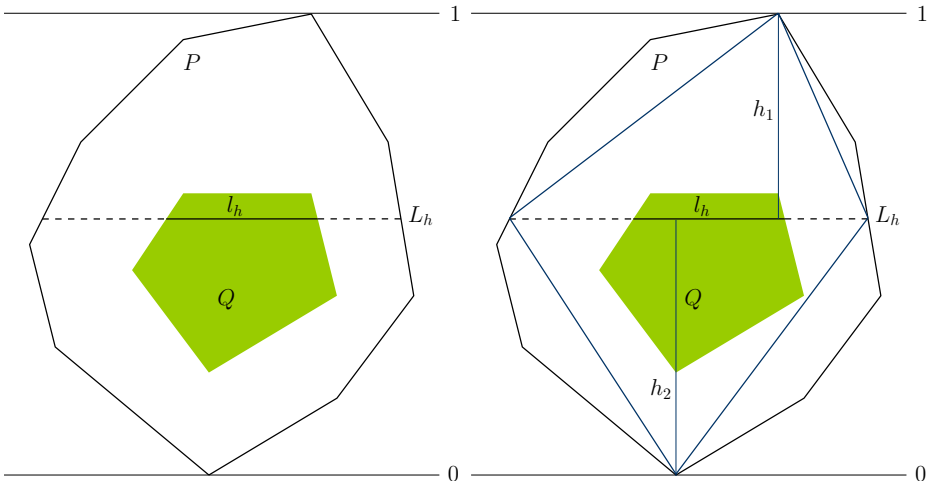


Figure 5.10: Sampling from a convex region Q in P.

ing the above sampling method lies in Q is $\int_0^1 \frac{l_h}{L_h}\,dh$. For any value of h, we can find a quadrilateral that fits inside P and has area at least $\frac{L_h}{2}$. This implies that $\frac{L_h}{2} \leqslant \int_0^1 L_h\,dh$ and

$$\int_0^1 \frac{l_h}{L_h}\,dh \geqslant \frac{1}{2}\frac{\int_0^1 l_h\,dh}{\int_0^1 L_h\,dh}.$$

Since each of these sample points can be generated in logarithmic time, the complexity of our algorithm is $\mathcal{O}(\frac{1}{\epsilon}\log n)$. We summarize the steps in Algorithm 6.

Algorithm 6

Input: A convex polygon P in the plane.

Output: An inscribed rectangle in P that is a $(1-\epsilon)$-approximation of the largest one.

1: Take $\Theta(1)$ points in P with the aforementioned distribution and store them in U.
2: Take $\Theta(1/\epsilon)$ points in P with the aforementioned distribution and store them in V.
3: $|R_{\text{apx}}| = 0$
4: **for all** $u \in U$ **do**
5: **for all** $v \in V$ **do**
6: Compute the largest inscribed rectangle S that is aligned to $\overline{uv}$.
7: **if** $|S| \geqslant |R_{\text{apx}}|$ **then**
8: $R_{\text{apx}} = S$
return R_{apx}

Theorem 5.1.10. *Let P be a convex polygon with n vertices given in cyclic or-*

der. An inscribed rectangle in P with area of at least $(1-\epsilon)$ times the area of a largest inscribed rectangle can be computed with probability t in $\mathcal{O}(\frac{1}{\epsilon}\log n \log \frac{1}{1-t})$ deterministic time for any constant $t < 1$.

Deterministic algorithm. For the deterministic case, it remains to show how the algorithm computes sample points in P. First, we compute an enclosing rectangle R_e of P such that $|R_e|$ is only a constant factor times bigger than $|P|$. This can be done using the following lemma due to Ahn et al. [7].

Lemma 5.1.11 ([7, Lemma 5])**.** *Let P be a convex polygon with n vertices given in cyclic order. Then there is an algorithm that computes an enclosing rectangle R such that $P \subset R$ and $|R| \leqslant 2\sqrt{2}|P|$ in $\mathcal{O}(\log n)$ time.*

Creating a grid of constant size in an enclosing rectangle R of Lemma 5.1.11 allows us to ensure a constant number of grid points in P. This is proven in Lemma 5.1.13 by using Pick's theorem.

Theorem 5.1.12 (Pick's Theorem [97])**.** *Let an integer grid and a simple polygon P with all vertices lying on the grid points be given. Let i be the number of grid points contained in P and b be the number of grid points on the boundary of P. Then $|P| = \frac{b}{2} + i - 1$.*

Lemma 5.1.13. *For a convex set S, a constant c, and every enclosing rectangle R of S with $|R| \leqslant c|S|$, S contains at least $\frac{k^2}{2c}$ grid points of an $k \times k$ grid on R for k being a sufficiently large constant ($k \geqslant 8c$).*

Proof. Let G be a $k \times k$ grid on R and let $|R| = 1$. We shrink S to a maximum area polygon $S' \subseteq S$ having all vertices on grid points of G. Because of convexity, $|S| - |S'|$ is at most the area of $4k$ grid cells.

$$
\begin{aligned}
|S'| &\geqslant |S| - 4k\frac{1}{k^2} \geqslant \frac{1}{c} - \frac{4}{k} \\
|S'| &= (\frac{b}{2} + i - 1)\frac{1}{k^2} \geqslant \frac{1}{c} - \frac{4}{k} \text{ (by Pick's theorem)} \\
b + i &\geqslant \frac{k^2}{c} - 4k + 1
\end{aligned}
$$

Thus, at least $\frac{1}{2c}k^2$ grid points lie in S, for k being a sufficiently large constant ($k \geqslant 8c$). □

It follows from Lemma 5.1.13 that choosing a grid with constant size on the rectangle R implies that R_{opt} in P contains a constant number of grid points (in fact $\frac{k^2}{2c}$). For the next algorithm (Algorithm 7), we will only use one of these grid points. Additionally, Lemma 5.1.13 shows that every ϵ-fraction of P, in particular

Algorithm 7

Input: A convex polygon P in the plane.

Output: An inscribed rectangle in P that is a $(1-\epsilon)$-approximation of the largest one.

Compute an enclosing rectangle R_e with area $|R_e| \leqslant 2\sqrt{2}|P|$
Compute a $\Theta(1) \times \Theta(1)$ grid on R_e. Let G_1 be the set of grid points.
Compute a $\Theta(\frac{1}{\epsilon}) \times \Theta(\frac{1}{\epsilon})$ grid on R_e. Let G_2 be the set of grid points.
$|R_{\text{apx}}| = 0$
for all $u \in G_1$ **do**
    for all $v \in G_2$ **do**
        Compute the largest inscribed rectangle S that is aligned to $\overline{uv}$
        if $|S| \geqslant |R_{\text{apx}}|$ **then**
            $R_{\text{apx}} = S$
return R_{apx}

every triangle T_i', contains many grid points for a big enough grid on R. This fact will be used later to improve the running time of the algorithm with ϵ-nets.

The idea is to take two grids G_1 and G_2 on R of size $\Theta(1)\times\Theta(1)$ and $\Theta(\frac{1}{\epsilon})\times\Theta(\frac{1}{\epsilon})$, respectively, iterate through all pairs of grid points in $G_1 \times G_2$ and get at least one pair (u, v) with $u \in R_{\text{opt}}$ and $v \in T_i'$ using Lemma 5.1.13. Hence, the direction of $\overline{uv}$ is ϵ-close. We summarize the steps in Algorithm 7.

Algorithm 7 is deterministic with running time $\mathcal{O}(\frac{1}{\epsilon^2}\log n)$. We can further reduce the running time to $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log n + \frac{1}{\epsilon^{28}})$ by using the tools from the theory of ϵ-nets. Here we just give an outline of how these tools can be used, and we refer the reader to [88] for more details.

A subset S' of a given set S of N points is called an ϵ-net for S with respect to a set of objects, if any object containing at least $\frac{\epsilon}{2}N$ points of S contains a point of S'. For objects with VC-dimension d, a subset S' of size $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ always exists and can be computed in deterministic time $\mathcal{O}(N^{2d})$. Triangles have VC-dimension 7, and we consider the set S of grid points of a $\frac{1}{\epsilon} \times \frac{1}{\epsilon}$ grid, so $N = \frac{1}{\epsilon^2}$. Thus, we can compute an ϵ-net for S of size $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ in time $\mathcal{O}(\frac{1}{\epsilon^{28}})$.

Theorem 5.1.14. *Let P be a convex polygon with n vertices given in cyclic order. An inscribed rectangle in P with area of at least $(1-\epsilon)$ times the area of a largest inscribed rectangle can be computed*

- *in $\mathcal{O}(\frac{1}{\epsilon^2}\log n)$ deterministic time.*
- *in $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log n + \frac{1}{\epsilon^{28}})$ deterministic time.*

5.2 Largest Inscribed Rectangles in Simple Polygons

The same ideas can be used to approximate the largest inscribed rectangle in a simple polygon with or without holes. The largest inscribed rectangle R_{opt} in a simple polygon (with or without holes) with n vertices has an area of at least $\frac{1}{2(n-2)}$ times the area of the polygon. (This can be proven by first triangulated the polygon; then the area of the largest inscribed rectangle in the largest of these triangles has an area of at least $\frac{1}{2(n-2)}$ times the area of the polygon.) Moreover, a largest axis-aligned rectangle in a simple polygon can be computed in $\mathcal{O}(n \log n)$ time [29] and in a simple polygon with holes in $\mathcal{O}(n \log^2 n)$ time [37]. Since $|R_{\text{opt}}|$ is an $\Omega(\frac{1}{n})$-fraction of P and the area of each of the four triangles inside R_{opt} is an $\Omega(\frac{\epsilon}{n})$-fraction of P, we get the following running times for computing an inscribed rectangle of area at least $(1-\epsilon)R_{\text{opt}}$.

- For simple polygons: With constant probability in time $\mathcal{O}(\frac{1}{\epsilon}n^3 \log n)$.
- For polygons with holes: With constant probability in time $\mathcal{O}(\frac{1}{\epsilon}n^3 \log^2 n)$.

In comparison with the algorithm of Hall-Holt et al. [66], which deals only with fat rectangles, our algorithm can handle general rectangles at the expense of a slower running time.

5.3 Conclusions and Open Problems

We gave efficient randomized and deterministic approximation algorithms for the problem of computing a largest area inscribed rectangle in convex and simple polygons.

One related open problem is to approximate a largest perimeter inscribed rectangle. Our algorithms base on the fact that the area of a largest area inscribed rectangle has constant fraction of the area of the polygon itself. This is not the case for a largest perimeter rectangle. Consider a triangle with two angles being strictly smaller than $\pi/4$. Then the largest perimeter rectangle is exactly the largest side of this triangle, hence its area is zero.

We have shown how to extend the approximation algorithm for the largest area inscribed rectangle in convex polygons to simple polygons. However, it is reasonable to ask whether there is a more efficient algorithm. Another remaining open problem is to find an efficient exact algorithm for computing a largest area inscribed rectangle in a convex polygon. The stated straightforward algorithm did not use the given characterization of R_{opt}. We believe that this characterization will help finding an efficient algorithm.

Concluding Remarks

In this thesis we studied different geometric stabbing and covering problems in the plane. The thesis consists of two parts. The first part dealt with stabbing problems. We first considered the problem of stabbing geometric objects with the boundary of a convex polygon; this boundary is then called a convex stabber. We proved the problem to be NP-hard for most types of geometric objects. Additionally, we studied the corresponding optimization problem and proved it to be NP-hard in most of our studied variants. In the next chapter, we studied the problem of stabbing objects in the plane with the vertices of a polygon. To the best of our knowledge, this problem was not studied so far. We gave the first polynomial time algorithm. The last chapter dealt with stabbing sequences of objects with sequences of points. More precisely, we studied the problem of stabbing two sequences of segments (resp. disks) with two point sequences where the discrete Fréchet distance between these point sequences is as small as possible. We gave efficient algorithms for both cases.

The second part of the thesis dealt with covering problem. We first considered a new version of the two center problem where the input is a set $\mathcal{D}$ of disks in the plane. We distinguished between two different variants: First, we studied the problem of finding two smallest congruent disks such that each disk in $\mathcal{D}$ is intersected by one of these two disks. Then, we studied the problem of covering the set $\mathcal{D}$ by two smallest congruent disks. We also investigated an optimization version. We gave efficient exact and approximation algorithms for all of these variants.

Finally, in the last chapter, we investigated the problem of computing a largest area rectangle inscribed in a convex polygon with n vertices. For the case where the order of the vertices of the polygon is given, we gave efficient approximation algorithms whose running times are only logarithmically dependent on n.

At the end of each chapter, we gave some conclusions and stated open problems. So, problems for future work on the different topics can be found at the end of the corresponding chapters.

Since we considered all problems in the plane, a question that arises for all studied problems, is the question how these problems behave in higher dimensions. We give a short overview of the known results so far:
For the problem of finding a convex stabber, it was proven by Arkin et al. [23] that in

three dimensions, it is NP-hard to find a convex (surface) stabber for a set of balls. To the best of our knowledge, the problem of stabbing objects in three dimensions (or higher) with the vertices of a polytope was not considered so far. However, our algebraic algorithm can be generalized to higher dimensions with an increase of the running time.

The algorithms we stated in Chapter 3 for stabbing sequences of segments (or disks) with two point sequences under the condition that the discrete Fréchet distance between these point sequences is a small as possible can also be generalized to higher dimension; for details see [11].

It is known, that the two-center problem for points in $\mathbb{R}^d$ is NP-complete, if d is not fixed [92]. Recently, Agarwal et al. [3] presented efficient randomized algorithms for the two-center problem in three dimension. We are not aware of if the two-center problem for balls in three or higher dimension has been considered so far.

As already stated, there is no efficient exact algorithm known that computes a largest inscribed rectangle in a convex polygon. Hence, it might be more reasonable to study the planar case more in detail than to consider the problem in higher dimensions.

Bibliography

[1] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9(5):495–514, 1993.

[2] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *CoRR*, abs/1204.5333, 2012.

[3] P. K. Agarwal, R. B. Avraham, and M. Sharir. The 2-center problem in three dimensions. In *Proceedings of the 26th ACM Symposium on Computational Geometry (SoCG'10)*, pages 87–96, 2010.

[4] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[5] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.

[6] P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[7] H.-K. Ahn, P. Braß, O. Cheong, H.-S. Na, C.-S. Shin, and A. Vigneron. Inscribing an axially symmetric polygon and other approximation algorithms for planar convex sets. *Comput. Geom.*, 33(3):152–164, 2006.

[8] H.-K. Ahn, S.-S. Kim, C. Knauer, L. Schlipf, C.-S. Shin, and A. Vigneron. Covering and piercing disks with two centers. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC'11)*, volume 7074 of *LNCS*, pages 50–59, 2011. http://dx.doi.org/10.1007/978-3-642-25591-5_7.

[9] H.-K. Ahn, S.-S. Kim, C. Knauer, L. Schlipf, C.-S. Shin, and A. Vigneron. Covering and piercing disks with two centers. *Comput. Geom.*, 46(3):253–262, 2013. http://www.sciencedirect.com/science/article/pii/S0925772112001125.

[10] H.-K. Ahn, C. Knauer, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the discrete Fréchet distance with imprecise input. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC'10)*,

volume 6507 of *LNCS*, pages 422–433, 2010. http://dx.doi.org/10.1007/978-3-642-17514-5_36.

[11] H.-K. Ahn, C. Knauer, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the discrete Fréchet distance with imprecise input. *Int. J. Comput. Geometry Appl.*, 22(1):27–44, 2012. http://www.worldscientific.com/doi/abs/10.1142/S0218195912600023.

[12] H.-K. Ahn, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the discrete Fréchet distance with imprecise input. In *Proceedings of the 12th Korea-Japan Joint Workhop on Algorithms and Computation (WAAC'09)*, pages 132–137, 2009.

[13] D. Alberts and M. R. Henzinger. Average case analysis of dynamic graph algorithms. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 312–321, 1995.

[14] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Inf. Process. Lett.*, 37(4):237–240, 1991.

[15] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

[16] H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[17] H. Alt, D. Hsu, and J. Snoeyink. Computing the largest inscribed isothetic rectangle. In *Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG'95)*, pages 67–72, 1995.

[18] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, volume 2010 of *LNCS*, pages 63–74, 2001.

[19] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[20] E. Althaus, K. Mehlhorn, S. Näher, and S. Schirra. Experiments on curve reconstruction. In *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX'00)*, pages 103–114, 2000.

[21] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 705–706, 2000.

[22] E. M. Arkin, C. Dieckmann, C. Knauer, J. S. Mitchell, V. Polishchuk, L. Schlipf, and S. Yang. Convex transversals. *Comput. Geom.*, 47(2):224–239, 2014. http://www.sciencedirect.com/science/article/pii/S0925772112001368.

[23] E. M. Arkin, C. Dieckmann, C. Knauer, J. S. B. Mitchell, V. Polishchuk, L. Schlipf, and S. Yang. Convex transversals. In *Proceedings of the 12th Algorithms and Data Structures Symposium (WADS'11)*, volume 6844 of *LNCS*, pages 49–60, 2011. http://dx.doi.org/10.1007/978-3-642-22300-6_5.

[24] B. Aronov and S. Har-Peled. On approximating the depth and related problems. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 886–894, 2005.

[25] T. Asano. Reporting and counting intersections of lines within a polygon. In *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC'95)*, volume 834 of *LNCS*, pages 652–659, 1994.

[26] D. Avis and M. E. Houle. Computational aspects of Helly's theorem and its relatives. *Int. J. Comput. Geometry Appl.*, 5(4):357–367, 1995.

[27] S. Basu, R. Pollack, and M.-F. Roy. On computing a set of points meeting every cell defined by a family of polynomials on a variety. *J. Complexity*, 13(1):28–37, 1997.

[28] S. Bespamyatnikh and D. G. Kirkpatrick. Rectilinear 2-center problems. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 68–71, 1999.

[29] R. P. Boland and J. Urrutia. Finding the largest axis aligned rectangle in a polygon in $O(n \log n)$ time. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 41–44, 2001.

[30] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[31] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura. Facility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research*, 202(1):99–106, 2010.

[32] S. Cabello, J. M. Díaz-Báñez, C. Seara, J. A. Sellarès, J. Urrutia, and I. Ventura. Covering point sets with two disjoint disks or squares. *Comput. Geom.*, 40(3):195–206, 2008.

[33] T. M. Chan. Deterministic algorithms for 2-d convex programming and 3-d online linear programming. *J. Algorithms*, 27(1):147–166, 1998.

[34] T. M. Chan. More planar two-center algorithms. *Comput. Geom.*, 13(3):189–198, 1999.

[35] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, 1987.

[36] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.

[37] K. L. Daniels, V. J. Milenkovic, and D. Roth. Finding the largest area axis-parallel rectangle in a polygon. *Comput. Geom.*, 7:125–148, 1997.

[38] L. Danzer, B. Grünbaum, and V. Klee. Helly's theorem and its relatives. In *Proceedings of the Seventh Symposium in Pure Mathematics*, pages 101–180, 1963.

[39] M. de Berg, S. Cabello, and S. Har-Peled. Covering many or few points with unit disks. *Theory Comput. Syst.*, 45(3):446–469, 2009.

[40] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008.

[41] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Proceedings of the 16th Annual International Conference on Computing and Combinatorics (COCOON'10)*, volume 6196 of *LNCS*, pages 216–225, 2010.

[42] A. DePano, Y. Ke, and J. O'Rourke. Finding largest inscribed equilateral triangles and squares. In *Proc. 25th Allerton Conf. Commun. Control Comput.*, pages 869–878, Oct. 1987.

[43] C. Dieckmann. Symmetry detection and approximation. *PhD thesis, FU Berlin*, 2012.

[44] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theor. Comput. Sci.*, 27:241–253, 1983.

[45] Z. Drezner. The planar two-center and two-median problems. *Transportation Science*, 18:351–361, 1984.

[46] Z. Drezner and H. W. Hamacher. *Facility location - applications and theory*. Springer, 2002.

[47] H. Edelsbrunner. Finding transversals for sets of simple geometric figures. *Theor. Comput. Sci.*, 35:55–69, 1985.

[48] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22(3):274–281, 1982.

[49] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[50] H. Edelsbrunner and M. Sharir. The maximum number of ways to stab n convex nonintersecting sets in the plane is 2n-2. *Discrete & Computational Geometry*, 5:35–42, 1990.

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

[52] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

[53] D. Eppstein. Faster construction of planar two-centers. In M. E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 131–138, 1997.

[54] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[55] S. Fekete. Geometry and the travelling salesman problem. *PhD thesis, University of Waterloo*, 1992.

[56] K. Fischer and B. Gärtner. The smallest enclosing ball of balls: combinatorial structure and algorithms. In *Proceedings of the 19th ACM Symposium on Computational Geometry (SoCG'03)*, pages 292–301, 2003.

[57] J. D. Foley, A. van Dam, S. Feiner, and J. F. Hughes. *Computer graphics - principles and practice.* Addison-Wesley, 2 edition, 1990.

[58] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, 18(7):409–413, 1975.

[59] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[60] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[61] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[62] J. E. Goodman, R. Pollack, and R. Wenger. Geometric transversal theory. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 163–198. Springer Verlag, 1993.

[63] M. T. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Computer Vision, Graphics, and Image Processing*, 49(2):152–170, 1990.

[64] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geometry Appl.*, 3(4):383–415, 1993.

[65] H. Hadwiger. Über Eibereiche mit gemeinsamer Treffgeraden. *Portugaliae Mathematica*, 16:23–29, 1957.

[66] O. A. Hall-Holt, M. J. Katz, P. Kumar, J. S. B. Mitchell, and A. Sityon. Finding large sticks and potatoes in polygons. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 474–483, 2006.

[67] S. Har-Peled and M. Lee. Weighted geometric set cover problems revisited. *JoCG*, 3(1):65–85, 2012.

[68] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht Deutsche Mathematiker Vereingung*, 32:175–176, 1923.

[69] D. S. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[70] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[71] D. Husemöller. *Elliptic Curves.* Graduate texts in mathematics. Springer, 2003.

[72] R. Z. Hwang, R. C. T. Lee, and R. C. Chang. The slab dividing approach to solve the Euclidean p-center problem. *Algorithmica*, 9(1):1–22, 1993.

[73] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. *Theor. Comput. Sci.*, 80(2):227–262, 1991.

[74] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. *PhD thesis, FU Berlin*, 1991.

[75] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

[76] M. Katchalski, T. Lewis, and J. Zaks. Geometric permutations for convex sets. *Discrete Mathematics*, 54(3):271–284, 1985.

[77] M. J. Katz and F. Nielsen. On piercing sets of objects. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry (SoCG'96)*, pages 113–121, 1996.

[78] L. E. Kavraki and S. M. LaValle. Motion planning. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 109–131. Springer, 2008.

[79] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96)*, pages 329–337, 1996.

[80] J. King and E. Krohn. Terrain guarding is NP-hard. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 1580–1593, 2010.

[81] C. Knauer, M. Löffler, M. Scherfenberg, and T. Wolle. The directed Hausdorff distance between imprecise point sets. *Theor. Comput. Sci.*, 412(32):4173–4186, 2011.

[82] C. Knauer, L. Schlipf, J. M. Schmidt, and H. R. Tiwary. Largest inscribed rectangles in convex polygons. *J. Discrete Algorithms*, 13:78–85, 2012. http://www.sciencedirect.com/science/article/pii/S1570866712000160.

[83] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[84] M. Löffler. Data imprecision in computational geometry. *PhD thesis, Utrecht University*, 2009.

[85] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010.

[86] M. Löffler and M. J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[87] M. Löffler and M. J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom.*, 43(4):419–433, 2010.

[88] J. Matousek. Approximations and optimal geometric divide-and-conquer. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC'91)*, pages 505–511, 1991.

[89] M. McKenna, J. O'Rourke, and S. Suri. Finding the largest rectangle in an orthogonal polygon. In *Proc. 23rd Allenon Conference on Communication, Control and Computing*, pages 486–495, 1985.

[90] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[91] N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4:605–610, 1989.

[92] N. Megiddo. On the complexity of some geometric problems in unbounded dimension. *J. Symb. Comput.*, 10(3/4):327–334, 1990.

[93] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984.

[94] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Oper. Res. Lett.*, 1(5):194–197, 1982.

[95] A. Moffat and A. Turpin. *Compression and Coding Algorithms*. Kluwer, 2002.

[96] K. Mulmuley. *Computational geometry - an introduction through randomized algorithms*. Prentice Hall, 1994.

[97] G. Pick. Geometrisches zur Zahlenlehre. *Sitzungber. Lotos, Naturwissen Zeitschrift, Prague*, 19:311–319, 1899.

[98] R. Pollack and M.-F. Roy. On the number of cells defined by a set of polynomials. *C. R. Acad. Sci. Paris*, 316:573–577, 1991.

[99] K. Radziszewski. Sur une problème extrémal relatif aux figures inscrites et circonscrites aux figures convexes. *Ann. Univ. Mariae Curie-Sklodowska, Sect. A6*, pages 5–18, 1952.

[100] D. Rappaport. Minimum polygon transversals of line segments. *Int. J. Comput. Geometry Appl.*, 5(3):243–256, 1995.

[101] M. Reichling. On the detection of a common intersection of k convex objects in the plane. *Inf. Process. Lett.*, 29(1):25–29, 1988.

[102] G. Rote. Computing the Fréchet distance between piecewise smooth curves. *Comput. Geom.*, 37(3):162–174, 2007.

[103] M. Scherfenberg. Searching point patterns, matching imprecise point patterns, and inducing polygons. *PhD thesis, FU Berlin*, 2012.

[104] L. Schlipf. New results on convex stabbers. In *Proceedings of the 29th European Workshop on Computational Geometry (EuroCG'13)*, pages 57–60, 2013.

[105] M. I. Shamos and D. Hoey. Geometric intersection problems. In *17th Annual Symposium on Foundations of Computer Science (FOCS'76)*, pages 208–215, 1976.

[106] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete & Computational Geometry*, 18(2):125–134, 1997.

[107] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS'92)*, volume 577 of *LNCS*, pages 569–579, 1992.

[108] C.-S. Shin, J.-H. Kim, S. K. Kim, and K.-Y. Chwa. Two-center problems for a convex polygon (extended abstract). In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA'98)*, volume 1461 of *LNCS*, pages 199–210, 1998.

[109] A. Tamir. Problem 4-2 (New York University, Dept. of Statistics and Operations Research), Problems Presented at the Fourth NYU Computational Geometry Day (3/13/87).

[110] L. Tan. The group of rational points on the unit circle. *Mathematics Magazine*, 69(3):pp. 163–171, 1996.

[111] M. J. van Kreveld and M. Löffler. Approximating largest convex hulls for imprecise points. *J. Discrete Algorithms*, 6(4):583–594, 2008.

[112] R. C. Veltkamp. Shape matching: Similarity measures and algorithms. In *Shape Modeling International*, pages 188–197. IEEE Computer Society, 2001.

[113] I. Wegener. *Complexity theory - exploring the limits of efficient algorithms.* Springer, 2005.

[114] R. Wenger. Upper bounds on geometric permutations for convex sets. *Discrete & Computational Geometry*, 5:27–33, 1990.

[115] R. Wenger. Helly-type theorems and geometric transversals. In *Handbook of Discrete and Computational Geometry*, pages 73–96. CRC Press, Florida, 2 edition, 2004.

[116] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit ohne fremde Hilfe und nur mit den angegebenen Quellen verfasst habe. Die Stellen, die ich dem Wortlaut oder dem Sinn nach anderen Werken entnommen habe, sind durch Angabe der Quellen kenntlich gemacht.

Berlin,

Curriculum Vitae

Der Lebenslauf ist in der Online-Version aus Gründen des Datenschutzes nicht enthalten.