ON PROXIMITY PROBLEMS IN EUCLIDEAN SPACES

by
Luis Barba

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
June, 2016

Table of Contents

Abstract

In this work, we focus on two kinds of problems involving the proximity of geometric objects. The first part revolves around intersection detection problems. In this setting, we are given two (or more) geometric objects, and we are allowed to preprocess them separately. Then, the objects are translated and rotated within a geometric space, and we need to efficiently test if they intersect in these new positions. We develop representations of convex polytopes in any (constant) dimension that allow us to perform this intersection test in logarithmic time.

In the second part of this work, we turn our attention to facility location problems. In this setting, we are given a set of sites in a geometric space and we want to place a facility at a specific place in such a way that the distance between the facility and its farthest site is minimized. We study first the constrained version of the problem, in which the facility can only be placed within a given geometric domain. We then study the facility location problem under the geodesic metric. In this setting, we consider a different way to measure distances: Given a simple polygon, we say that the distance between two points is the length of the shortest path that connects them while staying within the given polygon. In both cases, we present algorithms to find the optimal location of the facility. In the process of solving facility location problems, we rely heavily on geometric structures called Voronoi diagrams. These structures summarize the proximity information of a set of "simple" geometric objects in the plane and encode it as a decomposition of the plane into interior disjoint regions whose boundaries define a plane graph. We study the problem of constructing Voronoi diagrams incrementally by analyzing the number of edge insertions and deletions needed to maintain its combinatorial structure as new sites are added.

Chapter 1

Introduction

The main objective of this work is to present several results in the area of Computational Geometry. From a general point of view, Computational Geometry is a field within Computer Science that studies geometric objects embedded in a given space and their interactions. This field emerged as part of the design and implementation of algorithms during the 1970's. Since then, problems in Computational Geometry have received the attention of the research community due to their applications in fields such as computer graphics, geographical information systems (GIS), robotics, communication, networks, operation research, etc.

In this work, we focus on two kind of problems involving the proximity of geometric objects: Chapters 2 and 3 provide an introduction and background to the problems studied in this work. Chapter 4 deals with intersection detection while Chapters 5, 6 and 7 deal with Voronoi diagrams and Facility location problems. A brief introduction and summary of each part is given below. The first section of each chapter provides a more detailed introduction.

1.1 Intersection detection of convex polyhedra

Constructing or detecting the intersection between geometric objects is probably one of the first and most important applications of computational geometry. It was one of the main questions addressed in Shamos' seminal paper that lay the grounds of computational geometry [84], the first application of the plane sweep technique [85], and is still the topic of several volumes being published today.

It is hard to overstate the importance of finding efficient algorithms for intersection testing or collision detection as this class of problems has several applications in motion planning, robotics, computer graphics, Computer-Aided Design, VLSI design and more. They provide significant aid in engineering analysis as they allow to

simulate experiments which are impractical to conduct. Computer Graphics comprehensively includes a broad set of applications related to Virtual Reality, Physical Simulation and Computer-Aided Design that require fast algorithms to determine collisions between geometric objects.

In the modeling of 3D solids, computing the intersection of two 3-dimensional shapes is an important step in defining complex solids. In the field of computer graphics, testing if two or more objects overlap a viewing window is an example of a collision detection problem, as is determining the intersection of a ray and a collection of geometric solids.

In this work we focus on detecting intersections between convex polygons and between convex polyhedra. This topic has received much attention because it serves as a fundamental building block in many collision detection packages. For more information on collision detection and intersection detection refer to [57, 65] and to Chapter 38 of the Handbook of Computational Geometry [49].

In the plane and in 3D space, optimal linear-time algorithms are known to compute the intersection of two given convex polyhedra. A natural extension of this problem is to consider the effect of preprocessing on the complexity of intersection detection problems. In this case, significant improvements in the query time are possible. It is worth noting that each object should be preprocessed separately which allows us to work with large families of objects, and to introduce new objects without triggering a reconstruction of the whole structure.

There exist algorithms that allow us to test if two given convex polygons in the plane intersect in logarithmic time. However, most of them rely on complicated case analysis to solve the problem. In Chapter 4 we show an alternate (and arguably simpler) algorithm to determine if two convex polygons intersect.

In all these 2D algorithms, preprocessing is unnecessary if the polygon is represented by an array with the vertices of the polygon in sorted order along its boundary. In 3D-space (and in higher dimensions) however, the need for preprocessing is more evident as the traditional DCEL representation of the polyhedron is not sufficient to perform fast queries.

Whether the intersection of two preprocessed polyhedra can be tested in logarithmic time is an open question that was implicit in the paper of Chazelle and Dobkin [30] in STOC'80, and explicitly posed in 1983 by Dobkin and Kirkpatrick [37]. The open problem was listed again in 2004 by David Mount in Chapter 38 of the Handbook of Computational Geometry [49]. Together with this question in 3D-space, Dobkin and Kirkpatrick [37] asked if it is possible to extend these results to higher dimensions, i.e., to independently preprocess two polyhedra in $\mathbb{R}^d$ such that their intersection could be tested in logarithmic time.

In Chapter 4, we show how to independently preprocess polyhedra in any bounded dimension such that their intersection can be tested in logarithmic time. A preliminary version of this result first appeared in the proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15) [15].

1.2 Voronoi diagrams

When dealing with a few objects of large complexity, preprocessing each of them independently can lead to significant improvements for intersection detection queries. On the other hand, if many objects of low complexity are involved, then this approach may take a long time. For example, testing if a moving object (say a disk) intersects any element in a cloud of points S is a problem where too many intersections may need to be checked using the approach of the previous section. To speed up this intersection test, we can use general data structures that summarize the proximity information of each obstacle in S. In particular, if we could know which is the closest point in this cloud to the moving object, we could solve the intersection test by simply asking whether this object intersects this obstacle. A structure to answer this question is the *nearest-point Voronoi diagram* whose computation is crucial in motion planning and collision detection [57, 65].

Let S be a set of n sites (points) in the plane. Intuitively, the nearest-point Voronoi diagram of S assigns to each point in the plane its nearest site in S and encodes it as a decomposition of the plane into interior-disjoint regions, each of which is associated with a site $s \in S$ and contains all the points that are closer to s than to any other site in S.

Several variants of Voronoi diagrams exist, for example, if we ask to assign each point in the plane to its farthest site in S instead of the nearest, we obtain the so called *farthest-point Voronoi diagram*. A similar decomposition of the plane into (farthest-point) *Voronoi regions* is obtained, each of which is associated with a site $s \in S$ and contains all the points that have s as its farthest site. The boundaries of these regions define a plane graph of linear size in n which, together with the Voronoi regions, defines the farthest-point Voronoi diagram of S. In Section 3.4 we formally define the farthest-point Voronoi diagram and study several of its properties.

In general, Voronoi diagrams capture much of the proximity information of a set and have therefore several applications in computational geometry, shape reconstruction, computational biology, and machine learning. For more information on Voronoi diagrams refer to [10].

A major open question is whether or not these Voronoi diagrams can be efficiently maintained as new sites are inserted or removed (the fully dynamic problem). In this work, we focus solely on the addition of new sites (the semi-dynamic problem) which leads to incremental algorithms to construct Voronoi diagrams.

Formally, in Chapter 5 we study the amortized number of combinatorial changes (edge insertions and removals) needed to update the graph structure of Voronoi diagrams as sites are added to the set. To that effect, we define a general update operation for planar graphs that can be used to model the incremental construction of several variants of Voronoi diagrams as well as the incremental construction of an intersection of halfspaces in $\mathbb{R}^3$. We show that the amortized number of edge insertions and removals needed to add a new site to the Voronoi diagram is $O(\sqrt{n})$, where n is the number of sites previous to the insertion. A matching $\Omega(\sqrt{n})$ combinatorial lower bound is shown, even in the case where the graph representing the Voronoi diagram is a tree.

We then present a semi-dynamic data structure that maintains the farthest-point Voronoi diagram of a set S of n sites in convex position. This data structure supports the insertion of a new site p (and hence the addition of its Voronoi cell) and finds the asymptotically minimal number K of edge insertions and removals needed to obtain the diagram of $S \cup \{p\}$ from the diagram of S, in worst-case time $O(K \operatorname{polylog} n)$,

which is $O(\sqrt{n}\ \text{polylog}\ n)$ amortized by the aforementioned combinatorial result.

A preliminary version of this result first appeared in the proceedings of the 32nd International Symposium on Computational Geometry [4].

1.3 Facility location problems

In a facility location problem, we are provided with a set of clients and their locations that demand a service to be placed somewhere (gas station, hospital, stores, phone antennas, etc.); such a service is offered by a set of facilities that will need to be positioned in such a way that all clients have access to them. There exist many variants of this problem. In the most studied however, the goal is to find optimal locations for the facilities in such a way that the maximum distance that a client needs to travel to reach its closest facility is minimized.

A classical instance of this problem involves wireless communication. Particularly, we are interested in placing a set of antennas, each capable of transmitting a message to all the clients within its range of transmission. In this model, we assume that clients are receivers capable of receiving messages from an antenna if they are within its transmission range. We assume unidirectional communication from the antennas to the clients. Since a larger transmission range requires more energy, we are interested in minimizing the transmission range of the antennas while providing service to all the clients.

To represent this problem geometrically, each wireless antenna and each client are assigned a position and in our setting are treated as points in the plane. In addition, an antenna $\mathcal{A}$ is given a range of transmission that defines a radius r around it. Thus, this antenna is modeled by a circle of radius r centered at the antenna's location. A receiver $\mathcal{R}$ receives the message from $\mathcal{A}$ if and only if $\mathcal{R}$ lies within this circle. We say that a client is *covered* if there exists an antenna whose circle contains it.

Therefore, our problem can be formulated as finding a set of antennas with minimum transmission range such that each client is covered by at least one antenna.

1.3.1 The k-center problem

To formalize the problem presented above, we proceed as follows. Let P be a set of n *sites* in the plane (clients) and let k be a positive integer. A *k-center* of P is a set A of k points (antennas), such that the maximum distance between a site of P and its closest point from A is minimized.

Let $r = \max\{d(p, a) : p \in P, a \in A\}$. Notice that if we require all the antennas to have the same range of transmission, then the minimum radius that the antennas of A need to have to cover all points of P is exactly r. Otherwise, regardless of where we place the antennas, at least one site from P will be uncovered.

When $k = 1$, the 1-center problem is equivalent to that of finding the minimum enclosing circle of P which represents a single antenna covering the entire set of clients. It is well known that such a circle can be computed in $O(n)$ time [70]. Moreover, the 1-center is known to lie on an edge of the farthest-point Voronoi diagram of P.

In many cases however, the antennas cannot be arbitrarily positioned since the terrain or landscape does not allow it. As a consequence, the antenna must be placed within a certain area. That is, the center of the minimum enclosing circle of P is constrained to lie within a given region of the plane. This gives rise to the problem studied in Chapter 6: Given two subsets Γ and Φ of the plane, find the minimum enclosing circle of Γ whose center is constrained to lie on Φ. We first study the case when Γ is a set of n sites and Φ is either a set of points, a set of segments (lines) or a simple polygon. We propose several algorithms, the first solves the problem when Φ is a set of m segments (or of m points) in expected $\Theta((n + m) \log \omega)$ time, where $\omega = \min\{n, m\}$. Surprisingly, when Φ is a simple m-gon, we can improve the expected running time to $\Theta(m + n \log n)$. Moreover, if Γ is the set of vertices of a convex n-gon and Φ is a simple m-gon, we can solve the problem in expected $\Theta(n + m)$ time. These results rely heavily on the computation of the farthest-point Voronoi diagram of the sites in Γ.

In addition, we provide matching lower bounds in the algebraic decision-tree model for all the algorithms presented in this work. While proving these results, we obtained a $\Omega(n \log m)$ lower bound for the following problem: Given two sets A and B in $\mathbb{R}$ such that $|A| = m \leq n = |B|$, decide if A is a subset of B. These results first appeared

in the proceedings of the 11th Latin American Theoretical INformatics Symposium (LATIN'14) [16].

1.3.2 The geodesic 1-center

In the previous section, we considered the k-center problem using the Euclidean metric to measure the distance between points. However, there exist other metrics that can be studied within the same framework. An example is the geodesic distance.

Let P be a simple polygon with n vertices. Given two points x, y in P, the *geodesic path* between them is the shortest path contained in P connecting x with y. The *geodesic distance* between x and y is the Euclidean length of the geodesic path connecting them.

A *geodesic k-center* of P is a set A of k points contained in P such that the maximum geodesic distance between any point in the polygon P and its closest point from A is minimized.

When $k = 1$, the solution to the 1-center problem is also known as the *geodesic center* of P. In 1989, Pollack, Sharir and Rote [83] showed an $O(n \log n)$-time algorithm to compute the geodesic center of P. Since then, it has been an open problem whether the geodesic center can be computed in linear time (indeed, this problem was explicitly posed by Pollack et al. [83] and later by Mitchell [73, Chapter 27]). In Chapter 7, we show how to compute the geodesic center of P in $O(n)$ time. While solving this problem, we develop new machinery to work with problems involving the geodesic distance that is of independent interest. A preliminary version of this result first appeared in the proceedings of the 31st International Symposium on Computational Geometry [3].

Chapter 2

Related Work: State of the Art

In this chapter we present an overview of the literature related to the problems studied in this thesis.

2.1 Intersection detection of convex polyhedra

Let P and Q be two polyhedra to be tested for intersection. Let $|P|$ and $|Q|$ denote the combinatorial complexities of P and Q, respectively, i.e., the number of faces of all dimensions of the polygon or polyhedron (vertices are 0-dimensional faces while edges are 1-dimensional faces). Let $n = |P| + |Q|$ denote the total complexity.

In the plane, Shamos [84] presented an optimal $\Theta(n)$-time algorithm to construct the intersection of a pair of convex polygons. Another linear-time algorithm was later presented by O'Rourke et al. [79]. In $\mathbb{R}^3$, Muller and Preparata [74] proposed an $O(n \log n)$ time algorithm to test whether two polyhedra intersect. Their algorithm has a second phase which computes the intersection of these polyhedra within the same running time using geometric duality. Dobkin and Kirpatrick [38] introduced a hierarchical data structure to represent a polyhedra that allows them to test if two polyhedra intersect in linear time. In a subsequent paper, Chazelle [29] presented an optimal linear time algorithm to compute the intersection of two polyhedra in $\mathbb{R}^3$.

Significant improvements in the running time are possible if we consider the effect or preprocessing. Recall that each object should be preprocessed separately which allows us to work with large families of objects and to introduce new objects without triggering a reconstruction of the whole structure.

Chazelle and Dobkin [30, 31] were the first to formally define and study this class of problems and provided an $O(\log |P| + \log |Q|)$-time algorithm to test the intersection of two convex polygons P and Q in the plane. An alternate solution was given by Dobkin and Kirkpatrick [37] with the same running time. Edelsbrunner [42] then

used that algorithm as a preprocessing phase to find the closest pair of points between two convex polygons, within the same running time. Dobkin and Souvaine [40] extended these algorithms to test the intersection of two convex planar regions with piecewise curved boundaries of bounded degree in logarithmic time. These separation algorithms rely on an involved case analysis to solve the problem.

In $\mathbb{R}^3$ (and in higher dimensions), the algorithms rely on new representations of the polyhedra as the traditional representation of the polyhedron is not sufficient to perform fast queries.

Chazelle and Dobkin [31] presented a method to preprocess a 3D polyhedron P in $O(|P|^2)$ time and space. Using this representation, they show how to test if two preprocessed polyhedra intersect in $O(\log^3 n)$ time. Dobkin and Kirkpatrick [37] unified and extended these results, by combining concepts from Kirkpatrick's point location algorithm [60], and Overmars and van Leeuwen's dynamic convex hull techniques [81]. Using this, they show how to detect if two independently preprocessed polyhedra intersect in $O(\log^2 n)$ time. Both methods represent a polyhedron P by storing parallel slices of P through each of its vertices, and thus require $O(|P|^2)$ time, although the space used could be reduced using persistent data structures.

In 1990, Dobkin and Kirkpatrick [39] proposed a fast query algorithm that uses the linear-space hierarchical representation of a polyhedron P defined in their previous article [38]. Using this structure, they show how to determine in $O(\log |P| \log |Q|)$ time if the polyhedra P and Q intersect. They achieve this by maintaining the closest pair between subsets of the polyhedra P and Q as the algorithms walks down the hierarchical representation. While a naive implementation of this algorithm could take time $\Omega(|P| + |Q|)$, O'Rourke [78, Chapter 7] describes in detail an implementation that avoids this issue and restores the $O(\log |P| \log |Q|)$ bound. In Section 4.4, we detail the specifics of this issue, and then we provide a simple modification of the hierarchical representation that offers an alternative solution.

2.2 Incremental Voronoi diagrams

Maintaining the geometric structure of a space decomposition, such as the Voronoi diagram, through insert, delete or kinetic events has received the attention of the research community [8, 26, 50, 52]. For the Voronoi diagram, we can find several results addressing this problem. For moving points, Guibas, Mitchell and Roos presented a method for maintaining the Voronoi diagram over time, by tracking the topological changes and modifying the structure of the diagram on each topological event in $O(\log n)$ time [52].

Dynamic maintenance of space decompositions has been proved difficult, since in the worst case, the insertion or deletion of a point into the Voronoi diagram or Delaunay triangulation can lead to a linear number of operations in order to maintain its explicit structure. Nevertheless, there has been some improvements in this area. To overcome this worst-case behavior, Aronov et al. [8] studied what happens in a specific case (points in convex position added in clockwise order) if we count the number of elementary link (add an edge) and cut (delete an edge) operations in the tree needed to maintain its combinatorial structure after an insertion. They show that, in this model, it is possible to reconfigure the tree after each site insertion while performing $O(\log n)$ links and cuts, amortized; however their proof is existential and no algorithm is provided to find those links and cuts. Pettie [82] shows both an alternate proof of that fact using forbidden 0-1 matrices and a matching lower bound.

One important application of Voronoi diagrams is to solve *nearest-neighbor* (or *farthest-neighbor*) queries: given a point in the plane, find the site nearest (or farthest) to this point. In the static case, this is done by preprocessing the (nearest or farthest point) Voronoi diagram to answer point-location queries in $O(\log n)$ time. Without the need to maintain the Voronoi diagram explicitly, the problem of nearest neighbor queries is a *decomposable search problem* and can be made semi-dynamic using the standard dynamization techniques of Bentley and Saxe [18]. The best incremental data structure supporting nearest-neighbor queries performs queries and insertions in $O(\log^2 n/\log\log n)$-time [33, 80]. Recently, Chan [26] developed a randomized data structure supporting nearest-neighbor queries in $O(\log^2 n)$-time, insertions in $O(\log^3 n)$ amortized time, and deletions in $O(\log^6 n)$ amortized time.

2.3 The k-center problem

Finding the k-center of a set of sites P is an extensively studied problem. Meggido and Supowit proved that the problem is NP-complete [72]. Moreover, Feder and Greene showed that it is NP-hard to approximate by a factor smaller that 1.82 [46]. However, there are several approximation algorithms with a factor of 2, one of them shown in [48]. In its discrete version, where a finite set M of possible locations for the points of A is given, the problem is also NP-complete [47]. Nevertheless for the discrete version there is a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ [20]. Note that, if k is a fixed constant, the discrete k-center problem can be solved in polynomial time, by checking the $O(n^k)$ possible subsets of M with k elements, and determining which one is the optimal. In fact when $k = 2$, there is an $O(n \log^2 n \log^2 \log n)$ time algorithm by Chan to solve the 2-center problem [25]. When $k = 1$, the continuous 1-center problem is equivalent to that of finding the minimum enclosing circle of P. To solve this problem, Megiddo proposed an optimal $O(n)$ time algorithm using prune and search [70].

In the process of solving the problem, Megiddo solved a restricted version of the problem where the center of the circle was constrained to lie on a given line. Extending these ideas, Hurtado, Sacristan and Toussaint presented an optimal $\Theta(n+m)$-time algorithm to find the minimum P-circle whose center is constrained to satisfy m linear inequalities [56], where a *P-circle* is a circle that encloses P. In other words, they solved the problem when the center is constrained to lie in a convex polygon of size m. Bose and Toussaint [21] generalized this result by restricting the center of the P-circle to lie in a simple m-gon Q. They proposed an $O((n+m)\log(n+m)+k)$-time algorithm, where k is the number of intersections of Q with the farthest-point Voronoi diagram of P. The dependency on k was later removed from the running time [22]. To improve this result, Bose et al. addressed the query problem and proposed an $O(n \log n)$-time preprocessing on P, which allows them to find the minimum P-circle with center on a given query line in $O(\log n)$-time [23]. In a generalization of this result, Barba [14] presented an algorithm to compute the minimum P-circle whose center is constrained to lie in a given disk.

Using these results, the minimum P-circle whose center is constrained to lie in a

simple polygon Q of size m can be computed in $O((n+m)\log n)$-time by testing each edge of the polygon.

2.4 The geodesic 1-center

Since the early 80s the problem of computing the geodesic center (and its counterpart, the geodesic diameter) has received a lot of attention from the Computational Geometry community. Chazelle [27] gave the first algorithm for computing the geodesic diameter (which runs in $O(n^2)$ time using linear space). Afterwards, Suri [88] reduced it to $O(n \log n)$-time without increasing the space constraints. Finally, Hershberger and Suri [55] presented a fast matrix-search technique, one application of which is a linear-time algorithm for computing the diameter.

The first algorithm for computing the geodesic center was given by Asano and Toussaint [9], and runs in $O(n^4 \log n)$-time. In 1989, Pollack, Sharir, and Rote [83] improved it to $O(n \log n)$ time. Since then, it has been an open problem whether the geodesic center can be computed in linear time (indeed, this problem was explicitly posed by Pollack et al. [83] and later by Mitchell [73, Chapter 27]).

Several variations of these two problems have been considered. Indeed, the same problem has been studied under different metrics such as the L_1 geodesic distance [13], the link distance [36, 59, 87] (where we look for the path with the minimum possible number of bends or *links*), or even rectilinear link distance [75, 76] (a variation of the link distance in which only isothetic segments are allowed). The diameter and center of a simple polygon for both the L_1 and rectilinear link metrics can be computed in linear time (whereas $O(n \log n)$ time is needed for the link distance).

Another natural extension is the computation of the diameter and center in polygonal domains (i.e., polygons with one or more holes). Polynomial-time algorithms are known for both the diameter [11] and center [12], although the running times are significantly larger (i.e., $O(n^{7.73})$ and $O(n^{12+\varepsilon})$, respectively).

Chapter 3

Preliminaries

In this chapter, we develop some of the tools that will be used throughout this thesis. In Section 3.1, we explore the concept of ε-nets which is closely related to the concept of cuttings. Both concepts revolve around the idea of approximating a general set by a collection of simpler subsets.

While the exact definition is more general as we will see below, intuitively an ε-net requires two things, a set of objects (for example a set P of points), called the *ground set*, and a set of *ranges* which are simply subsets of the objects (for example the family of all subsets of P that can be obtained by intersecting P with a disk). These two elements together define what is called a set system (formal definitions are given in the next section).

For a given $\varepsilon > 0$, an ε-net of the ground set in a given set system is intuitively a sample of the ground set that hits every big range. In our example above, if we have a set of points P in the plane, there exists a constant number of points $N \subset P$ such that any disk that contains at least $\varepsilon|P|$ points of P contains at least one point of N.

This notion turns to be extremely useful in the design of geometric algorithms, as many geometric set systems (like the points-disks mentioned above) have "small" ε-nets. This allows us to test predicates with respect only to the ε-nets and say something about the whole set.

In Section 3.2, we study an optimization technique that allows us to transform a decision algorithm into an optimization algorithm with the same running time by using randomization. Intuitively, by splitting the problem into subproblems and by having a fast decision algorithm, we can use a random process that allows us to prune many subproblems and find the optimal solution without an exhaustive search of the whole search space.

In Section 3.3, we introduce tools that allow us to prove lower bounds for several

problems throughout this work. We use the equivalence of some decision problems to membership problems, i.e., to decide if a point belongs or not to a given set. We then review some results that explain the relation between the hardness of the membership problem in a set W and the complexity of W. The combination of these tools allows us to prove lower bounds for many geometric problems by studying only the complexity of the solution space.

In Section 3.4, we study Voronoi diagrams. As mentioned above, these diagrams assign to each point in the plane its farthest (or nearest) site and encode this information as a decomposition of the plane into interior disjoint regions. We focus on the farthest-point Voronoi diagram and present several of its properties.

3.1 Set systems

Let X be a set of geometric objects and let $\mathcal{R}$ be a family of subsets of X. The pair $(X, \mathcal{R})$ is called a *set system* (also known as hypergraph or range space) on the *ground set* X.

A subset $T \subset X$ is called a *transversal* of $F \subseteq \mathcal{R}$ if it intersects each set of F. Intuitively, if the size of each set in F is large, then it should be easier to hit them with a transversal than if they are small. The notion of ε-nets works with this premise.

Given a set system $(X, \mathcal{R})$ and a value $\varepsilon \in (0, 1]$, *an ε-net* for $(X, \mathcal{R})$ is a set $N \subset X$ (not necessarily a set of $\mathcal{R}$) such that $N \cap R \neq \emptyset$ for each $R \in \mathcal{R}$ with $|R| \geq \varepsilon |X|$.

Therefore, we can think of ε-nets as transversals of all the sets of $\mathcal{R}$ with at least $\varepsilon|X|$ elements. As mentioned above, set systems with small ε-nets are of particular interest to us. Therefore, we would like to have sufficient conditions to guarantee the existence of small ε-nets.

3.1.1 VC-dimension and ε-nets

In order to guarantee the existence of small ε-nets, we use the *Vapnik-Chervonenkis dimension* of a set system, or simply *VC-dimension* for short, a parameter that intuitively measures how well a set system classifies the elements of the ground set.

Let $(X, \mathcal{R})$ be a set system. Given $A \subseteq X$, we say that A is *shattered* by $\mathcal{R}$ if each of the subsets of A can be obtained as the intersection of A with some set of $\mathcal{R}$. That is, for each $B \subseteq A$, there exists $R \in \mathcal{R}$ such that $B = A \cap R$. The *VC-dimension* of the set system $(X, \mathcal{R})$ is the supremum of the sizes of all finite shattered subsets of X. If arbitrarily large sets can be shattered, then we define the VC-dimension to be infinite.

The following result shows the relation between VC-dimension and ε-nets and is known as the *epsilon net* theorem.

Theorem 3.1.1 (Theorem 10.2.4 of [68])**.** *Let $\varepsilon > 0$. For any set system $(X, \mathcal{R})$ with VC-dimension $k \geq 2$, there exists an ε-net for $(X, \mathcal{R})$ of size at most $Ck\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}$, where C is an absolute constant.*

By Theorem 3.1.1, if a set system has VC-dimension bounded by a constant, then it has an ε-net of constant size. This is a rather surprising result as the size of the net is independent of the size of the ground set.

3.1.2 Geometric set systems

In this work, we are interested in set systems induced by geometric objects. Let X and $\mathcal{H}$ be two sets of arbitrary geometric objects. For each $H \in \mathcal{H}$, let $X_H = \{x \in X : x \cap H \neq \emptyset\}$. We say that $\mathcal{H}$ *induces* a set system $(X, \mathcal{R}_{\mathcal{H}})$ on X where

$$\mathcal{R}_{\mathcal{H}} = \{X_H : H \in \mathcal{H}\}.$$

For example, imagine that X is a set of points in the plane and that $\mathcal{H}$ is the set of all halfplanes in $\mathbb{R}^2$. We claim that the VC-dimension of the set system induced by $\mathcal{H}$ on X is 3. Indeed, if we have 3 points in general position, then each of its subsets can be cut off by a halfplane, and so it is shattered. Next, we claim that no set of 4 points can be shattered. In general position there are essentially only two different cases to be considered: 3 points on the boundary of the convex hull and the fourth in the interior of the triangle, or 4 points in convex position. Regardless of the case, we can always find a subset of the points (the point inside if the convex hull is a triangle, or 2 opposite corners of the convex hull in the case of size 4) such that any halfplane

containing them will contain a point from the complement of this subset. Therefore, no set of 4 points is shattered proving our claim.

In what follows, we describe some families of geometric objects that induce set systems of bounded VC-dimension on sets of points in $\mathbb{R}^d$. Roughly speaking, if each geometric object considered is "well-behaved", then they will induce set systems of bounded VC-dimension. Intuitively an object in $\mathbb{R}^d$ is "well-behaved" if it can be defined by a Boolean combination of polynomial inequalities.

More formally, a set $A \subset \mathbb{R}^d$ is called *semialgebraic* if there are polynomials $p_1, \ldots, p_r$ of d variables with real coefficients and a Boolean formula $\Phi(X_1, \ldots, X_r)$ (such as $X_1 \wedge (X_2 \vee X_3)$), where $X_1, \ldots, X_r$ are variables attaining values "true" or "false" such that

$$A = \left\{ x \in \mathbb{R}^d : \Phi(p_1(x) \geq 0, \ldots, p_r(x) \geq 0) \right\}.$$

Note that Φ may include negations, so sets where strict inequality or strict equality is needed are also semialgebraic. Now that we know what a "well-behaved" set looks like, we can study the VC-dimension of the set systems that they induce. The *description complexity* of a semialgebraic set is the maximum of the degree of the polynomials and the size of the Boolean function used to define it (number of boolean operators and variables).

Theorem 3.1.2 (Proposition 10.3.3 of [68])**.** *Let X be a set of semialgebraic sets of bounded description complexity in $\mathbb{R}^d$. Any family $\mathcal{R}$ of semialgebraic sets of bounded description complexity induces a set system on X of bounded VC-dimension.*

By Theorems 3.1.1 and 3.1.2, any set system induced by semialgebraic sets of bounded description complexity on semialgebraic sets of bounded complexity has an ε-net of size $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ for any $\varepsilon > 0$. While the proof of Theorem 3.1.1 is constructive and yields an efficient randomized algorithm to compute ε-nets, we are interested also in deterministic algorithms for the same task. In this direction, Matoušek [67] showed how to compute such an ε-net deterministically in the same running time as its randomized counter part.

Theorem 3.1.3. *Let $(X, \mathcal{R})$ be a set system of VC-dimension k and let $\varepsilon > 0$. An ε-net for $(X, \mathcal{R})$ can be computed in $O(n(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})^k)$ time.*

Note that the above algorithm yields a linear-time algorithm whenever ε is constant, which is the case in all geometric set systems considered in this work.

3.1.3 Cuttings

Inspired by ε-nets, cuttings are a more advanced version of the sampling technique applied to sets of hyperplanes.

A *generalized simplex* is the intersection of $d+1$ closed halfspaces. Given a set H of n hyperplanes in $\mathbb{R}^d$ and $r > 0$, a $\frac{1}{r}$-*cutting* of H is a collection of generalized simplices that cover $\mathbb{R}^d$ such that the interior of each generalized simplex intersects at most n/r hyperplanes. The following result bounds the size of a cutting and is known as the *Cutting* lemma.

Theorem 3.1.4. *[Theorem 6.5.3 of [68]] Given a set H of n hyperplanes in $\mathbb{R}^d$ and $1 < r \leq n$, there exists a $\frac{1}{r}$-cutting of H consisting of $O(r^d)$ generalized simplices. Moreover, such a cutting can be computed in $O(nr^{d-1})$ time.*

While the proof of Theorem 3.1.4 is quite involved, simple proofs exist at the expense of slightly worse bounds. We provide a simple proof for the planar case to give some intuition.

Let H be a set of n lines in the plane and let $1 < r \leq n$. Since generalized triangles (simplices in the plane) are semialgebraic sets of bounded description complexity, the set system induced by generalized triangles on H has bounded VC-dimension by Theorem 3.1.2. Therefore, we can compute an $\frac{1}{r}$-net $N \subset H$ with $|N| = O(r \log r)$ such that N intersects each generalized triangle that contains at least n/r lines of H. By Theorem 3.1.3 we can compute N in $O(n)$ time.

Notice that N defines an arrangement of lines that subdivides the plane into $O(r^2 \log^2 r)$ cells. We can triangulate the faces of this arrangement in time linear on the complexity of the arrangment to obtain $O(r^2 \log^2 r)$ generalized triangles that cover the plane. Moreover, the interior of each of these triangles intersects no line of N and hence, it intersects at most n/r lines of H. Therefore, we constructed a $1/r$-cutting of H in $O(n)$ time whose size is $O(r^2 \log^2 r)$.

The use of cuttings will allows us throughout this work to split different problems

into subproblems with smaller complexity that will be easier to handle. That is, they are the stepping stone of many divide-and-conquer algorithms.

3.2 Decision to Optimization techniques

Given a random permutation of a set of r numbers, a *record* is a number that is smaller than all the numbers to its left in the sequence. It is well known that the expected number of records in a random permutation of r numbers is given by the harmonic series $1 + 1/2 + \ldots + 1/r \leq \ln r + 1$.

Consider the following simple randomized algorithm to compute the minimum of r numbers given in an array A.

Algorithm RAND-MIN

1. randomly pick a permutation $(i_1, \ldots i_r)$ of $[r]$.
2. $t \leftarrow \infty$
3. for $k = 1, \ldots r$ do
4. if $A[i_k] < t$, then
5. $t \leftarrow A[i_k]$
6. return t

Notice that step 5 is executed whenever a record is found. Therefore, step 5 is executed $1 + 1/2 + \ldots + 1/r \leq \ln r + 1$ in expectation. Imagine that the numbers $A[1], \ldots A[r]$ are not known in advance, but are evaluated "on demand". If a decision $A[i] < t$ can be made in D time for a given t while an evaluation of $A[i]$ takes E time, then algorithm RAND-MIN runs in $O(Dr + E \log r)$ expected time, which is an improvement over the trivial algorithm running in $O(Er)$ time whenever $D \ll E$.

This observation suggests a way to solve an optimization problem: Split your problem into subproblems of smaller size and apply the above algorithm to find the minimum. Intuitively, not all the subproblems will be evaluated by the algorithm, but only tested with the decision algorithm. In what follows, we formalize this intuition.

Let Π be a problem space such that each problem P in Π has a real solution $w(P)$. Assuming that a constant-size problem can be computed in constant time, the next lemma describes how to find $w(P)$ using a decision algorithm.

Lemma 3.2.1. *[Lemma 2.1 of [24]] Let* Π *be a* problem space. *Let* $\alpha < 1$ *and* r *be constants, and let* $D(\cdot)$ *be an increasing function such that* $D(n) = \Omega(n)$. *Given any problem* $P \in \Pi$, *suppose that within* $D(|P|)$*time,*

(*i*) *given* $t \in \mathbb{R}$, *we can decide whether* $w(P) < t$, *and*

(*ii*) *we can construct* r *subproblems* $P_1, \ldots P_r \in \Pi$, *each of size at most* $\lceil \alpha |P| \rceil$, *so that*

$$w(P) = \min\{w(P_1), \ldots, w(P_r)\}.$$

Then for any problem $P \in \Pi$, *we can compute the solution* $w(P)$ *in* $O(D(|P|))$ *time.*

Proof. We compute $w(P)$ by applying algorithm RAND-MIN to the (unknown) numbers $w(P_1), \ldots, w(P_r)$. Recall that deciding if $w(P_i) < t$ takes $D(|P_i|)$ time. Evaluating $w(P_i)$ is done recursively, unless $|P_i|$ drops below certain constant. This procedure not only computes $w(P)$ but identifies a constant-size subproblem attaining this minimum.

Let $T(P)$ be a random variable corresponding to the time needed to compute $w(P)$ by this procedure. Let $N(P_i)$ be an indicator random variable, having value 1 if and only if $w(P_i)$ is evaluated. Therefore, we have

$$T(P) = \sum_{i=1}^{r} N(P_i)T(P_i) + O(rD(|P|)).$$

As noted earlier, the expected number of evaluations by RAND-MIN is

$$E[\sum_{i=1}^{r} N(P_i)] \leq \ln r + 1.$$

Define $T(n) = \max_{|P| \leq n} E[T(P)]$. Since $N(P_i)$ and $T(P_i)$ are independent variables, we have

$$\begin{aligned} E[T(P)] &= \textstyle\sum_{i=1}^{r} E[N(P_i)]E[T(P_i)] + O(rD(|P|)) \\ &\leq (\ln r + 1)T(\lceil \alpha |P| \rceil) + O(rD(|P|)). \end{aligned}$$

This implies the following recurrence:

$$T(n) = (\ln r + 1)T(\lceil \alpha |P| \rceil) + O(D(n)).$$

If we assume that $(\ln r + 1)\alpha < 1$, then by induction we can show easily that $T(n) \leq C \cdot D(n)$ form some constant C depending on α and r.

To ensure that $(\ln r + 1)\alpha < 1$, we can compress ℓ levels of the recursion into one before applying algorithm RAND-MIN, where ℓ is a sufficiently large constant. That is, we subdivide the subproblems recursively ℓ times. In this way, the number of subproblems increases from r to r^ℓ and their size decreases from α to α^ℓ. To conclude the proof, simply note that $\lim_{\ell\to\infty}(\ln r^\ell + 1)\alpha^\ell = 0$, i.e., $(\ln r^\ell + 1)\alpha^\ell < 1$ for some sufficiently large constant ℓ. □

3.3 Lower bound for membership problems

In Section 6.6, we use the the tool presented in this section to describe lower bounds for several decision problems.

Let $W \subset \mathbb{R}^m$ be any set. The *membership problem* for W is the following: Given a point $x \in \mathbb{R}^m$, determine if $x \in W$.

Roughly speaking, we will show that many decision problems can, in fact, be seen as membership problems in high-dimensional spaces. Therefore, lower bounds for membership problems will translate to lower bounds for our decision algorithms.

We are interested in obtaining lower bounds on algorithms for solving the membership problem for W that allow both arithmetic operations and tests. Thus, a step in such an algorithm can evaluate a polynomial of degree d (for some constant d) and compare the result to zero (i..e, $>, <, =$), and branch according to the outcome of the test. We say that an algorithm of this type is in the *algebraic decision-tree model* (for further information see [17]).

In this model, the following result shows the relation between the structure of W and the "hardness" of the membership problem.

Theorem 3.3.1. *[Restatement of Theorem 3 of [17]] Let $W \subset \mathbb{R}^m$ be any set, and let A be an algorithm in the algebraic decision-tree model that solves the membership problem for W. If N is the number of disjoint connected components of W, and h is the worst-case running time of A, then $h = \Omega(\log N)$.*

Intuitively, the more complex W is, the harder it is to decide if a point lies in it.

In recent work, people have looked at other complexity measures of W, such as the Betty numbers and other topological features.

3.4 Farthest-point Voronoi diagrams

Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of n points in general position called *sites*. The farthest-point Voronoi diagram of P is a division of the plane into n *Voronoi regions* $R(p_1), \ldots, R(p_n)$, one for each site in P, with the property that a point lies in $R(p_i)$ only if x is farther from p_i than from any other point in P.

Let $\Pi_{i,j}$ be the closed halfplane that contains p_j bounded by the bisector of p_i and p_j. Then $R(p_i) = \bigcap_{j \neq i} \Pi_{i,j}$. Notice that $R(p_i)$ is a possibly empty convex polygon whose boundary is composed of edges, each of which is contained in the bisector of a pair of point of P. Formally, the *farthest-point Voronoi diagram* of P is defined as the division of the plane induced by the family of Voronoi regions $\{R(p_1), \ldots, R(p_2)\}$.

3.4.1 Properties of the farthest-point Voronoi diagram

In this section, we show several properties of the farthest-point Voronoi diagram. It is not hard to see that for any point in the plane, its farthest point in the set P must be a point that lies on the convex hull of P. Therefore, a point that lies inside the convex hull cannot have a cell in the farthest-point Voronoi diagram.

Lemma 3.4.1. *For each $p \in P$, the Voronoi region p is not empty if and only if p lies on the boundary of the convex hull of P.*

By Lemma 3.4.1, we can assume without loss of generality that the points of P are in convex position when dealing with the farthest-point Voronoi diagram.

Observation 3.4.2. *Let p be a site of P and let x be a point in the plane. If the circle with radius $d(x, p)$ centered on x encloses all sites of P, then x lies in $R(p)$.*

The above observations follows directly from the definition of farthest neighbor and Voronoi region.

Lemma 3.4.3. *If p is a site of P lying on the convex hull of P, then the region $R(p)$ is a non-empty unbounded convex polygon.*

Proof. Let x be a point on the boundary of $R(p)$ and let $\ell_{x,p}$ be the line passing through x and p. Consider the ray λ contained in $\ell_{x,p}$ shooting from x and avoiding p. Since $x \in R(p)$, the circle C_x centered on x with radius $d(x,p)$ encloses every site of P. Let y be a point on λ and let C_y be the circle centered on y with radius $d(y,p)$.

Since C_y contains C_x, we conclude that C_y encloses also every site of P. By Observation 3.4.2, we know that $y \in R(p)$ and hence $R(p)$ is unbounded. □

Because $R(p)$ is an unbounded convex polygon, we infer that its boundary consists of some bounded edges and two (unbounded) rays. We refer to these unbounded rays as the *unbounded edges* of $R(p)$.

Notice that each Voronoi region is a polygon defined by a set of vertices and edges. Thus, we can also view the Voronoi diagram as a graph. However, this graph, denoted by V_P, contains a linear number of unbounded edges. To solve this and obtain a simple graph, we add a leaf along each of the unbounded edges of V_p sufficiently far. In this way, we obtain a geometric plane graph that we denote by $\mathcal{V}(P)$; see Figure 3.1.

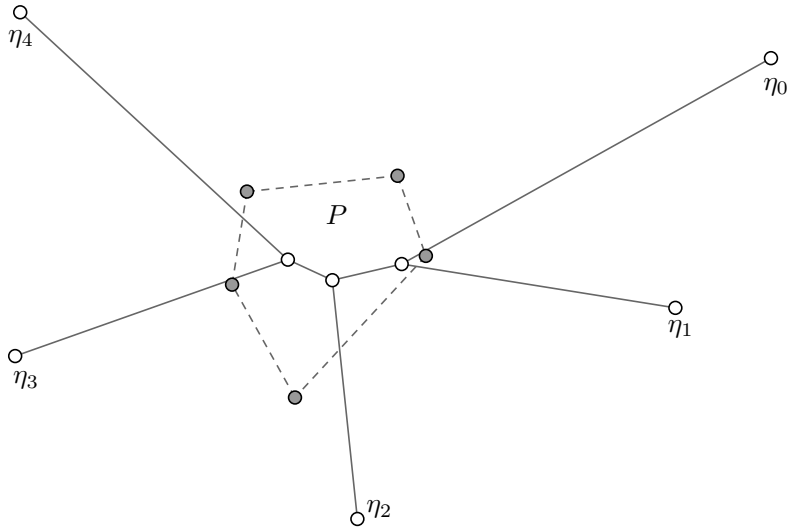

Figure 3.1: The farthest-point Voronoi diagram seen as a graph $\mathcal{V}(P)$ by adding leaves sufficiently far along each of the unbounded edges.

Lemma 3.4.4. *The vertices and edges of $\mathcal{V}(P)$ define a connected acyclic geometric plane graph.*

Proof. Since the farthest-point Voronoi diagram is a subdivision of the plane, we conclude that the graph formed by its vertices and edges is connected. Moreover, it cannot have cycles since it would imply the existence of a bounded region contradicting Lemma 3.4.3. □

Lemma 3.4.4 implies that $\mathcal{V}(P)$ is a tree formed by the boundaries of the Voronoi regions of all sites of P.

Lemma 3.4.5. *Let P be a set of n sites in the plane in convex position such that no 4 are co-circular. Then, the tree $\mathcal{V}(P)$ has exactly $2n-3$ edges and $n-2$ vertices.*

Proof. Let m_e, m_v and m_f be the number of edges, vertices and faces of $\mathcal{V}(P)$, respectively. Let $\mathcal{V}'(P)$ be the graph obtained from $\mathcal{V}(P)$ by identifying each leaf into a single vertex v_∞. Thus, the graph $\mathcal{V}'(P)$ is embeddable in the sphere.

Notice that $m_f = n$, since there is a one-to-one correspondence between the sites of P and the Voronoi regions of $\mathcal{V}(P)$. Using Euler's characteristic, we get the following:

$$(m_v + 1) - m_e + m_f = 2.$$

By the general position assumption, each vertex of $\mathcal{V}'(P)$ other than v_∞ has degree three. Moreover, v_∞ has degree n. Since $\sum_{v \in V} \delta(v) = 2|E|$, where $\delta(*)$ represents de degree of a node and E is the set of edges of $\mathcal{V}'(P)$, we get that:

$$3m_v + n = \sum_{v \in \mathcal{V}'(P)} \delta(v) = 2m_e.$$

Solving a simple system of equations, we conclude that

$$m_e = 2n - 3.$$

Since the number of edges of $\mathcal{V}'(P)$ is equal to the number of edges of $\mathcal{V}(P)$, our result follows. □

Notice that whenever we cannot guarantee general position (no four points cocircular), then we can only show that $m_e \leq 2n - 3$.

Given a set P of n sites, we are interested in computing $\mathcal{V}(P)$ efficiently. To this end, we use the algorithm developed by Aggarwal et al. [2] which runs in $O(n)$ time, provided that we know the combinatorial structure of the convex hull of P. If the convex hull of P is not known, then additional $O(n \log n)$ time is needed.

Theorem 3.4.6. *Given a set P of n sites, the farthest-point Voronoi diagram of P can be computed in $\Theta(n \log n)$ time. Moreover, if the convex hull of P is given, then $\Theta(n)$ time suffices to compute the farthest-point Voronoi diagram of P.*

Part I

Intersection Detection

Chapter 4

Intersection Detection of Convex Polyhedra

In this chapter, we consider the effect of preprocessing on the complexity of intersection-detection problems. Recall that each object should be preprocessed separately which allows us to work with large families of objects and to introduce new objects without triggering a reconstruction of the whole structure. Throughout this chapter, we assume that all polyhedra are convex.

Let P and Q be two polyhedra to be tested for intersection. Let $|P|$ and $|Q|$ denote the combinatorial complexities of P and Q, respectively, i.e., the number of faces of all dimensions of the polygon or polyhedra (vertices are 0-dimensional faces while edges are 1-dimensional faces). Let $n = |P| + |Q|$ denote the total complexity.

Chazelle and Dobkin [30, 31], and Dobkin and Kirkpatrick [37] were the first to study this class of problems. They provided algorithms running in $O(\log|P|+\log|Q|)$ time to test whether or not two convex polygons P and Q in the plane intersect. In Section 4.2, we show an alternate (and arguably simpler) algorithm to determine if two convex polygons P and Q intersect in $O(\log|P| + \log|Q|)$ time.

In all these 2D algorithms, preprocessing is unnecessary if the polygon is represented by an array with the vertices of the polygon stored in sorted order along its boundary. In 3D-space (and in higher dimensions) however, the need for preprocessing is more evident as the traditional representation of the polyhedron is not sufficient to perform fast queries.

Whether the intersection of two preprocessed polyhedra P and Q can be tested in $O(\log|P|+\log|Q|)$ time has been an open question posed in several places [30, 37, 49]. Together with this question, it was also asked if it is possible to extend these result to higher dimensions, i.e., to independently preprocess two polyhedra in $\mathbb{R}^d$ such that their intersection could be tested in $O(\log n)$ time.

These running times are best possible as, even in the plane, testing if a point

intersects a regular m-gon M has a lower bound of $\Omega(\log m)$ in the algebraic decision-tree model. To prove this bound, consider the circumcircle of M and reduce its radius by some $\varepsilon > 0$ to obtain a circle C. By choosing ε sufficiently small, C intersects P in m disjoint connected components. Therefore, a query point restricted to lie in C intersects P if and only if it lies in one of these m components. Thus, by Theorem 3.3.1, there is an $\Omega(\log m)$ lower bound for the running time of any algorithm that decides if a query point intersects M.

In this chapter, we match this lower bound by showing how to independently preprocess polyhedra P and Q in any bounded dimension such that their intersection can be tested in $O(\log n)$ time.

We further show how to preprocess a polyhedron in $\mathbb{R}^3$ in linear time to obtain a linear-space representation. In Section 4.5 we provide an algorithm that, given any translation and rotation of two preprocessed polyhedra P and Q in $\mathbb{R}^3$, tests if they intersect in $O(\log |P| + \log |Q|)$ time, where $|P|$ represents the combinatorial complexity of P. In Section 4.6 we generalize our results to any constant dimension d and show a representation that allows to test if two polyhedra P and Q in $\mathbb{R}^d$ (rotated and translated) intersect in $O(\log |P| + \log |Q|)$ time. The space used by the representation of a polyhedron P is then $O(|P|^{\lfloor d/2 \rfloor + \varepsilon})$ for any $\varepsilon > 0$ and $d \geq 4$. This increase in the space requirements for $d \geq 4$ is not unexpected as the problem studied here is at least as hard as performing halfspace emptiness queries for a set of m points in $\mathbb{R}^d$. For this problem, the best known log-query data structures use roughly $O(m^{\lfloor d/2 \rfloor})$ space [69], and super-linear space lower bounds are known for $d \geq 5$ [45].

4.1 Outline

To guide the reader, we give a rough sketch of the algorithm presented in this chapter, which is illustrated in Figure 4.1.

We use two types of hierarchical structures of logarithmic depth to represent a polyhedron. An *internal hierarchy* is obtained by recursively removing "large" sets of the vertices of the polyhedron and taking the convex hull of the remaining vertices. Since a polyhedron can also be seen as the intersection of halfspaces, an *external hierarchy* can be obtained by recursively removing "large" sets of halfspaces

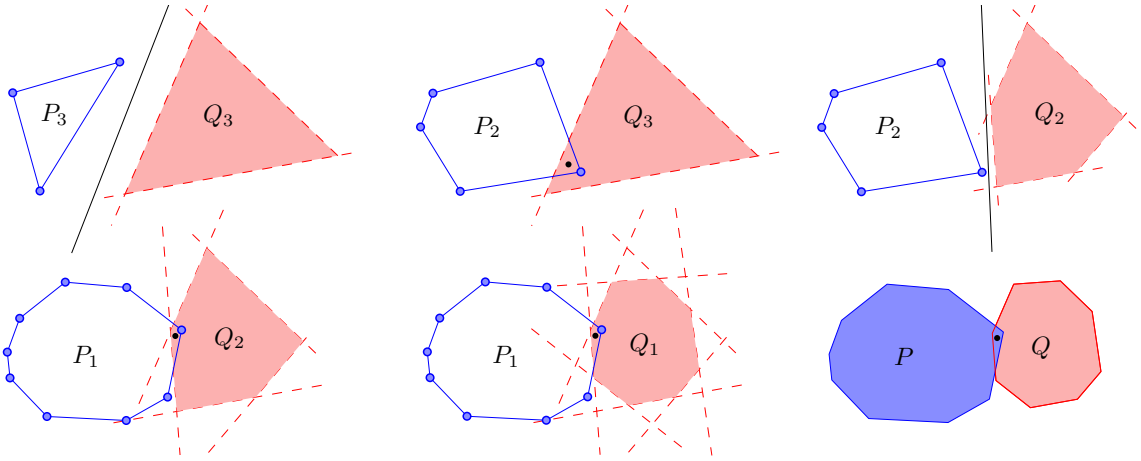

Figure 4.1: In each step, the algorithm moves down in either the internal hierarchy of P, say P_1, P_2, P_3, or the external hierarchy of Q, say Q_1, Q_2, Q_3. Throughout, the polyhedron in the hierarchy of P grows while the polyhedron in the hierarchy of Q shrinks. A separating (black) line or an intersection (black) point is maintained in each step.

and taking the intersection of the remaining halfspaces. (A similar structure was introduced by Dobkin et al. [41] to test how "deeply" two polyhedra intersect). Thus, at the top of these hierarchies we store constant-size polyhedra, while at the bottom the full polyhedra are stored.

To test two preprocessed polyhedra P and Q for intersection, we use an inner hierarchy for P and an external hierarchy for Q. Starting at the top, we make our way down by moving one step at a time in either hierarchy. We move down in the hierarchy of P by adding more vertices (which increases its volume), while we move down in the hierarchy of Q by adding halfspace constraints (which decreases its volume). Thus, in our algorithm one polyhedron grows while the other shrinks, whereas previous approaches grew both polyhedra simultaneously. Additionally, we maintain either a separating plane or an intersection point while moving down in these hierarchies. This allows us to determine the intersection of the polyhedra after reaching the bottom of the hierarchies.

The algorithm described in Section 4.2 directly implements this idea to test the intersection of two convex polygons in the plane.

For technical reasons, to capture this intuition in higher dimensions we make use of the polar transformation (see Section 4.3). This operation maps a polyhedron in a

primal space into the dual polyhedron in a polar space. Moreover, this transformation maps the inner hierarchy of a polyhedron into the external hierarchy of its dual counterpart. Consequently, being able to construct inner hierarchies is sufficient. To test the intersection of two preprocessed polyhedra, our algorithm switches back and forth between a primal and a polar space while moving down in the hierarchies of these polyhedra.

4.2 Algorithm in the plane

Let P and Q be two convex polygons in the plane with n and m vertices, respectively. We assume that a convex polygon is given as an array with the sequence of its vertices sorted in clockwise order along its boundary. Let $V(P)$ and $E(P)$ be the set of vertices and edges of P, respectively. Let ∂P denote the boundary of P. Analogous definitions apply for Q. As a warm-up, we describe an algorithm to determine if P and Q intersect whose running time is $O(\log n + \log m)$. Even though algorithms with these running time already exist in the literature, they require an involved case analysis whereas our approach avoids them. Moreover, it provides some intuition for the higher-dimension algorithms presented in subsequent sections.

For each edge $e \in E(Q)$, its *supporting halfplane* is the halfplane containing Q supported by the line extending e. Given a subset of edges $F \subseteq E(Q)$, the *edge hull* of F is the intersection of the supporting halfplanes of each of the edges in F. Throughout the algorithm, we consider a triangle $\mathcal{T}_P$ being the convex hull of three vertices of P and a triangle (possibly unbounded) $\mathcal{T}_Q$ defined as the edge hull of three edges of Q; see Figure 4.2 for an illustration. Notice that $\mathcal{T}_P \subseteq P$ while $Q \subseteq \mathcal{T}_Q$.

Intuitively, in each round the algorithm compares $\mathcal{T}_P$ and $\mathcal{T}_Q$ for intersection and, depending on the output, prunes a fraction of either the vertices of P or of the edges of Q. Then, the triangles $\mathcal{T}_P$ and $\mathcal{T}_Q$ are redefined should there be a subsequent round of the algorithm.

Let $V^*(P)$ and $E^*(Q)$ respectively be the sets of vertices and edges of P and Q remaining after the pruning steps performed so far by the algorithm. Initially, $V^*(P) = V(P)$ while $E^*(Q) = E(Q)$. After each pruning step, we maintain the *correctness invariant* which states that an intersection between P and Q can be

computed with the remaining vertices and edges after the pruning. That is, P and Q intersect if and only if $\textsc{ch}(V^*(P))$ intersects an edge of $E^*(Q)$, where $\textsc{ch}(V^*(P))$ denotes the convex hull of $V^*(P)$.

For a given polygonal chain, its *vertex-median* is a vertex whose removal splits this chain into two pieces that differ by at most one vertex. In the same way, the *edge-median* of this chain is an edge whose removal splits the chain into two parts that differ by at most one edge.

4.2.1 The 2D algorithm

To begin with, define $\mathcal{T}_P$ as the convex hull of three vertices whose removal splits the boundary of P into three chains, each with at most $\lceil (n-3)/3 \rceil$ vertices. In a similar way, define $\mathcal{T}_Q$ as the edge hull of three edges of Q that split its boundary into three polygonal chains each with at most $\lceil (m-3)/3 \rceil$ edges; see Figure 4.2.

A line *separates* two convex polygons if they lie in opposite closed halfplanes supported by this line. After each round of the algorithm, we maintain one of the two following invariants: The *separation invariant* states that there is a line ℓ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ such that ℓ is tangent to $\mathcal{T}_P$ at a vertex v. The *intersection invariant* states that there is a point in the intersection between $\mathcal{T}_P$ and $\mathcal{T}_Q$. Note that at least one among the separation and the intersection invariant must hold, and they only hold at the same time when $\mathcal{T}_P$ is tangent to $\mathcal{T}_Q$. The algorithm performs two different tasks depending on which of the two invariants holds (if both hold, we choose a task arbitrarily).

4.2.2 Separation invariant

If the separation invariant holds, then there is a line ℓ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ such that ℓ is tangent to $\mathcal{T}_P$ at a vertex v. Let ℓ^- be the closed halfplane supported by ℓ that contains $\mathcal{T}_P$ and let ℓ^+ be its complement.

Consider the two neighbors n_v and n'_v of v along the boundary of P. Because P is a convex polygon, if both n_v and n'_v lie in ℓ^-, then we are done as ℓ separates P from $\mathcal{T}_Q \supseteq Q$. Otherwise, by the convexity of P, either n_v or n'_v lies in ℓ^+ but not both. Assume without loss of generality that $n_v \in \ell^+$ and notice that the removal of the

vertices of $\mathcal{T}_P$ split ∂P into three polygonal chains. In this case, we know that only one of these chains, say c_v, intersects ℓ^+. Moreover, we know that v is an endpoint of c_v and we denote its other endpoint by u.

Because Q is contained in ℓ^+, only the vertices in c_v can define an intersection with Q. Therefore, we prune $V^*(P)$ by removing every vertex of P that does not lie on c_v and maintain the correctness invariant. We redefine $\mathcal{T}_P$ as the convex hull of v, u and the vertex-median of c_v. With the new $\mathcal{T}_P$, we can test in $O(1)$ time if $\mathcal{T}_P$ and $\mathcal{T}_Q$ intersect. If they do not, then we can compute a new line that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ and preserve the separation invariant. Otherwise, if $\mathcal{T}_P$ and $\mathcal{T}_Q$ intersect, then we establish the intersection invariant and proceed to the next round of the algorithm.

4.2.3 Intersection invariant

If the intersection invariant holds, then $\mathcal{T}_P \cap \mathcal{T}_Q \neq \emptyset$. In this case, let e_1, e_2 and e_3 be the three edges whose edge hull defines $\mathcal{T}_Q$. Notice that if $\mathcal{T}_P \subseteq P$ intersects $\textsc{ch}(e_1, e_2, e_3) \subseteq Q$, then P and Q intersect and the algorithm finishes. Otherwise, there are three disjoint connected components in $\mathcal{T}_Q \setminus \textsc{ch}(e_1, e_2, e_3)$ and $\mathcal{T}_P$ intersects exactly one of them; see Figure 4.2. Assume without loss of generality that $\mathcal{T}_P$ intersects the component bounded by the lines extending e_1 and e_2 and let x be a point on the boundary of $\mathcal{T}_Q$ in this intersection. Let C be the polygonal chain that connects e_1 with e_2 along ∂Q such that C passes through e_3. We claim that to test if P and Q intersect, we need only to consider the edges on $\partial Q \setminus C$. To prove this claim, notice that if P intersects C at a point y, then the edge xy is contained in Q. Because x and y lie in two disjoint connected components of $\mathcal{T}_Q \setminus \textsc{ch}(e_1, e_2, e_3)$, the edge xy also intersects ∂Q at another point lying on $\partial Q \setminus C$. Therefore, an intersection between P and Q will still be identified even if we ignore every edge on C. That is, P and Q intersect if and only if P and $\partial Q \setminus C$ intersect. Thus, we can prune $E^*(Q)$ by removing every edge along C while preserving the correctness invariant. After the pruning step, we redefine $\mathcal{T}_Q$ as the edge hull of e_1, e_2 and the edge-median of the remaining edges of $E(Q)$ after the pruning.

If $\mathcal{T}_P$ intersects $\mathcal{T}_Q$ after being redefined, then the intersection invariant is preserved and we proceed to the next round of the algorithm. Otherwise, if $\mathcal{T}_P$ does not intersect

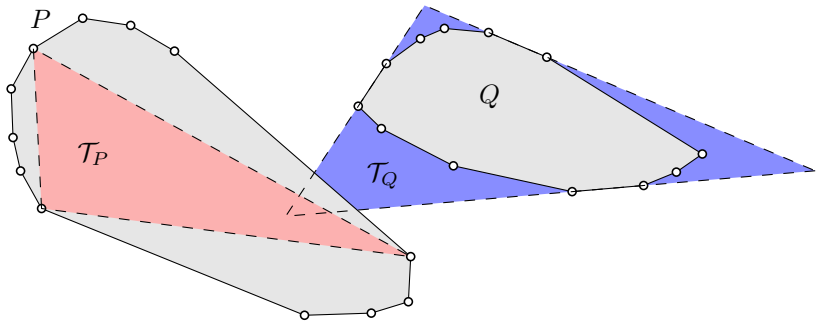

Figure 4.2: Two convex polygons P and Q and the triangles $\mathcal{T}_P$ and $\mathcal{T}_Q$ such that $\mathcal{T}_Q \subseteq P$ and $Q \subseteq \mathcal{T}_Q$. Moreover, $\mathcal{T}_Q \setminus Q$ consists of three connected components.

$\mathcal{T}_Q$, then we we can compute in $O(1)$ time a line ℓ tangent to $\mathcal{T}_P$ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$. That is, the separation invariant is reestablished should there be a subsequent round of the algorithm.

Theorem 4.2.1. *Let P and Q be two convex polygons with n and m vertices in the plane, respectively. The 2D-algorithm decides if P and Q intersect in $O(\log n + \log m)$ time. Moreover, it reports either a point in the intersection or a separating plane.*

Proof. Each time we redefine $\mathcal{T}_P$, we take three vertices that split the remaining vertices of $V^*(P)$ into two chains of roughly equal length along ∂P. Therefore, after each round where the separation invariant holds, we prune a constant fraction of the vertices of $V^*(P)$. That is, the separation invariant step of the algorithm can be performed at most $O(\log n)$ times.

Each time $\mathcal{T}_Q$ is redefined, we take three edges that split the remaining edges along the boundary of Q into equal-size pieces. Thus, we prune a constant fraction of the edges of $E^*(Q)$ after each round where the intersection invariant holds. Hence, this can be done at most $O(\log m)$ times before being left with only three edges of Q. Furthermore, the correctness invariant is maintained after each of the pruning steps.

Thus, if the algorithm does not find a separating line or an intersection point, then after $O(\log n + \log m)$ steps, $\mathcal{T}_P$ consists of the only three vertices left in $V^*(P)$ while $\mathcal{T}_Q$ consist of the only three edges remaining from $E^*(Q)$. If e_1, e_2 and e_3 are the edges whose edge hull defines $\mathcal{T}_Q$, then by the correctness invariant we know that P and Q intersect if and only if $\mathcal{T}_P$ intersects either e_1, e_2 or e_3. Consequently, we

can test them for intersection in $O(1)$ time and determine if P and Q intersect. □

4.3 The polar transformation

Let $\mathbb{0}$ be the *origin* of $\mathbb{R}^d$, i.e., the point with d coordinates equal to zero. Throughout this chapter, a *hyperplane* h is a $(d-1)$-dimensional affine space in $\mathbb{R}^d$ such that for some $z \in \mathbb{R}^d$, $h = \{x \in \mathbb{R}^d : \langle z, x\rangle = 1\}$, where $\langle *, *\rangle$ represents the inner product of Euclidean spaces. Therefore, in this chapter a hyperplane does not contain the origin. A *halfspace* is the closure of either of the two parts into which a hyperplane divides $\mathbb{R}^d$, i.e., a halfspace contains the hyperplane defining its boundary.

Given a point $x \in \mathbb{R}^d$, we define its *polar* to be the hyperplane $\rho(x) = \{y \in \mathbb{R}^d : \langle x, y\rangle = 1\}$. Given a hyperplane h in $\mathbb{R}^d$, we define its *polar* $\rho(h)$ as the point $z \in \mathbb{R}^d$ such that $h = \{y \in \mathbb{R}^d : \langle z, y\rangle = 1\}$. Let $\rho_{\mathbb{0}}(x) = \{y \in \mathbb{R}^d : \langle x, y\rangle \leq 1\}$ and $\rho_{\infty}(x) = \{y \in \mathbb{R}^d : \langle x, y\rangle \geq 1\}$ be the two halfspaces supported by $\rho(x)$, where $\mathbb{0} \in \rho_{\mathbb{0}}(x)$ while $\mathbb{0} \notin \rho_{\infty}(x)$. In the same way, $h_{\mathbb{0}}$ and h_{∞} denote the halfspaces supported by h such that $\mathbb{0} \in h_{\mathbb{0}}$ while $\mathbb{0} \notin h_{\infty}$.

Note that the polar of a point $x \in \mathbb{R}^d$ is a hyperplane whose polar is equal to x, i.e., the polar operation is inverse of itself (for more information on this transformation see Section 2.3 of [92]). Given a set of points (or hyperplanes), its *polar set* is the set containing the polar of each of its elements. The following result is illustrated in Figure 4.3(a).

Lemma 4.3.1. *Let x and h be a point and a hyperplane in $\mathbb{R}^d$, respectively. Then, $x \in h_{\mathbb{0}}$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x)$. Also, $x \in h_{\infty}$ if and only if $\rho(h) \in \rho_{\infty}(x)$. Moreover, $x \in h$ if and only if $\rho(h) \in \rho(x)$.*

Proof. Recall that $h_{\mathbb{0}} = \{y \in \mathbb{R}^d : \langle y, \rho(h)\rangle \leq 1\}$. Then, $x \in h_{\mathbb{0}}$ if and only if $\langle x, \rho(h)\rangle \leq 1$. Furthermore, $\langle x, \rho(h)\rangle \leq 1$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x) = \{y \in \mathbb{R}^d : \langle y, x\rangle \leq 1\}$. That is, $x \in h_{\mathbb{0}}$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x)$. Analogous proofs hold for the other statements. □

A polyhedron is the non-empty intersection of a finite set of halfspaces. Given a set of hyperplanes S in $\mathbb{R}^d$, let $\text{PH}_{\infty}[S] = \cap_{h \in S} h_{\infty}$ and $\text{PH}_{\mathbb{0}}[S] = \cap_{h \in S} h_{\mathbb{0}}$ be two polyhedra defined by S (the former may be empty). Let $P \subset \mathbb{R}^d$ be a polyhedron.

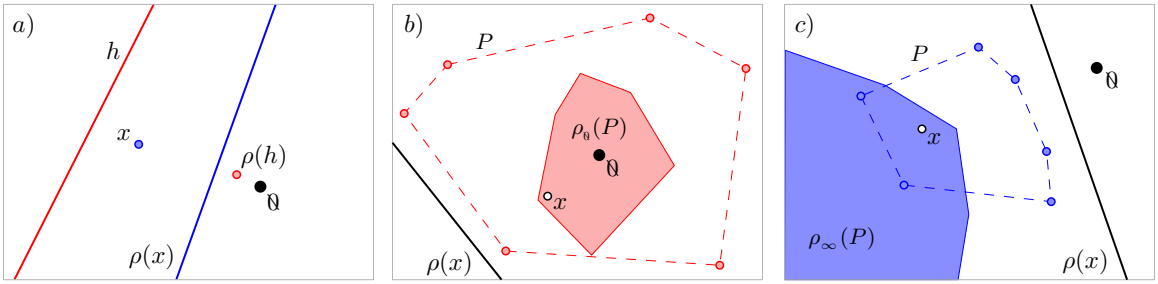

Figure 4.3: *a*) The situation described in Lemma 4.3.1. *b*) A polygon P containing the origin and its polarization $\rho_{\mathbb{0}}(P)$. The first statement of Lemma 4.3.3 is depicted. *c*) A polygon P that does not contains the origin and its polarization $\rho_{\infty}(P)$. The second statement of Lemma 4.3.3 is also depicted.

Let $V(P)$ denote the set of vertices of P and let $S(P)$ be the set of hyperplanes that extend the $(d-1)$-dimensional faces of P. Therefore, if P is bounded, then it can be seen as the convex hull of $V(P)$, denoted by $\text{CH}(V(P))$. Moreover, if P contains the origin, then P can be also seen as $\text{PH}_{\mathbb{0}}[S(P)]$.

To *polarize* P, let $\mathbb{S}(P)$ be the polar set of $V(P)$, i.e., the set of hyperplanes polars to the vertices of P. Therefore, we can think of $\text{PH}_{\mathbb{0}}[\mathbb{S}(P)]$ and $\text{PH}_{\infty}[\mathbb{S}(P)]$ as the possible polarizations of P. For ease of notation, we let $\rho_{\mathbb{0}}(P)$ and $\rho_{\infty}(P)$ denote the polyhedra $\text{PH}_{\mathbb{0}}[\mathbb{S}(P)]$ and $\text{PH}_{\infty}[\mathbb{S}(P)]$, respectively. Note that P contains the origin if and only if $\rho_{\infty}(P) = \emptyset$ and $\rho_{\mathbb{0}}(P)$ is bounded.

Lemma 4.3.2. *(Clause (v) of Theorem 2.11 of [92]) Let P be a polyhedron in $\mathbb{R}^d$ such that $\mathbb{0} \in P$. Then, $\rho_{\mathbb{0}}(\rho_{\mathbb{0}}(P)) = P$.*

As a consequence of Lemma 4.3.1 we obtain the following result depicted in Figures 4.3(*b*) and 4.3(*c*).

Lemma 4.3.3. *Let P be a polyhedron in $\mathbb{R}^d$ and let $x \in \mathbb{R}^d$. Then, $x \in \rho_{\mathbb{0}}(P)$ if and only if $P \subseteq \rho_{\mathbb{0}}(x)$. Moreover, $x \in \rho_{\infty}(P)$ if and only if $P \subseteq \rho_{\infty}(x)$.*

Proof. Let x be a point in $\rho_{\mathbb{0}}(P)$. Notice that for every hyperplane $s \in \mathbb{S}(P)$, $x \in s_{\mathbb{0}}$. Therefore, by Lemma 4.3.1 we know that the vertex $\rho(s) \in V(P)$ lies in $\rho_{\mathbb{0}}(x)$. Consequently, every vertex of P lies in $\rho_{\mathbb{0}}(x)$, i.e., $P \subseteq \rho_{\mathbb{0}}(x)$.

In the other direction, let v be a vertex of P, i.e., $\rho(v) \in \mathbb{S}(P)$. If $v \in \rho_{\mathbb{0}}(x)$, then by Lemma 4.3.1, $x \in \rho_{\mathbb{0}}(v)$. Therefore, for every $\rho(v) \in \mathbb{S}(P)$, we know that

$x \in \rho_{\mathbb{0}}(v)$, i.e., $x \in \rho_{\mathbb{0}}(P)$. The same proof holds for the second statement by replacing all instances of $\mathbb{0}$ by ∞. □

In the case that $\mathbb{0} \in P$, $\rho_{\infty}(P)$ is empty and the second conclusion of the previous lemma holds trivially. Thus, even though the previous result is always true, it is non-trivial only when $\mathbb{0} \notin P$.

Lemma 4.3.4. *Let P be a polyhedron in $\mathbb{R}^d$. If $x \in P$, then $\rho_{\mathbb{0}}(P) \subseteq \rho_{\mathbb{0}}(x)$ whereas $\rho_{\infty}(P) \subseteq \rho_{\infty}(x)$.*

Proof. Assume for a contradiction that there is a point $y \in \rho_{\mathbb{0}}(P)$ such that $y \notin \rho_{\mathbb{0}}(x)$. Therefore, by Lemma 4.3.1 we know that $x \notin \rho_{\mathbb{0}}(y)$. Moreover, because $y \in \rho_{\mathbb{0}}(P)$, Lemma 4.3.3 implies that $P \subseteq \rho_{\mathbb{0}}(y)$—a contradiction to the fact that $x \in P$ and $x \notin \rho_{\mathbb{0}}(y)$. An analogous proof holds to show that $\rho_{\infty}(P) \subseteq \rho_{\infty}(x)$. □

Note that the converse of Lemma 4.3.4 is not necessarily true.

Lemma 4.3.5. *Let P be a polyhedron in $\mathbb{R}^d$ and let γ be a hyperplane. If γ is either tangent to $\rho_{\mathbb{0}}(P)$ or to $\rho_{\infty}(P)$, then $\rho(\gamma)$ is a point lying on the boundary of P.*

Proof. Let γ be a hyperplane tangent to $\rho_{\mathbb{0}}(P)$ at a vertex v. Because $v \in \gamma$, Lemma 4.3.1 implies that $\rho(\gamma) \in \rho(v)$. We claim that $\rho(\gamma) \in P$. Assume for a contradiction that $\rho(\gamma) \notin P$. Since $v \in \rho_{\mathbb{0}}(P)$, we know that $P \subseteq \rho_{\mathbb{0}}(v)$ by Lemma 4.3.3. Therefore, because $\rho(\gamma) \in \rho(v)$ and from the assumption that $\rho(\gamma) \notin P$, we can slightly perturb $\rho(v)$ to obtain a hyperplane h such that $P \subseteq h_{\mathbb{0}}$ while $\rho(\gamma)$ lies in the interior of h_{∞}. Thus, since $\rho(\gamma) \in h_{\infty}$ while $\rho(\gamma) \notin h$, Lemma 4.3.1 implies that $\rho(h)$ lies in the interior of γ_{∞}. Moreover, because $P \subseteq h_{\mathbb{0}}$ we know by Lemma 4.3.3 that $\rho(h) \in \rho_{\mathbb{0}}(P)$. Therefore, there is a point of $\rho_{\mathbb{0}}(P)$, say $\rho(h)$, that lies in the interior of γ_{∞}—a contradiction to the fact that γ is tangent to $\rho_{\mathbb{0}}(P)$. Therefore, $\rho(\gamma) \in P$. Moreover, because $\rho(\gamma) \in \rho(v)$ and from the fact that $P \subseteq \rho_{\mathbb{0}}(v)$, $\rho(\gamma)$ cannot lie in the interior of P, i.e, $\rho(\gamma)$ lies on the boundary of P. An analogous proof holds for the case when γ is tangent to $\rho_{\infty}(P)$. □

Lemma 4.3.6. *Let P and Q be two polyhedra. If $P \subseteq Q$, then $\rho_{\mathbb{0}}(Q) \subseteq \rho_{\mathbb{0}}(P)$ and $\rho_{\infty}(Q) \subseteq \rho_{\infty}(P)$.*

Proof. Let $x \in \rho_{\mathfrak{0}}(Q)$. Then, Lemma 4.3.3 implies that $Q \subseteq \rho_{\mathfrak{0}}(x)$. Because we assumed that $P \subseteq Q$, $P \subseteq \rho_{\mathfrak{0}}(x)$. Therefore, we infer from Lemma 4.3.3 that $x \in \rho_{\mathfrak{0}}(P)$. That is, $\rho_{\mathfrak{0}}(Q) \subseteq \rho_{\mathfrak{0}}(P)$. An analogous proof holds to show that $\rho_{\infty}(Q) \subseteq \rho_{\infty}(P)$. □

A hyperplane π *separates* two geometric objects in $\mathbb{R}^d$ if they are contained in opposite halfspaces supported by π, note that both objects can contain points lying on π. We obtain the main result of this section illustrated in Figure 4.4.

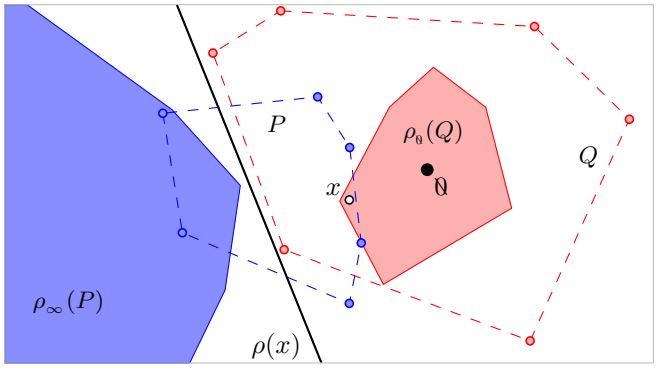

Figure 4.4: The statement of Theorem 4.3.7 where a point x lies in the intersection of P and $\rho_{\mathfrak{0}}(Q)$ if and only if $\rho(x)$ separates Q from $\rho_{\infty}(P)$.

Theorem 4.3.7. *Let P and Q be two polyhedra. The polyhedra P and $\rho_{\mathfrak{0}}(Q)$ intersect if and only if there is a hyperplane that separates $\rho_{\infty}(P)$ from Q. Also, (1) if $x \in P \cap \rho_{\mathfrak{0}}(Q)$, then $\rho(x)$ separates $\rho_{\infty}(P)$ from Q, and (2) if γ is a hyperplane that separates $\rho_{\infty}(P)$ from Q such that γ is tangent to $\rho_{\infty}(P)$, then $\rho(\gamma) \in P \cap \rho_{\mathfrak{0}}(Q)$. Moreover, the symmetric statements of (1) and (2) hold if we replace all instances of P (*resp.* ∞*) by Q (*resp.* $\mathfrak{0}$*) and vice versa.*

Proof. Let x be a point in $P \cap \rho_{\mathfrak{0}}(Q)$. Because $x \in P$, by Lemma 4.3.4 we know that $\rho_{\infty}(P) \subseteq \rho_{\infty}(x)$. Moreover, since $x \in \rho_{\mathfrak{0}}(Q)$, by Lemma 4.3.3, $Q \subseteq \rho_{\mathfrak{0}}(x)$. Therefore, $\rho(x)$ is a hyperplane that separates $\rho_{\infty}(P)$ from Q.

In the other direction, let γ' be a hyperplane that separates $\rho_{\infty}(P)$ from Q. Then, there is a hyperplane γ parallel to γ' that separates $\rho_{\infty}(P)$ from Q such that γ is tangent to $\rho_{\infty}(P)$. Therefore, $\rho(\gamma)$ is a point on the boundary of P by Lemma 4.3.5. Because $\rho(\gamma) \in P$, Lemma 4.3.4 implies that $\rho_{\mathfrak{0}}(P) \subseteq \gamma_{\mathfrak{0}}$ while $\rho_{\infty}(P) \subseteq \gamma_{\infty}$. Because

γ separates $\rho_\infty(P)$ from Q and from the fact that $\rho_\infty(P) \subseteq \gamma_\infty$, we conclude that $Q \subseteq \gamma_{\mathfrak{0}}$. Consequently, by Lemma 4.3.3 $\rho(\gamma) \in \rho_{\mathfrak{0}}(Q)$. That is, $\rho(\gamma)$ is a point in the intersection of P and $\rho_{\mathfrak{0}}(Q)$. The symmetric statements have analogous proofs. □

Notice that if $\mathfrak{0} \in P$, then P and $\rho_{\mathfrak{0}}(Q)$ trivially intersect. Moreover, $\rho_\infty(P) = \emptyset$ implying that every hyperplane trivially separates $\rho_\infty(P)$ from Q. Therefore, while being always true, this result is non-trivial only when $\mathfrak{0} \notin P$.

4.4 Polyhedra in 3D space

In this section, we focus on polyhedra in $\mathbb{R}^3$. Therefore, we can consider the 1-skeleton of a polyhedron being the planar graph connecting its vertices through the edges of the polyhedron.

Given a polyhedron P, a sequence $P_1, P_2, \ldots, P_k$ is a DK-hierarchy of P if the following properties hold [38].

*A*1. $P_1 = P$ and P_k a tetrahedron.

*A*2. $P_{i+1} \subseteq P_i$, for $1 \leq i < k$.

*A*3. $V(P_{i+1}) \subseteq V(P_i)$, for $1 \leq i < k$.

*A*4. The vertices of $V(P_i) \setminus V(P_{i+1})$ form an independent set in P_i, for $1 \leq i < k$.

*A*5. The *height* of the hierarchy $k = O(\log n)$, $\sum_{i=1}^{k} V(P_i) = O(n)$.

Given a polyhedron P on n vertices, a set $I \subseteq V(P)$ is a *P-independent set* if (1) $|I| \geq n/10$, (2) I forms an independent set in the 1-skeleton of P and (3) the degree of every vertex in I is $O(1)$.

Dobkin and Kirkpatrick [38] showed how to construct a DK-hierarchy. This construction was later improved by Biedl and Wilkinson [19]. Formally, they start by defining $P_1 = P$. Then, given a polyhedron P_i, they show how to compute a P_i-independent set I and define P_{i+1} as the convex hull of the set $V(P_i) \setminus I$.

Using this data structure, they provide an algorithm that computes the distance between two preprocessed polyhedra in $O(\log^2 n)$ time [39]. As we show below however, a straightforward implementation of their algorithm could be much slower than this claimed bound.

In our algorithm, as well as in the algorithm presented by Dobkin and Kirkpatrick [39], we are given a plane tangent to P_i at a vertex v and want to find a vertex of P_{i-1} lying on the other side of this plane (if it exists). Although they showed that at most one vertex of P_{i-1} can lie on the other side of this plane and that it has to be adjacent to v, they do not explain how to find such a vertex. An exhaustive walk through the neighbors of v in P_{i-1} would only be fast enough for their algorithm if v is always of constant degree. Unfortunately this is not always the case as shown in the following example.

Start with a tetrahedron P_k and select a vertex q of P_k. To construct the polyhedron P_{i-1} from P_i, we refine it by adding a vertex slightly above each face adjacent to q. In this way, the degree of the new vertices is exactly three. After k steps, we reach a polyhedron $P_1 = P$. In this way, the sequence $P = P_1, P_2, \ldots, P_k$ defines a DK-hierarchy of P. Moreover, when going from P_i to P_{i-1}, a new neighbor of q is added for each of its adjacent faces in P_i. Thus, the degree of q doubles when going from P_i to P_{i-1} and hence, the degree of q in P_1 is linear. Note that this situation can occur at a deeper level of the hierarchy, even if every vertex of P has degree three.

A solution to this problem is described by O'Rourke [78, Chapter 7]. In the next section, we provide an alternative solution to this problem by bounding the degree of each vertex in every polyhedron of the DK-hierarchy.

4.4.1 Bounded hierarchies

Let c be a fixed constant. We say that a polyhedron is *c-bounded* if at most c faces of this polyhedron can meet at a vertex, i.e., the degree of each vertex in its 1-skeleton is bounded by c.

Given a polyhedron P with n vertices, we describe a method to modify the structure of Dobkin and Kirkpatrick to construct a DK-hierarchy where every polyhedron other than P is c-bounded. As a starting point, we can assume that the faces of P

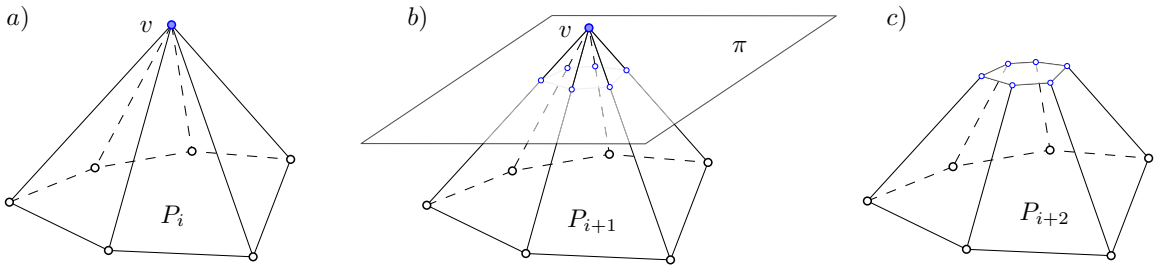

Figure 4.5: A polyhedron P and a vertex v of large degree. A plane π that separates v from $V(P) \setminus \{v\}$ is used to split the edges adjacent to v. New vertices are added to split these edges. Finally, the removal of v from the polyhedron leaves every one of its neighbors with degree three while adding a new face.

are in general position (i.e., no four planes of $S(P)$ go through a single point) by using Simulation of Simplicity [43]. This implies that every vertex of P has degree three. To avoid having vertices of large degree in the hierarchy, we introduce the following operation. Given a vertex $v \in V(P)$ of degree $k > 3$, consider a plane π that separates v from every other vertex of P. Let $e_1, e_2, \ldots, e_k$ be the edges of P incident to v. For each $1 \leq i \leq k$, let v_i be the intersections of e_i with π. *Split* the edge e_i at v_i to obtain a new polyhedron with k more vertices and k new edges; for an illustration see Figure 4.5 (a) and (b).

To construct a c-bounded DK-hierarchy (or simply BDK-hierarchy), we start by letting $P_1 = P$. Given a polyhedron P_i in this BDK-hierarchy, let I be a P_i-independent set. Compute the convex hull of $V(P_i) \setminus I$, two cases arise: **Case 1.** If $\textsc{ch}(V(P_i) \setminus I)$ has no vertex of degree larger than c, then let $P_{i+1} = \textsc{ch}(V(P_i) \setminus I)$. **Case 2.** Otherwise, let W be the set of vertices of P_i with degree larger than 3. For each vertex of W, split its adjacent edges as described above and let P_{i+1} be the obtained polyhedron. Notice that P_{i+1} is a polyhedron with the same number of faces as P_i. Moreover, because each edge of P_i may be split for each of its endpoints, P_{i+1} has at most three times the number the edges of P_i. Therefore $|V(P_{i+1})| \leq (2/3)|E(P_{i+1}) \leq 2|E(P_i)| \leq 6|V(P_i)|$ by Euler's formula.

Because each vertex of W is adjacent only to new vertices added during the split of its adjacent edges, the vertices in W form an independent set in the 1-skeleton of P_{i+1}. In this case, we let P_{i+2} be the convex hull of $V(P_{i+1}) \setminus W$. Therefore, (1) every vertex of P_{i+2} has degree three, and (2) the vertices in $V(P_{i+1}) \setminus V(P_{i+2})$ form an

independent set in P_{i+1}; see Figure 4.5(c). Note that P_{i+1} and P_{i+2} have new vertices added during the splits. However, we know that $|V(P_{i+2})| \leq |V(P_{i+1})| \leq 6|V(P_i)|$. Furthermore, we also know that $P_{i+2} \subseteq P_{i+1} \subseteq P_i$.

We claim that by choosing c carefully, we can guarantee that the depth of the BDK-hierarchy is $O(\log n)$. To prove this claim, notice that after a pruning step, the degree of a vertex can increase at most by the total degree of its neighbors that have been eliminated. Let v be a vertex with the largest degree in P_i. Note that its neighbors can also have at most degree $\delta(v)$, where $\delta(v)$ denotes the number of neighbors of v in P_i. Therefore, after removing a P_i-independent set, the degree of v can be at most $\delta(v)^2$ in P_{i+1}. That is, the maximum degree of P_i can be at most squared when going from P_i to P_{i+1}.

Therefore, if we assume Case 2 has just been applied and that every vertex of P_i has degree three, then after r rounds of Case 1, the maximum degree of any vertex is at most 3^{2^r}. Therefore, the degree of any of its vertices can go above c only after $\log_2(\log_3 c)$ rounds, i.e., we go through Case 1 at least $\log_2(\log_3 c)$ times before running into Case 2.

Since we removed at least 1/10-th of the vertices after each iteration of Case 1 [19], after $\log_2(\log_3 c)$ rounds the size of the current polyhedron is at most $(9/10)^{\log_2(\log_3 c)}|P_i|$. At this point, we run into Case 2 and add extra vertices to the polyhedron. However, by choosing c sufficiently large, we guarantee that the number of remaining vertices is at most $6 \cdot (9/10)^{\log_2(\log_3 c)}|P_i| < \alpha|P_i|$ for some constant $0 < \alpha < 1$. That is, after $\log_2(\log_3 c)$ rounds the size of the polyhedron decreases by constant factor implying a logarithmic depth. We obtain the following result.

Lemma 4.4.1. *Given a polyhedron P, the previous algorithm constructs a BDK-hierarchy $P_1, P_2, \ldots, P_k$ with following properties.*

*B*1. *$P_1 = P$ and P_k is a tetrahedron.*

*B*2. *$P_{i+1} \subseteq P_i$, for $1 \leq i \leq k$.*

*B*3. *The polyhedron P_i is c-bounded, for $1 \leq i \leq k$.*

*B*4. *The vertices of $V(P_i) \setminus V(P_{i+1})$ form an independent set in P_i, for $1 \leq i < k$.*

$B5$. *The* height *of the hierarchy* $k = O(\log n)$, $\sum_{i=1}^{k} V(P_i) = O(n)$.

The following property of a DK-hierarchy of P was proved in [39] and is easily extended to BDK-hierarchies because its proof does not use property $A3$. Note that all properties of DK and BDK hierarchies are identical except for $B3 \neq A3$.

Lemma 4.4.2. *Let* $P_1, \ldots, P_k$ *be a BDK-hierarchy of a polyhedron* P *and let* H *be a plane defining two halfspaces* H^+ *and* H^-. *For any* $1 \leq i \leq k$ *such that* P_{i+1} *is contained in* H^+, *either* $P_i \subseteq H^+$ *or there exists a unique vertex* $v \in V(P_i)$ *such that* $v \in H^- \setminus H$.

4.5 Detecting intersections in 3D

In this section, we show how to independently preprocess polyhedra in 3D-space so that their intersection can be tested in logarithmic time.

4.5.1 Preprocessing

Let P be a polyhedron in $\mathbb{R}^3$. Assume without loss of generality that the origin lies in the interior of P. Otherwise, modify the coordinate system. To preprocess P, we first compute the polyhedron $\rho_{\mathfrak{0}}(P)$ being the polarization of P. Then, we independently compute two BDK-hierarchies as described in Section 4.4, one for P and one for $\rho_{\mathfrak{0}}(P)$. Recall that in the construction of BDK-hierarchies, we assume that the faces of the polyhedra being processed are in general position using Simulation of Simplicity [43]. Assuming that both P and $\rho_{\mathfrak{0}}(P)$ have vertices in general position at the same time is not possible. However, this is not a problem as only one of the two BDK-hierarchies will ever be used in a single query. Therefore, we can independently use Simulation of Simplicity [43] on each of them.

4.5.2 Preliminaries of the algorithm

Let P and R be two independently preprocessed polyhedra with combinatorial complexities n and m, respectively. Throughout this algorithm, we fix the coordinate system used in the preprocessing of R, i.e., $\mathfrak{0} \in R$. For ease of notation, let $Q = \rho_{\mathfrak{0}}(R)$. Because $\mathfrak{0} \in R$, Lemma 4.3.2 implies that $R = \rho_{\mathfrak{0}}(Q)$.

The algorithm described in this section tests if P and $R = \rho_{\mathbb{0}}(Q)$ intersect. Therefore, we can assume that P and $\rho_{\mathbb{0}}(Q)$ lie in a *primal space* while $\rho_{\infty}(P)$ and Q lie in a *polar space*. That is, we look at the primal and polar spaces independently and switch between them whenever necessary. To test the intersection of P and $\rho_{\mathbb{0}}(Q)$ in the primal space, we use the BDK-hierarchies of P and Q stored in the preprocessing step. In an intersection query, we are given arbitrary translations and rotations for P and $\rho_{\mathbb{0}}(Q)$ and we want to decide if they intersect. Note that this is equivalent to answering the query when only a translation and rotation of P is given and $\rho_{\mathbb{0}}(Q)$ remains unchanged. This is important as we fixed the position of the origin inside $R = \rho_{\mathbb{0}}(Q)$. The idea of the algorithm is to proceed by rounds and in each of them, move down in one of the two hierarchies while maintaining some invariants. In the end, when reaching the bottom of the hierarchy, we determine if P and $\rho_{\mathbb{0}}(Q)$ are separated or not.

Let k and l be the depths of the hierarchies of P and Q, respectively. We use indices $1 \leq i \leq k$ and $1 \leq j \leq l$ to indicate our position in the hierarchies of P and Q. The idea is to decrement at least one of them in each round of the algorithm.

To maintain constant time operations, instead of considering a full polyhedron P_i in the BDK-hierarchy of P, we consider constant complexity polyhedra $P_i^* \subseteq P_i$ and $Q_j^* \subseteq Q_j$. Intuitively, both P_i^* and Q_j^* are constant size polyhedra that respectively represent the portions of P_i and Q_j that need to be considered to test for an intersection.

We also maintain a special point p^* in the primal space which is a vertex of both P_i^* and P_i, and a plane φ whose properties will be described later. In the polar space, we keep a point q^* which is a vertex of both Q_j^* and Q_j and a plane γ.

For ease of notation, given a polyhedron T and a vertex $v \in V(T)$, let $T \setminus v$ denote the convex hull of $V(T) \setminus \{v\}$. The *star invariant* consists of two parts, one in the primal and another in the polar space. In the primal space, this invariant states that if $i < k$, then (1) the plane φ separates $P_i \setminus p^*$ from $\rho_{\mathbb{0}}(Q_j)$ and (2) $\rho(\varphi) \in Q_j$. In the polar space, the star invariant states if $j < l$, then (1) the plane γ separates $Q_j \setminus q^*$ from $\rho_{\infty}(P_i)$ and (2) $\rho(\gamma) \in P_i$. Whenever the star invariant is established, we store references to φ and γ, and to the vertices p^* and q^*.

Other invariants are also considered throughout the algorithm. The *separation invariant* states that we have a plane π that separates P_i from $\rho_0(Q_j^*)$ such that π is tangent to P_i at one of its vertices. The *inverse separation invariant* states that there is a plane μ that separates $\rho_\infty(P_i^*)$ from Q_j such that μ is tangent to Q_j at one of its vertices.

Before stepping into the algorithm, we need a couple of definitions. Given a polyhedron T and a vertex $v \in V(T)$, let $N_v(T)$ be a polyhedron defined as the convex hull of v and its neighbors in T. Let $\kappa_v(T)$ be the convex hull of the set of rays apexed at v shooting from v to each of its neighbors in T. That is, $\kappa_v(T)$ is a convex cone with apex v that contains T and has complexity $O(\delta(v))$, where $\delta(v)$ denotes the number of neighbors of v in T. We say that $\kappa_v(T)$ *separates* T from another polyhedron if the latter does not intersect the interior of $\kappa_v(T)$.

4.5.3 The algorithm

To begin the algorithm, let $i = k$ and $j = l$, i.e., we start with $P_i^* = P_i$ and $Q_j^* = Q_j$ being both tetrahedra. Notice that for the base case, $i = k$ and $j = l$, we can determine in $O(1)$ time if P_i and $\rho_0(Q_j) = \rho_0(Q_j^*)$ intersect. If they do not, then we can compute a plane separating them and establish the separation invariant. Otherwise, if P_i and $\rho_0(Q_j)$ intersect, then by Theorem 4.3.7 we know that $\rho_\infty(P_i) = \rho_\infty(P_i^*)$ does not intersect Q_j. Thus, in constant time we can compute a plane tangent to Q_j in the polar space that separates $\rho_\infty(P_i) = \rho_\infty(P_i^*)$ from Q_j. That is, we can establish the inverse separation invariant. Thus, at the beginning of the algorithm the star invariant holds trivially, and either the separation invariant or the inverse separation invariant holds (maybe both if P_i and $\rho_0(Q_j)$ are tangent).

After each round of the algorithm, we advance in at least one of the hierarchies of P and Q while maintaining the star invariant. Moreover, we maintain at least one among the separation and the inverse separation invariants. Depending on which invariant is maintained, we step into the primal or the polar space as follows (if both invariants hold, we choose arbitrarily).

4.5.4 A walk in the primal space

We step into this case if the separation invariant holds. That is, P_i is separated from $\rho_0(Q_j^*)$ by a plane π tangent to P_i at a vertex v.

We know by Lemma 4.4.2 that there is at most one vertex p in P_{i-1} that lies in $\pi_0 \setminus \pi$. Moreover, this vertex must be a neighbor of v in P_{i-1}. Because P_{i-1} is c-bounded, we scan the $O(1)$ neighbors of v and test if any of them lies in $\pi_0 \setminus \pi$. Two cases arise:

Case 1. If P_{i-1} is contained in π_∞, then π still separates P_{i-1} from $\rho_0(Q_j^*)$ while being tangent to the same vertex v of P_{i-1}. Therefore, we have moved down one level in the hierarchy of P while maintaining the separation invariant.

To maintain the star invariant, let $P_{i-1}^* = N_v(P_{i-1})$ and let $p^* = v \in V(P_{i-1}^*) \cap V(P_{i-1})$. Because P_{i-1} is c-bounded, we know that P_{i-1}^* has constant size. Since $\rho_0(Q_j^*)$ has constant size, we can compute the plane φ parallel to π and tangent to $\rho_0(Q_j^*)$ in $O(1)$ time, i.e., φ also separates P_{i-1} from $\rho_0(Q_j^*)$. Because $\rho_0(Q_j^*) \supseteq \rho_0(Q_j)$ by Lemma 4.3.6 and from the fact that $P_{i-1} \setminus p^* \subset P_{i-1}$, we conclude that (1) φ separates $P_{i-1} \setminus p^*$ from $\rho_0(Q_j)$. Moreover, because $\rho(\varphi) \in Q_j^*$ by Lemma 4.3.5 and from the fact that $Q_j^* \subseteq Q_j$, we conclude that (2) $\rho(\varphi) \in Q_j$. Thus, the star invariant is maintained in the primal space.

In the polar space, if $j < l$, then since $\rho_\infty(P_{i-1}) \subseteq \rho_\infty(P_i)$ by Lemma 4.3.6, (1) the plane γ that separates $Q_j \setminus q^*$ from $\rho_\infty(P_i)$ also separates $Q_j \setminus q^*$ from $\rho_\infty(P_{i-1})$. Moreover, because $P_i \subseteq P_{i-1}$ and from the fact that $\rho(\gamma) \in P_i$, we conclude that (2) $\rho(\gamma) \in P_{i-1}$. Thus, the star invariant is also maintained in the polar space and we proceed with a new round of the algorithm in the primal space.

Case 2. If P_{i-1} crosses π, then by Lemma 4.4.2 there is a unique vertex p of P_{i-1} that lies in $\pi_0 \setminus \pi$. To maintain the star invariant, let $P_{i-1}^* = N_p(P_{i-1})$ and let $p^* = p$. Then, proceed as in to the first case. In this way, we maintain the star invariant in both the primal and the polar space.

Recall that $\kappa_{p^*}(P_{i-1})$ is the cone being the convex hull of the set of rays shooting from p^* to each of its neighbors in P_{i-1}. Since P_{i-1} is c-bounded, p^* has at most c neighbors in P_{i-1}. Thus, both $\kappa_{p^*}(P_{i-1})$ and $\rho_0(Q_j^*)$ have constant complexity and we

can test if they intersect in constant time. Two cases arise:

Case 2.1. If $\kappa_{p^*}(P_{i-1})$ and $\rho_{\emptyset}(Q_j^*)$ do not intersect, then as $P_{i-1} \subseteq \kappa_{p^*}(P_{i-1})$, we can compute in constant time a plane π' tangent to $\kappa_{p^*}(P_{i-1})$ at p^* that separates $P_{i-1} \subseteq \kappa_{p^*}(P_{i-1})$ from $\rho_{\emptyset}(Q_j^*)$. That is, we reestablish the separation invariant and proceed with a new round in the primal space.

Case 2.2. Otherwise, if $\kappa_{p^*}(P_{i-1})$ and $\rho_{\emptyset}(Q_j^*)$ intersect, then because $P_{i-1} \setminus p^* \subseteq \pi_\infty$ and $\rho_{\emptyset}(Q_j^*) \subseteq \pi_{\emptyset}$, we know that this intersection happens at a point of P_{i-1}^*, i.e., P_{i-1}^* intersects $\rho_{\emptyset}(Q_j^*)$. Therefore, by Theorem 4.3.7 there is a plane μ' that separates $\rho_\infty(P_{i-1}^*)$ from Q_j^* in the polar space. In this case, we would like to establish the inverse separation invariant which states that $\rho_\infty(P_{i-1}^*)$ is separated from Q_j. Note that if $j = l$, then $Q_j = Q_j^*$ and the inverse separation invariant is established. Therefore, assume that $j < l$ and recall that $q^* \in V(Q_j^*) \cap V(Q_j)$.

By the star invariant and from the assumption that $j < l$, the plane γ separates $Q_j \setminus q^*$ from $\rho_\infty(P_{i-1})$, i.e., $Q_j \setminus q^* \subseteq \gamma_{\emptyset}$. In this case, we enlarge P_{i-1}^* by adding the vertex $\rho(\gamma)$ to it, i.e., we let $P_{i-1}^* = \textsc{ch}(N_p(P_{i-1}) \cup \{\rho(\gamma)\})$. Note that this enlargement preserves the star invariant as p^* is still a vertex of the refined P_{i-1}^*. Moreover, because $\rho(\gamma) \in P_{i-1}$ by the star invariant, we know that $P_{i-1}^* \subseteq P_{i-1}$.

Because $\rho(\gamma) \in P_{i-1}^*$, Lemma 4.3.4 implies that $\rho_\infty(P_{i-1}^*) \subseteq \gamma_\infty$. Since $Q_j \setminus q^* \subseteq \gamma_{\emptyset}$, γ separates $\rho_\infty(P_{i-1}^*)$ from $Q_j \setminus q^*$. Because μ' separates $\rho_\infty(P_{i-1}^*)$ from $Q_j^* \supseteq N_{q^*}(Q_j)$, we conclude that there is a plane that separates $\rho_\infty(P_{i-1}^*)$ from Q_j and it only remains to compute it in $O(1)$ time.

In fact, because $Q_j \setminus q^* \subseteq \gamma_{\emptyset}$, all neighbors of q^* in Q_j lie in $\gamma_{\emptyset}$ and hence, the cone $\kappa_{q^*}(Q_j)$ does not intersect $\rho_\infty(P_{i-1}^*)$. Since $\kappa_{q^*}(Q_j)$ and $\rho_\infty(P_{i-1}^*)$ have constant complexity, we can compute a plane μ tangent to $\kappa_{q^*}(Q_j)$ at q^* such that μ separates $\kappa_{q^*}(Q_j)$ from $\rho_\infty(P_{i-1}^*)$. Because $Q_j \subseteq \kappa_{q^*}(Q_j)$, μ separates Q_j from $\rho_\infty(P_{i-1}^*)$ while being tangent to Q_j at q^*. That is, we establish the inverse separation invariant. In this case, we step into the polar space and try to move down in the hierarchy of Q in the next round of the algorithm.

4.5.5 A walk in the polar space

We step into this case if the inverse separation invariant holds. That is, we have a plane tangent to Q_j at one of its vertices that separates $\rho_\infty(P_i^*)$ from Q_j. In this case, we perform an analogous procedure to that described for the case when the separation invariant holds. However, all instances of P_i (*resp.* P) are replaced by Q_j (*resp.* Q) and vice versa, and all instances of $\rho_\infty(*)$ are replaced by $\rho_0(*)$ and vice versa. Moreover, all instances of the separation and the inverse separation invariant are also swapped. At the end of this procedure, we decrease the value of j and establish either the separation or the inverse separation invariant. Moreover, the star invariant is also preserved should there be a subsequent round of the algorithm.

4.5.6 Analysis of the algorithm

After going back and forth between the primal and the polar space, we reach the bottom of the hierarchy of either P or Q. Thus, we might reach a situation in which we analyze P_1 and $\rho_0(Q_j^*)$ in the primal space for some $1 \leq j \leq l$. In this case, if the separation invariant holds, then we have computed a plane π that separates P_1 from $\rho_0(Q_j^*) \supseteq \rho_0(Q)$. Because $P = P_1$, we conclude that π separates P from $R = \rho_0(Q)$.

We may also reach a situation in which we test Q_1 and $\rho_\infty(P_i^*)$ in the polar space for some $1 \leq i \leq k$. In this case, if the inverse separation invariant holds, then we have a plane μ that separates Q_1 from $\rho_\infty(P_i^*)$. Since $\rho_\infty(P_i^*)$ has constant complexity, we can assume that μ is tangent to $\rho_\infty(P_i^*)$ as we can compute a plane parallel to μ with this property. Because $Q = Q_1$, we conclude that μ is a plane that separates Q from $\rho_\infty(P_i^*)$ such that μ is tangent to $\rho_\infty(P_i^*)$. Therefore, Theorem 4.3.7 implies that $\rho(\mu)$ is a point in the intersection of $P_i^* \subseteq P$ and $\rho_0(Q)$, i.e., P and $R = \rho_0(Q)$ intersect.

In any other situation the algorithm can continue until one of the two previously mentioned cases arises and the algorithm finishes. Because we advance in each round in either the BDK-hierarchy of P or the BDK-hierarchy of Q, after $O(\log n + \log m)$ rounds the algorithm finishes. Because each round is performed in $O(1)$ time, we obtain the following result.

Theorem 4.5.1. *Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^3$*

with combinatorial complexities n and m, respectively. For any given translations and rotations of P and R, we can determine if P and R intersect in $O(\log n + \log m)$ time.

4.6 Detecting intersections in higher dimensions

In this section, we extend our algorithm to any constant dimension d at the expense of increasing the space to $O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$. To do that, we replace the BDK-hierarchy and introduce a new hierarchy produced by recursively taking ε-nets of the faces of the polyhedron. Our objective is to obtain a new hierarchy with logarithmic depth with properties similar to those described in Lemma 4.4.2. For the latter, we use the following definition.

Given a polyhedron P, the intersection of $(d+1)$ halfspaces is a *shell-simplex* of P if it contains P and each of these $(d+1)$ halfspaces is supported by a $(d-1)$-dimensional face of P.

Lemma 4.6.1. *Let P be a polyhedron in $\mathbb{R}^d$ with k vertices. We can compute a set $\Sigma(P)$ of at most $O(k^{\lfloor d/2 \rfloor})$ shell-simplices of P such that given a hyperplane π tangent to P, there is a shell-simplex $\sigma \in \Sigma(P)$ such that π is also tangent to σ.*

Proof. Without loss of generality assume that $\mathfrak{0} \in P$. Note that $\rho_{\mathfrak{0}}(P)$ has exactly k $(d-1)$-dimensional faces. Using Lemma 3.8 of [34] we infer that there exists a triangulation T of $\rho_{\mathfrak{0}}(P)$ such that the combinatorial complexity of T is $O(k^{\lfloor d/2 \rfloor})$. That is, T decomposes $\rho_{\mathfrak{0}}(P)$ into interior disjoint d-dimensional simplices.

Let s be a simplex of T. For each $v \in V(s)$, notice that since $v \in \rho_{\mathfrak{0}}(P)$, $P \subseteq \rho_{\mathfrak{0}}(v)$ by Lemma 4.3.4. Therefore, $P \subseteq \cap_{v \in V(s)} \rho_{\mathfrak{0}}(v) = \rho_{\mathfrak{0}}(s)$, i.e., $\sigma_s = \rho_{\mathfrak{0}}(s)$ is a shell-simplex of P obtained from polarizing s. Finally, let $\Sigma(P) = \{\sigma_s : s \in T\}$ and notice that $|\Sigma(P)| = O(k^{\lfloor d/2 \rfloor})$.

Because $\mathfrak{0} \in P$, Lemma 4.3.2 implies that $P = \rho_{\mathfrak{0}}(\rho_{\mathfrak{0}}(P))$. Let π be a hyperplane tangent to $P = \rho_{\mathfrak{0}}(\rho_{\mathfrak{0}}(P))$ and note that its polar is a point $\rho(\pi)$ lying on the boundary of $\rho_{\mathfrak{0}}(P)$ by Lemma 4.3.5. Hence, $\rho(\pi)$ lies on the boundary of a simplex s of T. Thus, by Lemma 4.3.4 we know that $\sigma_s \subseteq \pi_{\mathfrak{0}}$. Because $\rho(\pi)$ lies on the boundary of s, π is tangent to σ_s yielding our result. □

4.6.1 Hierarchical trees

Let P be a polyhedron with combinatorial complexity n. We can assume that the vertices of P are in general position (i.e., no $d+1$ vertices lie on the same hyperplane) using Simulation of Simplicity [43].

Let $F(P)$ be the set of all faces of P. Consider the family G such that a set $g \in G$ is the complement of the intersection of $d+1$ halfspaces. Let $(F(P), \mathcal{G}_{F(P)})$ be the geometric set system induced by G on $F(P)$ (see Section 3.1).

To compute the hierarchy of P, let $0 < \varepsilon < 1$ and consider the set system defined by $F(P)$ and $\mathcal{G}_{F(P)}$. Since the VC-dimension of this set system is finite by Theorem 3.1.2, we can compute an ε-net N of $(F(P), \mathcal{G}_{F(P)})$ of size $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}) = O(1)$ using Theorem 3.1.3. Because the vertices of P are in general position, each face of P has at most $d+1$ vertices. Therefore, $\textsc{ch}(N)$ has $O(|N|)$ vertices, i.e., $\textsc{ch}(N)$ has constant complexity. By Lemma 4.6.1 and since $|N| = O(1)$, we can compute the set $\Sigma(\textsc{ch}(N))$ having $O(|N|^{\lfloor d/2 \rfloor})$ shell-simplices of $\textsc{ch}(N)$ in constant time.

Given a shell-simplex $\sigma \in \Sigma(\textsc{ch}(N))$, let $\bar{\sigma} \in G$ be the complement of σ. Because $\textsc{ch}(N) \subseteq \sigma$, $\bar{\sigma}$ intersects no face of N. Let $F_{\bar{\sigma}} = \{f \in F(P) : f \cap \bar{\sigma} \neq \emptyset\}$ be the set of faces of P intersected by $\bar{\sigma}$. Therefore, since N is an ε-net of $(F(P), \mathcal{G}_{F(P)})$, we conclude that $F_{\bar{\sigma}}$ contains at most $\varepsilon|F(P)|$ faces of P.

We construct the *hierarchical tree* of a polyhedron P recursively. In each recursive step, we consider a subset F of the faces of P as input. As a starting point, let $F = F(P)$. The recursive construction considers two cases: (1) If F consists of a constant number of faces, then its tree consists of a unique node storing a reference to $\textsc{ch}(F)$. (2) Otherwise, compute the ε-net N of F as described above and store $\textsc{ch}(N)$ together with $\Sigma(\textsc{ch}(N))$ at the root node. Then, for each shell-simplex $\sigma \in \Sigma(\textsc{ch}(N))$ construct recursively the tree for $F_{\bar{\sigma}}$ and attach it to the root node. Because the size of the ε-net is independent of the size of the polyhedron, we obtain a hierarchical structure being a tree rooted at $\textsc{ch}(N)$ with maximum degree $O(|N|^{\lfloor d/2 \rfloor})$.

Lemma 4.6.2. *Given a polyhedron P in $\mathbb{R}^d$ with combinatorial complexity n and any $\delta > 0$, we can compute a hierarchical tree for P with $O(\log n)$ depth in $O(n^{\lfloor d/2 \rfloor + \delta})$ time using $O(n^{\lfloor d/2 \rfloor + \delta})$ space.*

Proof. Because we reduce the number of faces of the original polyhedron by a factor of ε on each branching of the hierarchical tree, the depth of this tree is $O(\log n)$.

The space $S(n)$ of this hierarchical tree of P can be described by the following recurrence $S(n) = O(|N|^{\lfloor d/2 \rfloor})S(\varepsilon n) + O(1)$. Recall that $|N| = O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ by Theorem 3.1.1. Moreover, if we let $r = 1/\varepsilon$, we can solve this recurrence using the master theorem and obtain that $S(n) = O(n^{\frac{\lfloor d/2 \rfloor \log(Cr\log r)}{\log r}})$ for some constant $C > 0$. Therefore, by choosing $r = 1/\varepsilon$ sufficiently large, we obtain that the total space is $S(n) = O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$ arbitrarily small. To analyze the time needed to construct this hierarchical tree, recall that an ε-net can be computed in linear time [67] which leads to the following recurrence $T(n) = O(|N|^{\lfloor d/2 \rfloor})S(\varepsilon n) + O(n)$. Using the same arguments as for the space we solve this recurrence and obtain that the total time is $T(n) = O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$ arbitrarily small. □

4.6.2 Testing intersection in higher dimensions

Using hierarchical trees, we extend the ideas used for the 3D algorithm presented in Section 4.5 to higher dimensions. We start by describing the preprocessing of a polyhedron.

4.6.3 Preprocessing

Let P be a polyhedron $\mathbb{R}^d$ with combinatorial complexity n. Assume without loss of generality that the origin lies in the interior of P. Otherwise, modify the coordinate system. To preprocess P, we first compute the polyhedron $\rho_0(P)$ being the polarization of P. Then, we compute two hierarchical trees as described in the previous section, one for P and another for $\rho_0(P)$. Similarly to the 3D case, because only one of the two hierarchical trees will ever be used in a single intersection query, we can independently use Simulation of Simplicity [43] in the construction of each of the trees. Because $|F(\rho_0(P))| = |F(P)| = n$ by Corollary 2.14 of [92], the total size of these hierarchical trees is $O(n^{\lfloor d/2 \rfloor + \delta})$.

4.6.4 Preliminaries of the algorithm

Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^d$ with combinatorial complexities n and m, respectively. Throughout this algorithm, we fix the coordinate system used in the preprocessing of R, i.e., we assume that $\mathbb{0} \in R$. For ease of notation, let $Q = \rho_{\mathbb{0}}(R)$. Because $\mathbb{0} \in R$, Lemma 4.3.2 implies that $R = \rho_{\mathbb{0}}(Q)$. Assume that P and $\rho_{\mathbb{0}}(Q)$ lie in a *primal space* while $\rho_{\infty}(P)$ and Q lie in a *polar space*. As in the 3D-algorithm, we look at the primal and polar spaces independently and switch between them whenever necessary.

To test the intersection of P and $R = \rho_{\mathbb{0}}(Q)$, we use the hierarchical trees of P and Q computed during the preprocessing step. The idea is to walk down these trees using paths going from the root to a leaf while maintaining some invariants.

Throughout the algorithm, we prune the faces of P and keep only those that can define an intersection. Formally, we consider a set $F^*(P) \subseteq F(P)$ such that P intersects $\rho_{\mathbb{0}}(Q)$ if and only if a face of $F^*(P)$ intersects $\rho_{\mathbb{0}}(Q)$. In the same way, we prune $F(Q)$ and maintain a set $F^*(Q) \subseteq F(Q)$ such that Q intersects $\rho_{\infty}(P)$ if and only if a face of $F^*(Q)$ intersects $\rho_{\infty}(P)$. If these properties hold, we say that the *correctness invariant* is maintained.

At the beginning of the algorithm let $F^*(P) = F(P)$ and $F^*(Q) = F(Q)$. In each round of the algorithm we discard a constant fraction of either $F^*(P)$ or $F^*(Q)$ while maintaining the correctness invariant. Note that these sets are not explicitly maintained.

Throughout, we consider constant size polyhedra $P_N \subseteq P$ and $Q_N \subseteq Q$ being the convex hull of ε-nets of $F^*(P)$ and $F^*(Q)$, respectively. The algorithm tests if P_N and $\rho_{\mathbb{0}}(Q_N)$ intersect to determine either the separation or the inverse separation invariant, both analogous to those used by the 3D-algorithm. Formally, the *separation invariant* states that we have a hyperplane π that separates P_N from $\rho_{\mathbb{0}}(Q_N)$ such that π is tangent to P_N at one of its vertices. The *inverse separation invariant* states that there is a hyperplane μ that separates $\rho_{\infty}(P_N)$ from Q_N such that μ is tangent to Q_N at one of its vertices. By Theorem 4.3.7 at least one of the invariants must hold.

At the beginning of the algorithm, we let $P_N \subseteq P$ and $Q_N \subseteq Q$ be the convex hulls

of the ε-nets computed for $F(P)$ and $F(Q)$ at the root of their respective hierarchical trees. Because they have constant complexity, we can test if the separation or the inverse separation invariant holds. Depending on which invariant is established, we step into the primal or the polar space as follows (if both invariants hold, we choose arbitrarily).

4.6.5 Separation invariant

If the separation invariant holds, then we have a hyperplane π tangent to P_N at a vertex v such that π separates P_N from $\rho_{\vartheta}(Q_N)$. Therefore, by Lemma 4.6.1 there is a simplex $\sigma \in \Sigma(P_N)$ such that π is also tangent to σ at v. Because we stored $\Sigma(P_N)$ in the hierarchical tree, we go through the $O(1)$ shell-simplices of $\Sigma(P_N)$ to find σ. Recall that $F_{\bar{\sigma}}$ is the set of faces of $F^*(P)$ that intersect the complement of σ. Thus, every face of P intersecting the halfspace π_{ϑ} belongs to $F_{\bar{\sigma}}$.

Because π separates P_N from $\rho_{\vartheta}(Q) \subseteq \rho_{\vartheta}(Q_N) \subseteq \pi_{\vartheta}$, the only faces of $F^*(P)$ that could define an intersection with $\rho_{\vartheta}(Q)$ are those in $F_{\bar{\sigma}}$, i.e., a face of $F^*(P)$ intersects $\rho_{\vartheta}(Q)$ if and only if a face of $F_{\bar{\sigma}}$ intersects $\rho_{\vartheta}(Q)$. Because the correctness invariant held prior to this step, we conclude that a face of $F_{\bar{\sigma}}$ intersects $\rho_{\vartheta}(Q)$ if and only if P intersects $\rho_{\vartheta}(Q)$.

Recall that we have recursively constructed a tree for $F_{\bar{\sigma}}$ which hangs from the node storing P_N. In particular, in the root of this tree we have stored the convex hull of an ε-net of $F_{\bar{\sigma}}$. Therefore, after finding σ in $O(1)$ time, we move down one level to the root of the tree of $F_{\bar{\sigma}}$. Then, we redefine P_N to be the convex hull of the ε-net of $F_{\bar{\sigma}}$ stored in this node. Moreover, we let $F^*(P) = F_{\bar{\sigma}}$ which preserves the correctness invariant. Then, we test if the new P_N and $\rho_{\vartheta}(Q_N)$ intersect to determine if either the separation or inverse separation invariant holds. In this way, we moved down one level in the hierarchical tree of P and proceed with a new round of the algorithm.

4.6.6 Inverse separation invariant

If the inverse separation invariant holds, then we have a hyperplane that separates $\rho_{\infty}(P_N)$ from Q_N. Applying an analogous procedure to the one described for the separation invariant, we redefine Q_N and move down one level in the hierarchical tree

of Q while maintaining the correctness invariant. Then, we test if $\rho_\infty(P_N)$ intersects the new Q_N to determine if either the separation or inverse separation invariants holds and proceed with the algorithm.

After $O(\log n + \log m)$ rounds, the algorithm reaches the bottom of the hierarchical tree of either P or Q. If we reach the bottom of the hierarchical tree of P and the separation invariant holds, then because $\rho_\vartheta(Q_N) \supseteq \rho_\vartheta(Q)$ by Lemma 4.3.6, we have a hyperplane that separates $P_N = \textsc{ch}(F^*(P))$ from $\rho_\vartheta(Q_N)$. That is, no face of $F^*(P)$ intersects $\rho_\vartheta(Q_N)$. Because P and $\rho_\vartheta(Q)$ intersect if and only if a face of $F^*(P)$ intersects $\rho_\vartheta(Q)$ by the correctness invariant, we conclude that P and $R = \rho_\vartheta(Q)$ do not intersect.

Analogously, if we reach the bottom of the hierarchical tree of Q and the inverse separation invariant holds, then we have a hyperplane that separates $Q_N = \textsc{ch}(F^*(Q))$ from $\rho_\infty(P_N) \supseteq \rho_\infty(P)$. That is, no face of $F^*(Q)$ intersects $\rho_\infty(P)$. Thus, by the correctness invariant, we conclude that Q and $\rho_\infty(P)$ do not intersect. Therefore, Theorem 4.3.7 implies that P and $R = \rho_\vartheta(Q)$ intersect.

In any other situation the algorithm can continue until one of the two previously mentioned cases arises and the algorithm finishes. Recall that the hierarchical trees of P and Q have logarithmic depth by Lemma 4.6.2. Because in each round we move down in the hierarchical tree of either P or Q, after $O(\log n + \log m)$ rounds the algorithm finishes. Moreover, since each round can be performed in $O(1)$ time, we obtain the following result.

Theorem 4.6.3. *Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^d$ with combinatorial complexities n and m, respectively. For any given translations and rotations of P and R, we can determine if P and R intersect in $O(\log n + \log m)$ time.*

Note that this algorithm does not construct a hyperplane that separates P and $\rho_\vartheta(Q)$ or a common point, but only determines if such a separating plane or intersection point exists. In fact, if P is disjoint from $\rho_\vartheta(Q)$, then we can take the $O(\log n)$ hyperplanes found by the algorithm, each of them separating some portion of P from $\rho_\vartheta(Q)$. Because all these hyperplanes support a halfspace that contains $\rho_\vartheta(Q)$, their

intersection defines a polyhedron S that contains $\rho_0(Q)$ and excludes P. Therefore, we have a certificate of size $O(\log n)$ that guarantees that P and $\rho_0(Q)$ are separated.

Similarly, if Q is disjoint from $\rho_\infty(P)$, then we can find a polyhedron of size $O(\log m)$ whose boundary separates Q from $\rho_\infty(P)$. In this case, we have a certificate that guarantees that Q and $\rho_\infty(P)$ are disjoint which by Theorem 4.3.7 implies that P and $\rho_0(Q)$ intersect.

Part II

Voronoi Diagrams and Facility Location

Chapter 5

Incremental Voronoi

Let S be a set of n sites in the plane. The graph structures of different kinds of Voronoi diagrams capture much of the proximity information of that set and have several applications in computational geometry, shape reconstruction, computational biology, and machine learning.

One of the most popular algorithms for constructing Voronoi diagrams inserts sites in random order, incrementally updating the diagram [35]. In that case, backward analysis shows that the expected number of changed edges in $\mathcal{V}(S)$ is constant, offering some hope that an efficient dynamic—or at least semi-dynamic—data structure for maintaining $\mathcal{V}(S)$ could exist. These hopes, however, are rapidly squashed, as it is easy to construct examples where the complexity of each successively added face is $\Omega(n)$, and thus each insertion changes the position of a linear number of vertices and edges of $\mathcal{V}(S)$. The goal of this chapter is to show that despite this worst-case behavior, the amortized number of *structural changes* to the graph of the Voronoi diagram of S, that is, the minimum number of edge-insertions and edge-deletions needed to update $\mathcal{V}(S)$ throughout any sequence of site insertions to S, is much smaller.

To overcome the worst-case behavior, Aronov et al. [8] studied what happens in the specific case of points in convex position added in clockwise order using the more elementary *link* (add an edge) and *cut* (delete an edge) operations in the Voronoi diagram. They show that, in that model, it is possible to reconfigure the tree after each site insertion while performing $O(\log n)$ links and cuts, amortized; however their proof is existential and no algorithm is provided to find those links and cuts. Pettie [82] shows both an alternate proof of that fact using forbidden 0-1 matrices and a matching lower bound.

5.1 Flarbs

In the mid-1980's it was observed that a number of variants of Voronoi diagrams and Delaunay triangulations using different metrics (Euclidean distance, L_p norms, convex distance functions) or different kinds of sites (points, segments, circles) could all be handled using similar techniques. To formalize this, several abstract frameworks were defined, such as the one of Edelsbrunner and Seidel [44] and the two variants of abstract Voronoi diagrams of Klein [61, 62]. In this chapter we define a new abstract framework to deal with Voronoi diagrams constructed incrementally by inserting new sites.

Let G be a 3-regular embedded plane graph with n vertices[1]. We seek to bound the number of edge removals and insertions needed to implement the following operation, hereafter referred to as a *flarb*: Given a simple closed curve $\mathcal{C}$ in the plane whose interior intersects G in a connected component, split both $\mathcal{C}$ and all the edges that it crosses at the point of intersection, remove every edge and vertex that lies in the interior of $\mathcal{C}$, and add each curve in the subdivision of $\mathcal{C}$ as a new edge; see Figure 5.1. This operation can be used to represent the insertion of new regions in different types of Voronoi diagrams. It can also be used to represent the changes to the 1-skeleton of a polyhedron in $\mathbb{R}^3$ after it is intersected with a halfspace.

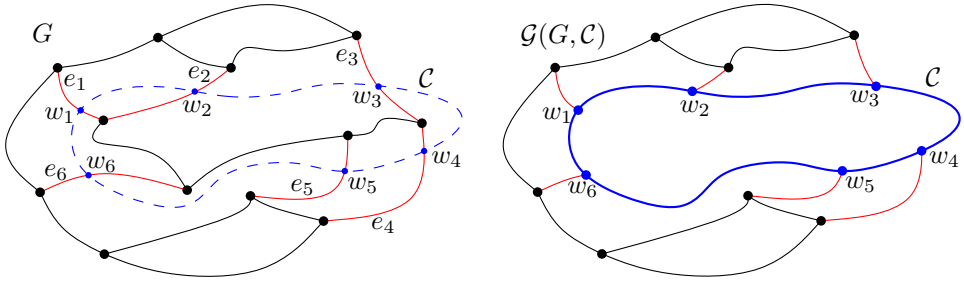

Figure 5.1: The flarb operation on a graph G induced by a flarbable curve $\mathcal{C}$, produces a graph $\mathcal{G}(G,\mathcal{C})$ with 2 more vertices. Fleeq-edges crossed by $\mathcal{C}$ are shown in red.

[1]While the introduction used n for the number of sites in S, the combinatorial part of this article uses n for the number of vertices in the Voronoi diagram. By Euler's formula, those two values are asymptotically equivalent, up to a constant factor.

5.2 Results

We show that the amortized *cost* of a flarb operation, where the combinatorial cost is defined to be the minimum number of edge insertions and removals needed to perform it, is $O(\sqrt{n})$. We also show a matching lower bound: some sequences of flarbs require $\Omega(\sqrt{n})$ links and cuts per flarb, even when the graph is a tree (or more precisely a Halin graph—a tree with all leaves connected by a cycle to make it 3-regular). This contrasts with the $O(\log n)$ upper bound of Aronov et al. [8] for the Voronoi diagram of points in convex position (also a tree) when points are added in clockwise order.

We complement these combinatorial bounds with an algorithmic result. We present an output-sensitive data structure that maintains the farthest-point Voronoi diagram of a set S of n points in convex position as new points are added to S. Upon an insertion, the data structure finds the minimum (up to within a constant factor) number K of edge insertions and deletions necessary to update the farthest-point Voronoi diagram of S. The running time of each insertion is $O(K \log^7 n)$, and by our combinatorial bounds, $K = O(\sqrt{n})$ amortized. This solves the open problem posed by Aronov et al. [8].

The distinguishing feature of this data structure is that it explicitly maintains the graph structure of the farthest-point Voronoi diagram after every insertion, a property that is not provided by any farthest (or nearest) neighbor data structure that uses decomposable searching problem techniques. Further, the data structure also maintains the farthest-point Voronoi diagram in a *grappa tree* [8], a variant of the link-cut trees of Sleator and Tarjan [86] that allows a powerful query operation called *oracle-search*. Roughly speaking, the oracle-search query has access to an oracle specifying a vertex to find. Given an edge of the tree, the oracle determines which of the two subtrees attached to its endpoints contains that vertex. Grappa trees use $O(\log n)$ time and oracle calls to find the sought vertex. A grappa tree is in some sense a dynamic version of the centroid decomposition for trees, which is used in many algorithms for searching in Voronoi diagrams. Using this structure, it is possible to solve a number of problems for the set S at any moment during the incremental construction, for example:

- Given a point q, find the Voronoi region containing q in $O(\log n)$ time. This not

only gives the nearest neighbor of q, but a pointer to the explicit description of its region.

- Find the smallest disk enclosing S, centered on a query segment $[pq]$, in $O(\log n)$ time [23].

- Find the smallest disk enclosing S, centered on a query circle C, in $O(\log n)$ time [14].

- Given a convex polygon P (counterclockwise array of its m vertices), find the smallest disk enclosing S and excluding P in $O(\log n + \log m)$ time [5].

The combinatorial bound for Voronoi diagrams also has direct algorithmic consequences, the most important being that it is possible to store all versions of the graph throughout a sequence of insertions using persistence in $O(n^{3/2})$ space. Since the entire structure of the graph is stored for each version, this provides a foundation for many applications that, for instance, would require searching the sequence of insertions for the moment during which a specific event occurred.

5.3 Outline

The main approach used to bound the combinatorial cost of a flarb is to examine how the complexity of the faces changes. Notice that faces whose size remains the same do not require edge insertions and deletions. The other faces either grow or shrink, and a careful counting argument reveals that the cost of a flarb is at most the number faces that shrink (or disappear) upon execution of the flarb (Section 5.4). By using a potential function that sums the sizes of all faces, the combinatorial cost of shrinking faces is paid for by the reduction of their potential. To avoid incurring a high increase in potential for a large new face, the potential of each face is capped at $\sqrt{n}$. Then at most $O(\sqrt{n})$ large faces can shrink without changing potential and are accounted for separately (Section 5.5). The matching $\Omega(\sqrt{n})$ lower bound is presented in Section 5.6, and Section 5.7 presents the data structure for performing flarbs for the farthest-point Voronoi diagrams of points in convex position.

5.4 The flarb operation

In this section we formalize the flarb operation that models the insertion of new sites in Voronoi diagrams, and present a preliminary analysis of the cost of a flarb.

Let $G = (V, E)$ be a planar 3-regular graph embedded in $\mathbb{R}^2$ (not-necessarily with a straight-line embedding). Let $\mathcal{C}$ be a simple closed Jordan curve in the plane. Define $\textsc{in}(\mathcal{C})$ to be the set of vertices of G that lie in the interior of $\mathcal{C}$ and let $\textsc{ex}(\mathcal{C}) = V \backslash \textsc{in}(\mathcal{C})$. We say that $\mathcal{C}$ is *flarbable* for G if the following conditions hold:

1. the graph induced by $\textsc{in}(\mathcal{C})$ is connected,
2. $\mathcal{C}$ intersects each edge of G either at a single point or not at all,
3. $\mathcal{C}$ passes through no vertex of G, and
4. the intersection of $\mathcal{C}$ with each face of G is path-connected.

In the case where the graph G is clear from context, we simply say that $\mathcal{C}$ is flarbable. The *fleeq* of $\mathcal{C}$ is the circular sequence $\mathcal{E}_\mathcal{C} = e_1, \ldots, e_k$ of edges in E that are crossed by $\mathcal{C}$; we call the edges in $\mathcal{E}_\mathcal{C}$ *fleeq-edges*. A face whose interior is crossed by $\mathcal{C}$ is called a *$\mathcal{C}$-face*. We assume without loss of generality that $\mathcal{C}$ is oriented clockwise and that the edges in $\mathcal{E}_\mathcal{C}$ are ordered according to their intersection with $\mathcal{C}$. Given a flarbable curve $\mathcal{C}$ on G, we present the following definition.

Definition 1. *For a planar graph G and a curve $\mathcal{C}$ flarbable for G, we define a* flarb *operation $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$ which produces a new 3-connected graph $\mathcal{G}(G, \mathcal{C})$ as follows (see Figure 5.1 for a depiction):*

1. *For each edge $e_i = (u_i, v_i)$ in $\mathcal{E}_\mathcal{C}$ such that $u_i \in \textsc{in}(\mathcal{C})$ and $v_i \in \textsc{ex}(\mathcal{C})$, create a new vertex $w_i = \mathcal{C} \cap e_i$ and connect it to v_i along e_i.*
2. *For each pair e_i, e_{i+1} of successive edges in $\mathcal{E}_\mathcal{C}$, create a new edge (w_i, w_{i+1}) between them along $\mathcal{C}$. We call (w_i, w_{i+1}) a* $\mathcal{C}$-edge *(all indices are taken modulo k).*
3. *Delete all vertices of $\textsc{in}(\mathcal{C})$ along with their incident edges.*

Lemma 5.4.1. *For each flarbable curve $\mathcal{C}$ for a 3-regular planar graph G, $\mathcal{G}(G,\mathcal{C})$ has at most 2 more vertices than G does.*

Proof. Let $\mathcal{E}_\mathcal{C} = e_1, \ldots, e_k$ be the fleeq of $\mathcal{C}$ and let f be the new face in $\mathcal{G}(G,\mathcal{C})$ that is bounded by $\mathcal{C}$ and created by the flarb operation $\mathcal{F}(G,\mathcal{E}_\mathcal{C})$. Notice that the vertices of f are the points $w_1, \ldots, w_k$ along edges $e_1, \ldots, e_k$, where $w_i = \mathcal{C} \cap e_i$. Since $\mathcal{C}$ is flarbable, the subgraph induced by the vertices of $\text{IN}(\mathcal{C}) \cup \{w_1, \ldots, w_k\}$ is also a connected graph T with $w_1, \ldots, w_k$ as its leaves and every other vertex of degree 3; see Figure 5.1. Therefore T has at least $k-2$ internal vertices. The flarb operation adds k vertices, namely $w_1, \ldots, w_k$, and the internal vertices of T are deleted. Therefore, the net increase in the number of vertices is at most 2. □

Since each newly created vertex has degree three and all remaining vertices are unaffected, the new graph is 3-regular. In other words, the flarb operation $\mathcal{F}(G,\mathcal{E}_\mathcal{C})$ creates a cycle along $\mathcal{C}$ and removes the portion of the graph enclosed by $\mathcal{C}$. Note that for any point set in general position (no four points lie on the same circle), its Voronoi diagram is a 3-regular planar graph, assuming we use the line at infinity to join the endpoints of its unbounded edges in clockwise order. Therefore, a flarb can be used to represent the changes to the Voronoi diagram upon insertion of a new site.

Observation 5.4.2. *Given a set S of points in general position, let $\mathcal{V}(S)$ be the graph of the farthest-point Voronoi diagram of S. For a new point q, there exists some curve $\mathcal{C}_S^q$ such that $\mathcal{G}(\mathcal{V}(S),\mathcal{C}_S^q) = \mathcal{V}(S)(S \cup \{q\})$; namely, $\mathcal{C}_S^q$ is the boundary of the Voronoi region of q in $\mathcal{V}(S)(S \cup \{q\})$. The same holds for the nearest-point Voronoi diagram of S.*

More generally, convex polytopes defined by the intersection of halfspaces in $\mathbb{R}^3$ behave similarly: the intersection of a new halfspace with a convex polytope modifies the structure of its 1-skeleton by adding a new face. This structural change can be obtained performing a flarb operation in which the flarbable curve consists of the boundary of the new face.

Definition 2. *Given a $\mathcal{C}$-face f of G, the* modified face *of f is the face f' of $\mathcal{G}(G,\mathcal{C})$ that coincides with f outside of $\mathcal{C}$. In other words, f' is the face that remains from*

f after performing the flarb $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$. We say that a $\mathcal{C}$-face f is preserved *(by the flarb $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$) if $|f| = |f'|$. Moreover, we say that each edge in a preserved face is* preserved *(by $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$). Denote by $\mathcal{P}(G, \mathcal{C})$ the set of faces preserved by $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$ and let $\mathcal{B}(G, \mathcal{C})$ be the set of faces wholly contained in the interior of $\mathcal{C}$.*

Since a preserved $\mathcal{C}$-face bounded by two fleeq-edges e_i and e_{i+1} has the same size before and after the flarb, there must be an edge e of G connecting e_i with e_{i+1} which is replaced by a $\mathcal{C}$-edge e^* after the flarb. In this case, we say that the edge e *reappears* as e^*.

The following auxiliary lemma will help us bound the number of operations needed to produce the graph $\mathcal{G}(G, \mathcal{C})$, and follows directly from the Euler characteristic of connected planar graphs:

Lemma 5.4.3. *Let H be a connected planar graph with vertices of degree either 1, 2 or 3. For each $i \in \{1, 2, 3\}$, let δ_i be the number of vertices of H with degree i. Then, H has exactly $2\delta_1 + \delta_2 + 3F_H - 3$ edges, where F_H is the number of bounded faces of H.*

Given a 3-regular graph $G = (V, E)$ and a flarbable curve $\mathcal{C}$ we want to analyze the number of structural changes that G needs to undergo to perform $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$. To this end, we define the combinatorial *cost* of $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$, denoted by $\textsc{cost}(G, \mathcal{C})$, to be the minimum number of links and cuts needed to transform G into $\mathcal{G}(G, \mathcal{C})$ (note that the algorithm may not implement the flarb operation according to the procedure described in Definition 1). We assume that any other operation has no cost and is therefore not included in the cost of the flarb.

Consider the fleeq $\mathcal{E}_\mathcal{C} = e_1, \ldots, e_k$ and the $\mathcal{C}$-edges created by $\mathcal{F}(G, \mathcal{E}_\mathcal{C})$. Let e be an edge adjacent to some e_i and e_{i+1} that reappears as the $\mathcal{C}$-edge e^*. Notice that we can obtain e^* without any links or cuts to G: simply shrink e_i and e_{i+1} so that their endpoints in $\textsc{in}(\mathcal{C})$ now coincide with their intersections with $\mathcal{C}$. Then modify e to coincide with the portion of $\mathcal{C}$ connecting the new endpoints of e_i and e_{i+1}. Using this *preserving operation*, we obtain the $\mathcal{C}$-edge e^* with no cost to the flarb. Intuitively, preserved edges are cost-free in a flarb while non-preserved edges have a nonzero cost. This notion is formalized in the following lemma.

Lemma 5.4.4. *For a flarbable curve $\mathcal{C}$,*

$$(|\mathcal{E}_\mathcal{C}| + |\mathcal{B}(G,\mathcal{C})| - |\mathcal{P}(G,\mathcal{C})|)/2 \leq \textsc{cost}(G,\mathcal{C}) \leq 4|\mathcal{E}_\mathcal{C}| + 3|\mathcal{B}(G,\mathcal{C})| - 4|\mathcal{P}(G,\mathcal{C})|.$$

Proof. For the upper bound, we construct $\mathcal{G}(G,\mathcal{C})$ from G using at most $|\mathcal{E}_\mathcal{C}| + 3|\mathcal{B}(G,\mathcal{C})| - 4|\mathcal{P}(G,\mathcal{C})$ link and cuts. Consider the subgraph $\mathcal{G}_\mathcal{C}$ induced by $\textsc{in}(\mathcal{C}) \cup \{v : v$ is an endpoint to some edge in $\mathcal{E}_\mathcal{C}\}$. Since $\mathcal{C}$ is flarbable, we know that $\mathcal{G}_\mathcal{C}$ is a connected graph such that each vertex of $\textsc{in}(\mathcal{C})$ has degree 3 while the endpoints of the fleeq-edges outside of $\mathcal{C}$ have degree 1. Note that if two preserved faces share a non-fleeq edge e, then there are four neighbors of the endpoints of e that lie outside of $\mathcal{C}$. Since $\mathcal{G}_\mathcal{C}$ is connected, e and its four adjacent edges define the entire graph $\mathcal{G}_\mathcal{C}$ and the bound holds trivially. Therefore, we assume from now on that no two preserved faces share a non-fleeq-edge.

Note that the bounded faces of $\mathcal{G}_\mathcal{C}$ are exactly the bounded faces in $\mathcal{B}(G,\mathcal{C})$. Since $\mathcal{G}_\mathcal{C}$ has $|\mathcal{E}_\mathcal{C}|$ vertices of degree 1, no vertex of degree 2, and $|\mathcal{B}(G,\mathcal{C})|$ bounded faces, by Lemma 5.4.3, $\mathcal{G}_\mathcal{C}$ has at most $2|\mathcal{E}_\mathcal{C}| + 3|\mathcal{B}(G,\mathcal{C})|$ edges. We consider every edge of $\mathcal{G}_\mathcal{C}$ that is not preserved and remove it with a cut operation (we remove isolated vertices afterwards). Note that each preserved face contains at least three preserved edges, two fleeq-edges and a third edge of G. Since we assumed that no two preserved faces share a non-fleeq-edge, the third edge is not double-counted, while the fleeq-edges may be counted at most twice. Therefore, we can charge each preserved face with at least 2 preserved edges, meaning that we perform a total of at most $2|\mathcal{E}_\mathcal{C}| + 3|\mathcal{B}(G,\mathcal{C})| - 2|\mathcal{P}(G,\mathcal{C})|$ cut operations. Note that each non-preserved fleeq-edge has been cut and we need to reintroduce it later to obtain $\mathcal{G}(G,\mathcal{C})$.

Recall that no edge bounding a preserved face has been cut. For each preserved face, perform a preserving operation on it which requires no link or cut operation. Since no two preserved faces share a non-fleeq edge, we obtain all the $\mathcal{C}$-edges bounding the preserved faces after the flarb with no cost. To complete the construction of $\mathcal{G}(G,\mathcal{C})$, create each fleeq-edge that is not preserved and then add the remaining $\mathcal{C}$-edges bounding non-preserved $\mathcal{C}$-faces. Because at least $|\mathcal{P}(G,\mathcal{C})|$ fleeq-edges were preserved, we need to reintroduce at most $|\mathcal{E}_\mathcal{C}| - |\mathcal{P}(G,\mathcal{C})|$ fleeq-edges. Moreover, since only $|\mathcal{E}_\mathcal{C}| - |\mathcal{P}(G,\mathcal{C})|$ $\mathcal{C}$-faces are not preserved, we need to create at most $|\mathcal{E}_\mathcal{C}| - |\mathcal{P}(G,\mathcal{C})|$ $\mathcal{C}$-edges. Therefore, this last step completes the flarb and construct $\mathcal{G}(G,\mathcal{C})$

using a total of at most $2|\mathcal{E}_\mathcal{C}| - 2|\mathcal{P}(G,\mathcal{C})|$ link operations. Consequently, the total number of link and cuts needed to obtain $\mathcal{G}(G,\mathcal{C})$ from G is at most $4|\mathcal{E}_\mathcal{C}|+3|\mathcal{B}(G,\mathcal{C})|-4|\mathcal{P}(G,\mathcal{C})|$ as claimed.

To show that $\textsc{cost}(G,\mathcal{C}) > (|\mathcal{E}_\mathcal{C}| + |\mathcal{B}(G,\mathcal{C})| - |\mathcal{P}(G,\mathcal{C})|)/2$, simply note that in every non-preserved $\mathcal{C}$-face, the algorithm needs to perform at least one cut, either to augment the size or reduce the size of the face. Because G and $\mathcal{C}$ define exactly $|\mathcal{E}_\mathcal{C}| + |\mathcal{B}(G,\mathcal{C})|$ faces, and since in all but $|\mathcal{P}(G,\mathcal{C})|$ of them at least one of its edges must be cut, we conclude that at least $|\mathcal{E}_\mathcal{C}| + |\mathcal{B}(G,\mathcal{C})| - |\mathcal{P}(G,\mathcal{C})|$ edges need to be cut. Since an edge belongs to at most two faces a cut can be over-counted at most twice and the claimed bound holds. □

5.5 The combinatorial upper bound

In this section, we define a potential function to bound the amortized cost of each operation in a sequence of flarb operations. For a 3-regular embedded planar graph $G = (V, E)$, we define two potential functions: a local potential function μ to measure the potential of each face, and a global potential function Φ to measure the potential of the whole graph.

Definition 3. *Let F be the set of faces of a 3-regular embedded planar graph $G = (V, E)$. For each face $f \in F$, let $\mu(f) = \min\{\lceil\sqrt{|V|}\rceil, |f|\}$, where $|f|$ is the number of edges on the boundary of f. The potential $\Phi(G)$ of G is defined as follows:*

$$\Phi(G) = \lambda \sum_{f \in F} \mu(f),$$

for some sufficiently large positive constant λ to be defined later.

Notice that the potential $\mu(f)$ of a $\mathcal{C}$-face f remains unchanged as long as $|f|, |f'| \geq \sqrt{|V|}$, where f' is the modified face of f after the flarb. Since there is no change in potential that we can use within large $\mathcal{C}$-faces, we exclude them from our analysis and focus only on smaller $\mathcal{C}$-faces. We formalize this notion in the following section.

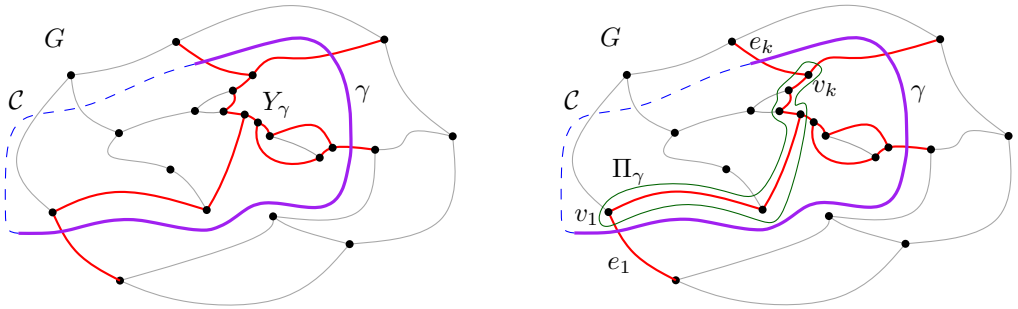

Figure 5.2: Left: A flarbable sub-curves γ is contained in a flarbable curve $\mathcal{C}$. The graph Y_γ is the union of all edges bounding a γ-face. Right: The path Π_γ connects the endpoints of the first and last fleeq-edges crossed by γ by going along the boundary of the outer-face of Y_γ.

5.5.1 Flarbable sub-curves

Given a flarbable curve $\mathcal{C}$, a (connected) curve $\gamma \subseteq \mathcal{C}$ is a *flarbable sub-curve*. Let $\epsilon_\gamma = e_1, \ldots, e_k$ (or simply ϵ) be the set of fleeq-edges intersected by γ given in order of intersection after orienting γ arbitrarily. We call ϵ the *subfleeq* induced by γ. We say that a face is a γ-face if two of its fleeq-edges are crossed by γ (if γ has an endpoint in the interior of this face, it is not a γ-face).

Consider the set of all edges of G intersected or enclosed by $\mathcal{C}$ that bound some γ-face. Since $\mathcal{E}_\mathcal{C}$ is flarbable, these edges induce a connected subgraph Y_γ of G with $|\epsilon| = k$ leaves (vertices of degree 1), namely the endpoints outside of $\mathcal{C}$ of each fleeq-edge in ϵ; see Figure 5.2. Notice that Y_γ may consist of some bounded faces contained in the interior of $\mathcal{C}$. Let H_γ be the set of bounded faces of Y_γ and let δ_2 be the number of vertices of degree 2 of Y_γ. Since Y_γ consists of k vertices of degree 1, Lemma 5.4.3 implies the following result.

Corollary 5.5.1. *The graph Y_γ consists of exactly $2k + \delta_2 + 3|H_\gamma| - 3$ edges.*

Recall that a $\mathcal{C}$-face f is preserved if its corresponding modified face f' in $\mathcal{G}(G, \mathcal{C})$ has the same number of edges, i.e., $|f'| = |f|$. We say that f is *augmented* if $|f'| = |f| + 1$ and we call f *shrinking* if $|f'| < |f|$. Notice that these are all the possible cases as f gains at most one new edge during the flarb, namely the $\mathcal{C}$-edge crossing this face.

In the context of a particular flarbable sub-curve γ, let a_γ, s_γ and p_γ be the number

of augmented, shrinking and preserved γ-faces, respectively (or simply a, s and p if γ is clear from the context). We further differentiate among the s shrinking γ-faces. A shrinking γ-face is *interior* if it contains no vertex of degree 2 of Y_γ and does not share an edge with an augmented γ-face. Let s_a be the number of shrinking γ-faces that share an edge with an augmented face, let s_b be the number of shrinking γ-faces not adjacent to an augmented face that have a vertex of degree 2 of Y_γ, and let s_c be the number of interior shrinking γ-faces. Therefore, $s = s_a + s_b + s_c$ is the total number of shrinking γ-faces.

Since each augmented face has at most two edges and because there are a augmented faces, we know that $s_a \leq 2a$. Let v_1 and v_k be the endpoints of the edges e_1 and e_k that lie inside $\mathcal{C}$. Let Π_γ be the unique path connecting v_1 and v_k in Y_γ that traverses along the boundary of the outer face of Y_γ and stays in the interior of $\mathcal{C}$; see Figure 5.2.

Notice that Π_γ contains all the edges of γ-faces that may bound a γ'-face for some other flarbable sub-curve γ' disjoint from γ. In the end, we aim to have bounds on the number of edges that will be removed from the γ-faces during the flarb, but some of these edges may be double-counted if they are shared with a γ'-face. Therefore, we aim to bound the length of Π_γ and count precisely these possible-shared edges.

Lemma 5.5.2. *The path Π_γ has length at most $k + 3|H_\gamma| + \delta_2 - a - s_c$.*

Proof. Notice that no fleeq-edge can be part of Π_γ or this path would go outside of $\mathcal{C}$, i.e., there are k fleeq-edges of Y_γ that cannot be part of Π_γ.

We say that a vertex is *augmented* if it is incident to two fleeq-edges and a third edge that is not part of ϵ, which we call an *augmented edge*. Because each augmented γ-face has exactly one augmented vertex, there are exactly a augmented vertices in Y_γ. Moreover, Π_γ contains at most 2 augmented vertices (if v_1 or v_k is augmented). Thus, at most two augmented edges can be traversed by Π_γ and hence, at least $a - 2$ augmented edges of Y_γ do not belong to Π_γ.

Let f be an internal shrinking γ-face. Since f is not adjacent to an augmented γ-face, it has no augmented edge on its boundary. We claim that f has at least one edge that is not traversed by Π_γ. If this claim is true, then there are at least s_c non-fleeq non-augmented edges that cannot be used by Π_γ—one for each internal

shrinking γ-face. Thus, since Y_γ consists of $2k + 3|H_\gamma| + \delta_2 - 2$ edges, the number of edges in Π_γ is at most

$$2k + 3|H_\gamma| + \delta_2 - 2 - (k + a - 2 + s_c) = k + 3|H_\gamma| + \delta_2 - a - s_c.$$

It remains to show that each internal shrinking γ-face f has at least one non-fleeq edge that is not traversed by Π_γ. If Π_γ contains no edge on the boundary of f, then the claim holds trivially. If Π_γ contains exactly one edge of f, then since f is shrinking, it has at least 4 edges and two of them are not fleeq-edges. Thus, in this case there is one edge of f that is not traversed by Π_γ. We assume from now on that Π_γ contains at least two edges of f.

We claim that that Π_γ needs to visit a contiguous sequence of edges along the boundary of f. To see this, note that each face of Y_γ lying between Π_γ and the boundary of f cannot be crossed by $\mathcal{C}$. Therefore, if we consider the first edge of Π_γ that is not on f after visiting f the first time, the this edge is incident to the outer face of Y_γ and the only face of Y_γ that it is incident with does not intersect $\mathcal{C}$. This is a contradiction, since this edge should not be part of Y_γ by definition. Therefore, Π_γ visits a contiguous sequence of edges along f.

If Π_γ visits 2 consecutive edges of f, then the vertex in between them must have degree 2 in Y_γ, as the two edges are incident to the outer face—a contradiction since f is an internal shrinking face with no vertex of degree 2. Consequently, if f is an internal shrinking face, it has always at least one non-fleeq edge that is not traversed by Π_γ. □

5.5.2 How much do faces shrink in a flarb?

In order to analyze the effect of the flarb operations on flarbable sub-curves, we think of each edge as consisting of two *half-edges*, each adjacent to one of the two faces incident to this edge. For a given edge, the algorithm may delete its half-edges during two separate flarbs of differing flarbable sub-curves.

We define the operation $\mathcal{F}(G, \gamma)$ to be the operation which executes steps 1 and 2 of the flarb on the flarbable sub-curve γ and then deletes each half-edge with both endpoints in $\text{IN}(\mathcal{C})$ adjacent to a γ-face. Since $\mathcal{F}(G, \gamma)$ removes and adds half-edges,

we are interested in bounding the net balance of half-edges throughout the flarb. To do this, we measure the change in size of a face during the flarb.

Recall that a, s and p are the number of augmented, shrinking and preserved γ-faces, respectively. The following result provides a bound on the total "shrinkage" of the faces crossed by a given flarbable sub-curve.

Theorem 5.5.3. *Given a flarbable curve $\mathcal{C}$ on G and a flarbable sub-curve γ crossing the fleeq-edges $\epsilon = e_1, \ldots, e_k$, let $f_1, \ldots, f_k$ be the sequence of γ-faces and let $f'_1, \ldots, f'_k$ be their corresponding modified faces after the flarb $\mathcal{F}(G, \gamma)$. Then,*

$$\sum_{i=1}^{k} (|f_i| - |f'_i|) \geq s/2. \tag{5.1}$$

Proof. Recall that no successive γ-faces can both be augmented unless $\mathcal{E}_{\mathcal{C}}$ consists of three edges incident to a single vertex. In this case, at most 3 γ-faces can be augmented, so $\sum_{i=1}^{k}(|f_i| - |f'_i|) = 3$ and the result holds trivially; hence, we assume from now on that no two successive faces are both augmented.

Let Δ be the number of half-edges removed during $\mathcal{F}(G, \gamma)$. Notice that to count how much a face f_i shrinks when becoming f'_i after the flarb, we need to count the number of half-edges of f_i that are deleted and the number that are added in f'_i. Since exactly one half-edge is added in each f'_i, we know that $\sum_{i=1}^{k}(|f_i| - |f'_i|) = \Delta - k$. We claim that $\Delta \geq k + s/2$. If this claim is true, then $\sum_{i=1}^{k}(|f_i| - |f'_i|) \geq s/2$ implying the theorem. In the remainder of this proof, we show this bound on Δ.

Let $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ be the subgraph of Y_γ obtained by removing its k fleeq-edges. Therefore, we know that $|E_{\mathcal{T}}| = k + 3|H_\gamma| + \delta_2 - 3$ by Corollary 5.5.1. To have a precise counting of Δ, notice that for some edges of $\mathcal{T}$, $\mathcal{F}(G, \gamma)$ removes only one of their half-edges and for others it will remove both of them. Since the fleeq-edges are present in each of the faces $f_1, \ldots, f_k$ before and after the flarb, we get that

$$\Delta = 2|E_{\mathcal{T}}| - S_{\mathcal{T}}, \tag{5.2}$$

where $S_{\mathcal{T}}$ denotes the number of edges in $\mathcal{T}$ with only one half-edge incident to a face of $f_1, \ldots, f_k$.

Note that the edges of $S_{\mathcal{T}}$ are exactly the edges on the path Π_γ bounded in Lemma 5.5.2. Therefore, $S_{\mathcal{T}} \leq k + 3|H_\gamma| + \delta_2 - a - s_c$. By using this bound in (5.2),

we get

$$\Delta \geq 2(k + 3|H_\gamma| + \delta_2 - 3) - (k + 3|H_\gamma| + \delta_2 - a - s_c) = k + 3|H_\gamma| + \delta_2 + a + s_c - 6.$$

Since each shrinking γ-face accounted for by s_b has a vertex of degree 2 in Y_γ, we know that $\delta_2 \geq s_b$. Moreover, $s_a \leq 2a$ as each shrinking γ-face can be adjacent to at most two augmented γ-faces. Therefore, since $s = s_a + s_b + s_c$, we get that $\Delta \geq k + 3|H_\gamma| + s_a/2 + s_b + s_c \geq k + s/2$, where s is the number of shrinking γ-faces proving the claimed bound on Δ. □

5.5.3 Flarbable sequences

Let $\mathcal{G}^0 = G$. A sequence of curves $\mathscr{C} = \mathcal{C}_1, \ldots, \mathcal{C}_k$ is *flarbable* if for each $i \in [k]$, $\mathcal{C}_i$ is a flarbable on

$$\mathcal{G}^i = \mathcal{G}(\mathcal{G}^{i-1}, \mathcal{C}_i).$$

As a notational shorthand, let $\mathcal{F}^i$ denote the flarb operation $\mathcal{F}(\mathcal{G}^{i-1}, \mathcal{C}_i)$ when $\mathscr{C}$ is a flarbable sequence for G.

Theorem 5.5.4. *For a 3-regular planar graph $G = (V, E)$ and some flarbable sequence $\mathscr{C} = \mathcal{C}_1, \ldots, \mathcal{C}_N$ of flarbable fleeqs, for all $i \in [N]$,*

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq O(\sqrt{|V_i|}),$$

where V_i is the set of vertices of $\mathcal{G}^i$.

Proof. Split $\mathcal{C}_i$ into smaller curves $\gamma_1, \ldots, \gamma_h$ such that for all $j \in [h]$, γ_j is a maximal curve contained in $\mathcal{C}_i$ that does not intersect the interior of a face with more than $\sqrt{|V_i|}$ edges (we ignore the portion of $\mathcal{C}_i$ inside this large faces). Since there can be at most $\sqrt{|V_i|}$ faces of size $\sqrt{|V_i|}$, we know that $h \leq \sqrt{|V_i|}$. Let ϵ_j be the subfleeq containing each fleeq-edge crossed by γ_j. Let a_j, s_j and p_j be the number of augmented, shrinking and preserved γ_j-faces, respectively. Notice that $|\epsilon_j| = a_j + s_j + p_j + 1$. Moreover, since each augmented face is adjacent to a shrinking face, we know that $a_j \leq s_j + 1$. Therefore, $|\epsilon_j| \leq 2s_j + p_j + 2$.

Let $\mathcal{L}_i$ be the set of $\mathcal{C}_i$-faces with at least $\sqrt{|V_i|}$ edges and let ω_i be the set of all faces of $\mathcal{G}^{i-1}$ completely enclosed in the interior of $\mathcal{C}_i$.

First, we upper bound $\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i)$. By Lemma 5.4.4, we know that

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) \leq 4|\mathcal{E}_{\mathcal{C}_i}| + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \tag{5.3}$$

$$= 4\sum_{j=1}^{h} |\epsilon_j| + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \tag{5.4}$$

$$\leq 4\sum_{j=1}^{h} (2s_j + p_j + 2) + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \tag{5.5}$$

Because each preserved face is crossed by exactly one flarbable sub-curve, $\sum_{j=1}^{h} p_j = |\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. Therefore,

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) \leq 4\sum_{j=1}^{h} (2s_j + 2) + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| = 8h + 8\sum_{j=1}^{h} s_j + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \ .$$

Since $h \leq \sqrt{|V_i|}$, we conclude that

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) \leq 8\sqrt{|V_i|} + 8\sum_{j=1}^{h} s_j + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \ . \tag{5.6}$$

Next, we upper bound the change in potential $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1})$. Given a flarbable curve or sub-curve γ, let $\mathcal{A}(\gamma)$ denote the set of γ-faces. Recall that for a γ-face $f \in \mathcal{A}(\gamma)$, f' is the modified face of f. Also, let f_n be the new face created by $\mathcal{F}^i$, i.e., the face of $\mathcal{G}^i$ bounded by $\mathcal{C}_i$. Recall that for each face $f \in \mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)$, f disappears and there is a drop in potential of $\mu(f) \geq 1$. Using this, we can break up the summation so that

$$\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) = \mu(f_n) + \lambda \sum_{f \in \mathcal{A}(\mathcal{C}_i)} (\mu(f') - \mu(f)) - \lambda \sum_{f \in \mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)} \mu(f) \tag{5.7}$$

$$\leq \mu(f_n) + \lambda \sum_{f \in \mathcal{A}(\mathcal{C}_i)} (\mu(f') - \mu(f)) - \lambda|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \ . \tag{5.8}$$

We now break up the first summation by independently considering the large faces in $\mathcal{L}_i$ and the remaining smaller faces which are crossed by some flarbable sub-curve.

$$\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \mu(f_n) + \lambda \sum_{j=1}^{h} \left(\sum_{f \in \mathcal{A}(\gamma_j)} (\mu(f') - \mu(f)) \right) \tag{5.9}$$

$$+ \lambda \sum_{f \in \mathcal{L}_i} (\mu(f') - \mu(f)) - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|. \tag{5.10}$$

Since each face can gain at most one edge, in particular we know that $\mu(f') - \mu(f) \leq 1$ for each $f \in \mathcal{L}_i$. Moreover, $\mu(f_n) \leq \sqrt{|V_i|}$ by definition. Thus,

$$\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \sqrt{|V_i|} + \lambda \sum_{j=1}^{h} \left(\sum_{f \in \mathcal{A}(\gamma_j)} (\mu(f') - \mu(f)) \right) + \lambda |\mathcal{L}_i| - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|.$$

Note that $\mu(f) = |f|$ for each face $f \in \mathcal{A}(\gamma_j)$, $1 \leq j \leq h$. Thus, applying Theorem 5.5.3 to the first summation, we get

$$\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \sqrt{|V_i|} - \frac{\lambda}{2} \sum_{j=1}^{h} s_j + \lambda |\mathcal{L}_i| - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|.$$

Since there can be at most $\sqrt{|V_i|}$ faces of size $\sqrt{|V_i|}$, we know that $|\mathcal{L}_i| \leq \sqrt{|V_i|}$. Therefore,

$$\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq (\lambda + 1)\sqrt{|V_i|} - \frac{\lambda}{2} \sum_{j=1}^{h} s_j - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \tag{5.11}$$

Putting (5.6) and (5.11) together, we get that

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq (\lambda + 9)\sqrt{|V_i|} + (8 - \frac{\lambda}{2}) \sum_{j=1}^{h} s_j + (3 - \lambda)|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$$

By letting λ be a sufficiently large constant (namely $\lambda = 16$), we get that

$$\textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) = O(\sqrt{|V_i|}). \qquad \square$$

Corollary 5.5.5. *Let G be a 3-regular plane graph with ν vertices. For a sequence $\mathscr{C} = \mathcal{C}_1, \ldots, \mathcal{C}_N$ of flarbable fleeqs for graph $G = (V, E)$ where $\nu = |V|$,*

$$\sum_{i=1}^{N} \textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) = O(\nu + N\sqrt{\nu + N}),$$

where V_i is the set of vertices of $\mathcal{G}^i$.

Proof. Using the result of Theorem 5.5.4, we can write

$$\sum_{i=1}^{N} \textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^N) - \Phi(G) = O(N\sqrt{|V_i|}).$$

Because $\Phi(G) = \lambda \sum_{f \in F} \mu(f)$, we know that $\Phi(G) = O(\nu)$. Analogously, since each flarb operation adds at most 2 vertices by Lemma 5.4.1, we know that the number of vertices in $\mathcal{G}^N$ is $O(\nu+N)$ which, in turn, implies that $\Phi(\mathcal{G}^N) = O(\nu+N)$. Therefore,

$$\sum_{i=1}^{N} \textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) = O(N\sqrt{|V_i|} + \Phi(G) - \Phi(\mathcal{G}^N)) = O(\nu + N\sqrt{\nu+N}) \qquad \square$$

5.6 The lower bound

In Section 5.6, we present an example of a 3-regular Halin graph G with ν vertices—a tree with all leaves connected by a cycle to make it 3-regular—and a corresponding flarb operation with cost $\Omega(\sqrt{\nu})$ that yields a graph isomorphic to G. Because this sequence can be repeated, the amortized cost of a flarb is $\Theta(\sqrt{\nu})$.

Let $\nu = 2k(k+1)-2$ for some positive integer k. The construction of the 3-regular graph with ν vertices is depicted in Figure 5.3. In this graph, we show the existence of a flarbable curve $\mathcal{C}$ (dashed in the figure) such that the flarb operation on G produces a graph $\mathcal{G}(G, \mathcal{C})$ isomorphic to G. Moreover, $\mathcal{C}$ crosses at least k augmented $\mathcal{C}$-faces and k shrinking $\mathcal{C}$-faces. Therefore, $\textsc{cost}(G, \mathcal{C}) \geq k = \Omega(\sqrt{\nu})$ by Lemma 5.4.4. Since we end up with a graph that is isomorphic to the original, we can produce a new flarbable curve having the same effect. That is, there is a sequence of N flarbable curves $\mathcal{C}_1, \ldots, \mathcal{C}_N$ such that $\sum_{i=1}^{N} \textsc{cost}(\mathcal{G}^{i-1}, \mathcal{C}_i) = \Omega(N\sqrt{\nu})$,

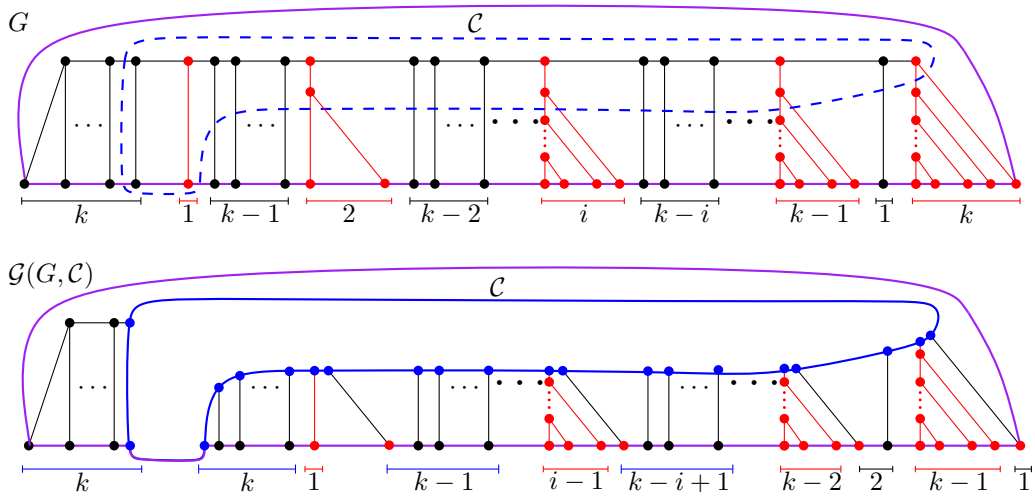

Figure 5.3: A 3-regular graph G with $\nu = 2k(k+1) - 2$ vertices. A flarbable curve $\mathcal{C}$ induces a flarb such that $\mathcal{G}(G, \mathcal{C})$ is isomorphic with G.

5.7 Computing the flarb

In this section, we describe a data structure to maintain the farthest-point Voronoi diagram of a set S of n sites in convex position as new sites are added to S. Our structure allows us to find the edges of each preserved face and ignore them, thereby focusing only on necessary modifications to the combinatorial structure. The time we spend in these operations is then proportional to the number of non-preserved edges. Since this number is proportional to the cost of the flarb, our data structure supports site insertions in time that is almost optimal (up to a polylogarithmic factor).

5.7.1 Grappa trees

Grappa trees [8] are a modification of link-cut trees, a data structure introduced by Sleator and Tarjan [86] to maintain the combinatorial structure of trees. They support the creation of new isolated vertices, the *link* operation which adds an edge between two vertices in disjoint trees, and the *cut* operation which removes an edge, splitting a tree into two trees.

We use this structure to maintain the combinatorial structure of the farthest-point Voronoi diagram $\mathcal{V}(S)$ of a set S of sites in convex position throughout the incremental construction. Recall that each insertion defines a flarbable curve $\mathcal{C}$, namely the

boundary of the Voronoi region of the inserted site. Our algorithm performs this flarb operation in time $O(\textsc{cost}(\mathcal{V}(S), \mathcal{C}) \log^7 n)$, where n is the number of vertices inserted so far. That is, we obtain an algorithm whose running time depends on the minimum number of link and cut operations that the farthest-point Voronoi diagram, which is a tree, must undergo after each insertion. Moreover, this Voronoi diagram answers nearest neighbor queries in $O(\log n)$ time.

A grappa tree, as introduced by Aronov et al. [8], is a data structure is based on the worst-case version of the link-cut tree construction of Sleator and Tarjan [86]. This structure maintains a forest of fixed-topology trees subject to many operations, including Make-Tree, Link, and Cut, each in $O(\log n)$ worst-case time while using $O(n)$ space.

As in [8, 86], we decompose a rooted binary tree into a set of maximal vertex-disjoint downward paths, called *heavy paths*, connected by tree edges called *light edges*. Each heavy path is in turn represented by a biased binary tree whose leaf-nodes correspond to the vertices of the heavy path. Non-leaf nodes represent edges of this heavy path, ordered in the biased tree according to their depth along the path. Therefore, vertices that are higher (closer to the root) in the path correspond to leaves farther left in the biased tree. Each leaf node ℓ of a biased tree B represents an internal vertex v of the tree which has a unique light edge l_v adjacent to it. We keep a pointer from ℓ to this light edge. Note that the other endpoint of l_v is the root of another heavy path which in turn is represented by another biased tree, say B'. We merge these two biased trees by adding a pointer from ℓ to the root of B'. After merging all the biased trees in this way, we obtain the grappa tree of a tree T. A node of the grappa tree that is non-leaf in its biased tree represents a heavy edge and has two children, whereas a node that is a leaf of its biased tree represents a vertex of the heavy path (and its unique adjacent light edge) and has only one child. By a suitable choice of paths and biasing, as described in [86], the grappa-tree has height $O(\log n)$.

In addition, grappa trees allow us to store left and right marks on each of its nodes, i.e., on each edge of T. To assign the mark of a node, grappa trees support the $O(\log n)$-time operation Left-Mark(T, v, m_l) which sets the mark m_l to every

edge in the path from v to the super root of T (RIGHT-MARK(T, v, m_l) is defined analogously). In our setting, we use the marks of an edge e to keep track of the faces adjacent to this edge in a geometric embedding of T. Since T is rooted, we can differentiate between the left and the right faces adjacent to e.

The following definition formalizes the operations supported by a Grappa-tree.

Definition 4. *Grappa trees solve the following data-structural problem: maintain a forest of rooted binary trees with specified topology subject to:*

$T =$ Make-Tree(v): *Create a new tree T with a single internal vertex v (not previously in another tree).*

$T =$ Link(v, w): *Given a vertex v in one tree T_v and the root w of a different tree T_w, connect v and w and merge T_v with T_w into a new tree T.*

$(T_1, T_2) =$ Cut(e): *Delete the existing edge $e = (v, w)$ in tree T, splitting into T two trees T_1 and T_2 containing v and w, respectively.*

Evert(v): *Make external node v the root of its tree, reversing the orientation (which endpoint is closer to the root) of every edge along the root-to-v path.*

Left-Mark(T, v, m_ℓ): *Set the left mark of every edge on the root-to-v path in T to the new mark m_ℓ, overwriting the previous left marks of these edges.*

Right-Mark(T, v, m_r): *Set the right mark of every edge on the root-to-v path in T to the new mark m_r, overwriting the previous right marks of these edges.*

$(e, m_\ell^*, m_r^*) =$ Oracle-Search(T, O_e): *Search for the edge e in tree T. The data structure can find e only via* oracle queries*: given two incident edges f and f' in T, the provided oracle $O_e(f, f', m_\ell, m_r, m'_\ell, m'_r)$ determines in constant time which "side" of f contains e, i.e., whether e is in the component of $T - f$ that contains f', or in the rest of the tree (which includes f itself). The data structure provides the oracle with the left mark m_ℓ and the right mark m_r of edge f, as well as the left mark m'_ℓ and the right mark m'_r of edge f', and at the end, it returns the left mark m_ℓ^* and the right mark m_r^* of the found edge e.*

Theorem 5.7.1. *[Theorem 7 from [8]] A grappa-tree maintains the combinatorial structure of a forest and supports each operation described above in $O(\log n)$ worst-case time per operation, where n is the total size of the trees affected by the operation.*

5.7.2 The Voronoi diagram

Let S be a set of n sites in convex position and let $\mathcal{V}(S)$ be the binary tree representing the farthest-point Voronoi diagram of S. We store $\mathcal{V}(S)$ using a grappa tree. In addition, we assume that each edge of $\mathcal{V}(S)$ has two *face-markers*, its left and right markers which respectively store the site of S whose Voronoi region is adjacent to this edge. While a grappa tree stores only the topological structure of $\mathcal{V}(S)$, with the aid of the face-markers we can retrieve the geometric representation of $\mathcal{V}(S)$. Namely, for each vertex v of $\mathcal{V}(S)$, we can look at its adjacent edges and their face-markers to retrieve the point in the plane representing the location of v in the farthest-point Voronoi diagram of S in $O(1)$ time. Therefore, we refer to v also as a point in the plane. Recall that each vertex v of $\mathcal{V}(S)$ is the center of a circle that passes through at least three sites of S, we call these sites the *definers* of v and we call this circle the *definer circle* of v.

Observation 5.7.2. *Given a new site q in the plane such that $S' = S \cup \{q\}$ is in convex position, the vertices of $\mathcal{V}(S)$ that are farther from q than from any other site of S' are exactly the vertices whose definer circle does not contain q.*

Let q be a new site such that $S' = S \cup \{q\}$ is in convex position. Let $R(q, S')$ be the Voronoi region of q in the Voronoi diagram of S' and let $\partial R(q, S')$ denote its boundary. Recall that we can think of $\mathcal{V}(S)$ as a Halin graph by connecting all its leaves by a cycle to make it 3-regular. While we do not explicitly use this cycle, we need it to make our definitions consistent. In this Halin graph, the curve $\partial R(q, S')$ can be made into a closed curve by going around the leaf of $\mathcal{V}(S)$ contained in $R(q, S')$; namely the point at infinity of the bisector between the two neighbors of q along the convex hull of S'. In this way, $\partial R(q, S')$ becomes a flarbable curve. Therefore, we are interested in performing the flarb operation it induces which leads to a transformation of $\mathcal{V}(S)$ into $\mathcal{V}(S')$.

5.7.3 Heavy paths in Voronoi diagrams

Recall that for the grappa tree of $\mathcal{V}(S)$, we computed a heavy path decomposition of $\mathcal{V}(S)$. In this section, we first identify the portion of each of these heavy paths that lies inside $R(q, S')$. Once this is done, we test if any edge adjacent to an endpoint of these paths is preserved. Then within each heavy path, we use the biased trees built on it to further find whether there are non-preserved edges inside this heavy path. After identifying all the non-preserved edges, we remove them which results in a split of $\mathcal{V}(S)$ into a forest where each edge inside $R(q, S')$ is preserved. Finally, we show how to link back the disjoint components into the tree resulting from the flarb operation.

We first find the heavy paths of $\mathcal{V}(S)$ whose roots lie in $R(q, S')$. Additionally, we find the portion of each of these heavy paths that lies inside $R(q, S')$.

Recall that there is a leaf ρ of $\mathcal{V}(S)$ that lies in $R(q, S')$ being the point at infinity of the bisector between the two neighbors of q along the convex hull of S'. As a first step, we root $\mathcal{V}(S)$ at ρ by calling Evert(ρ). In this way, ρ becomes the root of $\mathcal{V}(S)$ and all the heavy paths have a *root* which is their endpoint closest to ρ.

Let Ψ be the set the of roots of all heavy paths of $\mathcal{V}(S)$, and let $\Psi_q = \{r \in \Psi : r \in R(q, S')\}$. We focus now on computing the set Ψ_q. By Observation 5.7.2, each root in Ψ_q has a definer circle that contains q. We use a dynamic data structure that stores the definer circles of the roots in Ψ and returns those circles containing a given query point efficiently.

Lemma 5.7.3. *There is a fully dynamic $O(n)$-space data structure to store a set of circles (not necessarily with equal radii) that can answer queries of the form: Given a point q in the plane, return a stored circle that does not contain q, where insertions take $O(\log^3 n)$ amortized time, deletions take $O(\log^6 n)$ amortized time, and queries take $O(\log^2 n)$ worst-case time.*

Proof. Chan [26] presented a fully dynamic randomized data structure that can answer queries about the convex hull of a set of n points in three dimensions, where insertions take $O(\log^3 n)$ amortized time, deletions take $O(\log^6 n)$ amortized time, and extreme-point queries take $O(\log^2 n)$ worst-case time. We use this structure to

solve our problem, but first, we need to transform our input into an instance that can be handled by this data structure.

Let $\mathscr{C}$ be the dynamic set of circles we want to store. Consider the paraboloid-lifting which maps every point $(x, y) \mapsto (x, y, x^2 + y^2)$. Using this lifting, we identify each circle $C \in \mathscr{C}$ with a plane π_C in $\mathbb{R}^3$ whose intersection with the paraboloid projects down as C in the xy-plane. Moreover, a point $q = (x, y)$ lies outside of C if and only if point $(x, y, x^2 + y^2)$ lies above the plane π_C.

Let $\Pi = \{\pi_C : C \in \mathscr{C}\}$ be the set of planes corresponding to the circles in $\mathscr{C}$. In the above setting, our query can be translated as follows: Given a point $q' = (x, y, x^2+y^2)$ on the paraboloid, find a plane $\pi_C \in \Pi$ that lies below q'.

Using standard point-plane duality in $\mathbb{R}^3$, we can map the set of planes Π to a point set Π^*, and a query point q' to a plane q^* such that a query translates to a *plane query*: Given a query plane q^*, find a point of Π^* that lies above it.

Using the data structure introduced by Chan [26] to store Π^*, we can answer plane queries as follows. Consider the direction orthogonal to q^* pointing in the direction above q^*. Then, find the extreme point of the convex hull of Π^* in this direction in $O(\log^2 n)$ time. If this extreme point lies above q^*, then we return the circle of $\mathscr{C}$ corresponding to it. Otherwise, we return that no point of Π^* lies above q^* which implies that no circle of $\mathscr{C}$ excludes q. Insertions take $O(\log^3 n)$ time while removals from the structure take $O(\log^6 n)$ time. □

For our algorithm, we store each root in Ψ into the data structure given by Lemma 5.7.3. Notice that after the insertion or deletion of an edge in the grappa tree, there may be some new heavy paths created, and we need to insert their roots in our data structure. Using this structure, we obtain the following result.

Lemma 5.7.4. *We can compute each root in Ψ_q in total $O(|\Psi_q| \log^6 n)$ amortized time.*

Proof. After querying for a root whose definer circle excludes q, we remove it from the data structure and query it again to find another root with the same property until no such root exists. Since queries and removals take $O(\log^2 n)$ and $O(\log^6 n)$ time, respectively, we can find all roots in Ψ_q in $O(|\Psi_q| \log^6 n)$ time. □

Given a root $r \in \Psi$, let h_r be the heavy path whose root is r. Because the portion of $\mathcal{V}(S)$ that lies inside $R(q, S')$ is a connected subtree, we know that, for each $r \in \Psi_q$, the portion of the path h_r contained in $R(q, S')$ is also connected. In order to compute this connected subpath, we want to find the last vertex of h_r that lies inside of $R(q, S')$, or equivalently, the unique edge of h_r having exactly one endpoint in the interior of $R(q, S')$. We call such an edge the *q-transition edge* of h_r (or simply *transition edge*).

Lemma 5.7.5. *For a root $r \in \Psi_q$, we can compute the transition edge of h_r in $O(\log n)$ time.*

Proof. Let e_r be the transition edge of h_r. We make use of the oracle search proper of a grappa-tree to find the edge e_r. To this end, we must provide the data structure with an oracle such that: given two incident edges f and f' in $\mathcal{V}(S)$, the oracle determines in constant time which side of f contains the edge e_r, i.e., whether e_r is in the component of $\mathcal{V}(S) \setminus f$ that contains f', or in the rest of the tree (which includes f itself). The data structure provides the oracle with the left and the right marks of f and f'. Given such an oracle, a grappa-tree allows us to find the edge e_r in $O(\log n)$ time by Theorem 5.7.1.

Given two adjacent edges f and f' of $\mathcal{V}(S)$ that share a vertex v, we implement the oracle described above as follows. Recall that the left and right face-marks of f and f' correspond to the sites of S whose Voronoi region is incident to the edges f and f'. Thus, we can determine the definers of the vertex v, find their circumcircle, and test whether q lies inside it or not in constant time. Thus, by Observation 5.7.2, we can test in $O(1)$ time whether v lies in Ψ_q or not and hence, decide if e_r is in the component of $\mathcal{V}(S) \setminus f$ that contains f', or in the rest of the tree. □

5.7.4 Finding non-preserved edges

Observation 5.7.6. *Given a 3-regular graph G and a flarbable curve $\mathcal{C}$, if we can test whether a point is enclosed by $\mathcal{C}$ in $O(1)$ time, then we can test whether an edge is preserved in $O(1)$ time.*

Proof. First note that we can test in $O(1)$ time whether an edge reappears by testing whether its two adjacent edges are fleeq-edges. Since a preserved edge is either an

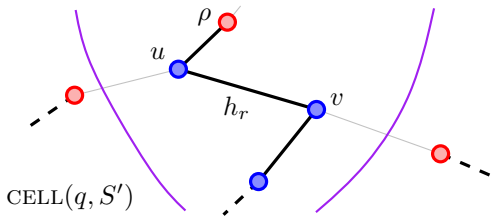

Figure 5.4: Path h_r contains two adjacent vertices u and v such that the light edge of u is a left edge while the light edge of v is a right edge. The edge uv cannot be preserved.

edge that reappears or a fleeq-edge adjacent to an edge that reappears, this takes only $O(1)$ time. □

Let $\mathcal{V}_q(S)$ be the subtree induced by all the edges of $\mathcal{V}(S)$ that intersect $R(q, S')$. Now, we work towards showing how to identify each non-preserved edge of $\mathcal{V}_q(S)$ in the fleeq induced by $\partial R(q, S')$. For each root $r \in \Psi_q$, we compute the transition edge e_r of h_r using Lemma 5.7.5 in $O(\log n)$ time per edge. Assume that w is the vertex of e_r that is closer to r (or is equal to r). We consider each edge adjacent to w and test whether or not it is preserved. Since each vertex of $\mathcal{V}_q(S)$ has access to its definers via the faces markers of its adjacent edges, we can test if this vertex lies in $R(q, S')$. Thus, by Observation 5.7.6, we can decide whether an edge of $\mathcal{V}_q(S)$ is preserved in $O(1)$ time.

We mark each non-preserved edge among them as *shadow*. Because we can test if an edge is preserved in $O(1)$ time, and since computing e_r takes $O(\log n)$ time by Lemma 5.7.5, this can be done in total amortized $O(|\Psi_q| \log n)$ time. In addition, notice that if h_r contains two adjacent vertices u and v such that the light edge of u is a left edge while the light edge of v is a right edge (or vice versa), then the edge uv cannot be preserved; see Figure 5.4. In this case, we say that uv is a *bent edge*. We want to mark all the bent edges in $\mathcal{V}_q(S)$ as shadow, but first we need to identify them efficiently.

Note that it suffices to find all the bent edges of h_r for a given root $r \in \Psi_q$, and then repeat this process for each root in Ψ_q. To find the bent edges in h_r, we further extend the grappa-tree in such a way that the biased tree representing h_r allows us to search for bent edges in $O(\log n)$ time. This extension is described as follows. Recall that each leaf s_v of a biased tree corresponds to a vertex v of the heavy path and has

a pointer to the unique light edge adjacent to v. Since each light edge is either left or right, we can extend the biased tree to allow us to search in $O(\log n)$ time for the first two consecutive leaves where a change in direction occurs. From there, standard techniques allow us to find the next change in direction in additional $O(\log n)$ time. Therefore, we can find all the bent edges of a heavy path h_r in $O(\log n)$ time per bent edge. After finding each bent edge in h_r, we mark it is as a shadow edge.

Lemma 5.7.7. *An edge of $\mathcal{V}_q(S)$ is a preserved edge if and only if it was not marked as a shadow edge.*

Proof. Since we only mark non-preserved edges as shadow, we know that if an edge is preserved, then it is not shadow.

Assume that there is a non-preserved edge uv of $\mathcal{V}_q(S)$ that is not marked as shadow. If uv is a heavy edge, then it belongs to some heavy path h_r for some $r \in \Psi_q$. We know that uv cannot be the transition edge of h_r since it would have been shadowed when we tested if it was preserved. Thus, uv is completely contained in $R(q, S')$. We can also assume that uv is no bent edge, otherwise uv would have been shadowed. Therefore, the light children of u and v are either both left or both right children, say left. Since uv is not preserved, either the light child of u or the light child of v must be inside $R(q, S')$. Otherwise if both edges cross the boundary of $R(q, S')$, then uv is preserved by definition.

Assume that u has a light left child r' that is inside $R(q, S')$. That is, r' must be the root of some heavy path and hence belongs to Ψ_q. However, in this case we would have checked all the edges adjacent to u while processing the root $r' \in \Psi_q$. Therefore, every edge that is non-shadow and intersects $R(q, S')$ is a preserved edge. □

Let σ be the number of shadow edges of $\mathcal{V}(S)$, which is equal to the number of non-preserved edges by Lemma 5.7.7.

Corollary 5.7.8. *It holds that $\sigma = \Theta(\textsc{cost}(\mathcal{V}(S), \partial R(q, S')))$.*

The following relates the size of Ψ_q with the value of σ.

Lemma 5.7.9. *It holds that $|\Psi_q| = O(\sigma \log n)$.*

Proof. Given a root r of Ψ_q, let p_r be the parent of r and notice that the edge rp_r is a light edge that is completely contained in Ψ_q. Note that p_r belongs to another heavy path h_t, for some $t \in \Psi_q$. If p_r is the endpoint of the transition edge of h_t closest to the root, then we add a *dependency pointer* from r to t. This produces a *dependency graph* with vertex set Ψ_q. Since there is only transition edge per heavy path, that the indegree of each vertex in this dependency graph is one. Therefore, the dependency graph is a collection of (oriented)*dependency paths*.

Since any path from a vertex to the root ρ of $\mathcal{V}(S)$ traverses $O(\log n)$ light edges, each dependency path has length $O(\log n)$. Let $r \in \Psi_q$ be the sink of a dependency path. Consider the light edge rp_r and notice that it cannot be preserved as p_r is not adjacent to a transition edge. Therefore, we can charge this non-preserved edge with the dependency path with sink r. Since a non-preserved can be charged only once, we have that σ is at least the number of dependency paths. Finally, as each dependency path has length $O(\log n)$, there are at least $\Omega(|\Psi_q|/\log n)$ of them. Therefore $\sigma = \Omega(|\Psi_q|/\log n)$, or equivalently, $|\Psi_q| = O(\sigma \log n)$ which yields our result. □

5.7.5 The compressed tree

Let $\mathcal{F}$ be the forest obtained from $\mathcal{V}_q(S)$ by removing all the shadow edges (this is just for analysis purposes, so far no cut has been performed). Note that each connected component of $\mathcal{F}$ consists only of preserved edges that intersect $R(q, S')$. Thus, each component inside $R(q, S')$ is a comb, with a path as *spine* and each child of a spine vertex pointing to the same side; see Figure 5.5. Thus, we have right and left combs, depending on whether the children of the spine are left or right children.

Our objective in the long term is to cut all the shadow edges and link the remaining components in the appropriate order to complete the flarb. To this end, we would like to perform an Eulerian tour on the subtree $\mathcal{V}_q(S)$ to find the order in which the subtrees of $\mathcal{V}(S) \setminus \mathcal{V}_q(S)$ that hang from the leaves of $\mathcal{V}_q(S)$ appear along this tour. However, this may be too expensive as we want to perform this in time proportional to the number of shadow edges, and the size of $\mathcal{V}_q(S)$ can be much larger. To make this process efficient, we compress $\mathcal{V}_q(S)$ by contracting each comb of $\mathcal{F}$ into a single super-node. By performing an Eulerian tour around this compressed tree, we obtain the

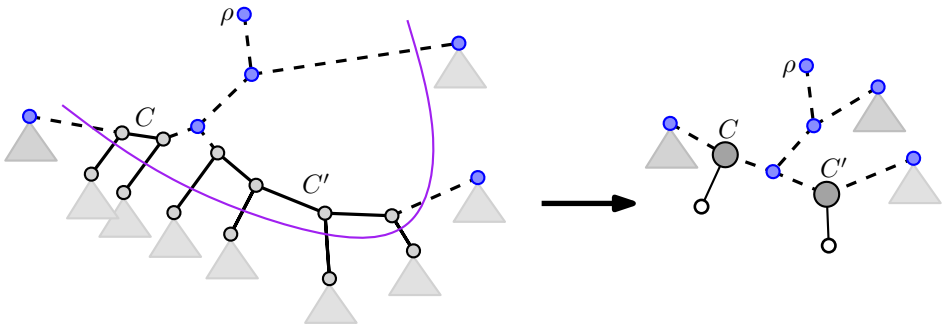

Figure 5.5: Two combs of $\mathcal{F}$ that are compressed into super-nodes with their respective dummy leaves. An Eulerian tour around the compressed tree provides us with the order in which the trees hanging outside of $R(q, S')$ should be attached.

order in which each component needs to be attached. We construct the compressed flarb and then we decompress as follows.

Note that each comb has exactly two shadow edges that connect it with the rest of the tree. Thus, we contract the entire component containing the comb into a single super-node and add a left or right dummy child to it depending on whether this comb was left or right, respectively; see Figure 5.5. After the compression, the shadow edges together with the super-nodes and the dummy vertices form a tree called the *compressed tree* that has $O(\sigma)$ vertices and edges, where σ is the total number of shadow edges.

Lemma 5.7.10. *We can obtain the compressed tree in $O(\sigma \log \sigma)$ time.*

Proof. Notice that each shadow edge is adjacent to two faces, its left and its right face. Recall that each face is bounding the Voronoi region of a site of S and that each shadow edge has two markers pointing to the sties defining its adjacent faces. Using hashing, we can group the shadow edges that are adjacent to the same face in $O(\sigma)$ time. Since preserved faces have no shadow edge on their boundary, we have at most $O(\sigma)$ groups.

Finally, we can sort the shadow edges adjacent to a given face along its boundary. To this end, we use the convex hull position of the sites defining the faces on the other side of each of these shadow edges. Computing this convex hull takes $O(\sigma \log \sigma)$ time. Once the shadow edges are sorted along a face, we can walk and check if consecutive shadow edges are adjacent or not. If their are not, then the path between

them consists only of preserved edges forming a comb; see Figure 5.5. Therefore, we can compress this comb and continue walking along the shadow edges. Since each preserved edge that reappears is adjacent to a face containing at least one shadow edge (namely the face that is not preserved), all the combs will be compressed during this procedure. □

The compressed tree is then a binary tree where each super-node has degree three and each edge is a shadow edge. We now perform an Eulerian tour around this compressed tree and retrieve the order in which the leaves of this tree are visited. Some leaves are dummy leaves and some of them are original leaves of $\mathcal{V}_q(S)$; see Figure 5.5.

5.7.6 Completing the flarb

We now proceed to remove each of the shadow edges which results in a (compressed) forest with $O(\sigma)$ components. Note that each of the original leaves of $\mathcal{V}_q(S)$ was connected with its parent via a shadow edge and hence it lies now as a single component in the resulting forest. For each of these original leaves of $\mathcal{V}_q(S)$, we create a new *anchor* node and link it as the parent of this leaf. Moreover, there could be internal vertices that become isolated. In particular this will be the case of the root ρ. These vertices are deleted and ignored for the rest of the process. To complete the flarb,

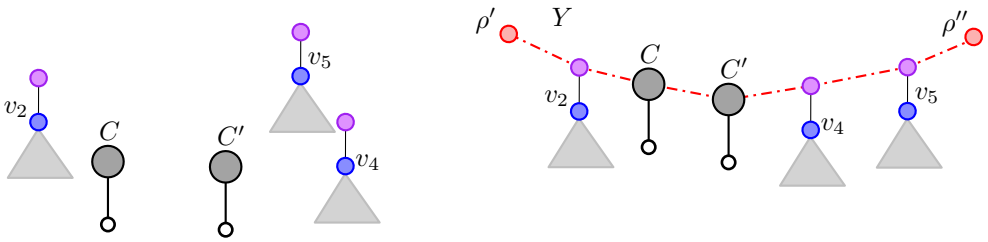

Figure 5.6: Left: An anchor node is created for each isolated leaf of $\mathcal{V}_q(S)$ and attached as its parent. Other isolated nodes are ignored. Right: A super comb is created connecting two new leaves ρ' and ρ'' through a path. This path connects anchor and super-nodes in the order retrieved by the Eulerian tour around the compressed tree.

we create two new nodes ρ' and ρ'' which will be the two new leaves of the Voronoi diagram, one of them replacing ρ. Then, we construct a path with endpoints ρ and ρ' that connects the super-nodes and the anchor nodes according to the traversal order

of their leaves; see Figure 5.6. The resulting tree is a *super comb* Y, where each vertex on the spine is either a super-node or an anchor node, and all the leaves are either dummy leaves or original leaves of $\mathcal{V}_q(S)$. Since we glued $O(\sigma)$ components into a tree, we need $O(\sigma)$ time.

We proceed now to decompress Y. To decompress a super-node of Y that corresponds to a comb, we consider the two neighbors of the super-node in Y and attach each of them to the ends of the spine of the comb. For an anchor node, we simply note that there is a component of $\mathcal{V}(S)$ hanging from its leaf; see Figure 5.7. In this way, we obtain all the edges that need linking. After the decompression, we end with the tree $\mathcal{V}(S')$ resulting from the flarb. Thus, the flarb operation of inserting q can be implemented with $O(\sigma)$ link and cuts.

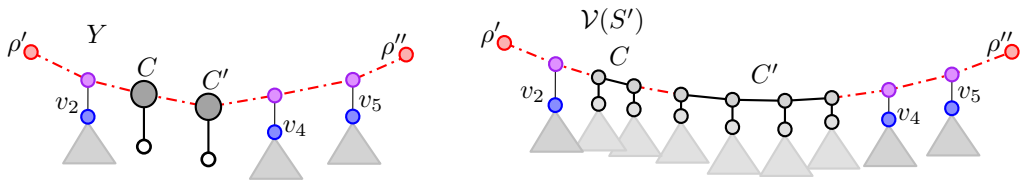

Figure 5.7: The tree $\mathcal{V}(S')$ achieved after the decompression.

Recall that any optimal algorithm needs to perform a cut for each edge that is not preserved. Since each non-preserved edge is shadow by Lemma 5.7.7, the optimal algorithm needs to perform at least $\Omega(\sigma)$ operations. Therefore, our algorithm is optimal and computes the flarb using $\Theta(\sigma)$ link and cuts. Moreover, by Lemmas 5.7.4 and 5.7.5 we can compute the flarb in $O(|\Psi_q| \log^6 n + \sigma \log n)$ amortized time using $\Theta(\sigma)$ link and cuts. Since $|\Psi_q| = O(\sigma \log n)$ by Lemma 5.7.9, we obtain the following.

Theorem 5.7.11. *The flarb operation of inserting q can be implemented with $O(K)$ link and cuts, where K is the cost of the flarb. Moreover, it can be implemented in $O(K \log^7 n)$ amortized time.*

Chapter 6

Constrained Minimum Enclosing Circle

Let P be a set of n points in the plane. The minimum enclosing circle problem, originally posed by Sylvester in 1857 [89], asks to identify the center and radius of the minimum enclosing circle of P. For ease of notation we say that every circle enclosing P is a P-circle. Megiddo [70] settled the complexity of this problem presenting a $\Theta(n)$-time algorithm using prune and search.

Finding the minimum P-circle is also known as the 1-center facility location problem: Given the position of a set of clients (represented by P), compute the optimal location for a facility such that the maximum distance between a client and its closest facility is minimized. The aforementioned algorithm provides a solution to this problem. However, in most situations the location of the facility is constrained by external factors such as the geography and features of the terrain. Therefore, the study of constrained versions of the 1-center problem is of importance and has received great attention from the research community [21, 22, 23, 56].

Bose *et al.* [23] addressed the query version of the problem and proposed an $O(n \log n)$-time preprocessing on P, that allows them to find the minimum P-circle with center on a given query line in $O(\log n)$ time. Using this result, they showed how to compute the minimum P-circle, whose center is constrained to lie on a set of m segments, in $O((n + m) \log n)$ time. However, when $m = O(1)$, the problem can be solved in $O(n)$ time by using Megiddo's algorithm [70] a constant number of times. Moreover, when $n = O(1)$, the problem can be solved in $O(m)$ time by finding the farthest point of P from every given segment. Therefore, one would expect an algorithm that behaves like the algorithm presented in [23] when $m = O(n)$ but that converges to a linear running time as the difference between n and m increases (either to $O(n)$ or to $O(m)$). In this chapter we show that such an algorithm exists and prove its optimality. When constraining the center to lie on a simple m-gon however, the

order of the vertices along its boundary allows us to further speed up our algorithm, provided that m is larger than n.

Let M be a set of m points, let S be a set of m segments and let Q be a simple polygon on m vertices. We say that a P-circle C has its center on M, on S or on Q if the center of C is either a point of M, lies on a segment of S or belongs to Q, respectively. The (P, M)-*optimization problem* asks to find the minimum P-circle with center on M. Given a radius r, the $(P, M)_r$-*decision problem* asks if there is a P-circle of radius r with center on M. Analogous problems exist for S and Q. In Section 6.3, we show a $\Theta((n + m) \log \omega)$-time algorithm for the $(P, S)_r$-decision problem where $\omega = \min\{n, m\}$. In Section 6.4, we transform this decision problem into the (P, S)-optimization problem and solve it in the same running time. In Section 6.6, we show a matching lower bound in the algebraic decision-tree model provided that $n \leq m$ for the case when the center is restricted to be in a set of either points, lines, segments or even in a simple polygon. When $m > n$ however, we only prove a matching lower bound when the center is restricted to be on a set of points, segments or lines, yet the lower bound breaks down when the restriction is on a simple polygon. Indeed, given a simple m-gon Q, we show an $\Theta(m + n \log n)$-time algorithm for the (P, Q)-optimization problem. To put this in perspective, note that whenever $m = \Omega(n \log n)$, the algorithm runs in $\Theta(m)$ time. Since the bottleneck of this algorithm is the computation of the farthest-point Voronoi diagram, if we assume that P is the set of vertices of a given convex n-gon we can reduce the running time to $\Theta(m + n)$ [1, 63]. Finally, we show a matching lower bound for these algorithms, thereby solving the problem for all ranges of n and m and all possible restrictions on points, lines, segments and simple polygons.

As a side note, while proving these lower bounds, we stumbled upon the following problem: Given two subsets A and B of $\mathbb{R}$ of sizes m and n, respectively, is $A \subseteq B$? While a lower bound of $\Omega(n \log n)$ is known in the case where $n = m$ [17], no lower bounds were known when m and n differ. Using a method of Yao [91] and the topology of affine subspace families, we were able to prove an $\Omega(n \log m)$ lower bound, even when A is restricted to be a sorted set of m real numbers.

Although the main techniques used in this chapter have been around for a while [32,

66], they are put together in a different way in this work, showing the potential of several tools that were not specifically designed for this purpose. Furthermore, these results provide significant improvements over previous algorithms when n and m differ widely as it is the case in most applications.

6.1 Outline

Let P be a set of n points in the plane. Let S be a set of m segments and let r be a real number larger than the radius of the minimum enclosing circle of P. In Section 6.3 we present an $O((n+m)\log\omega)$-time algorithm to solve the $(P,S)_r$-decision problem, where $\omega = \min\{n,m\}$.

To this end, we use divide a conquer to decide if there is a segment of S that has a point belonging to all the disks of radius r centered at points of P. The partition is obtained by careful applications of ε-nets. While these techniques have been used before, in most cases the subproblems are not disjoint which leads to a "blow up" in the running time. In our algorithm however, we are able to make each of the subproblems disjoint using several geometric observations.

Intuitively, we obtain our partition by constructing a constant number of cones having the center of the minimum enclosing circle of P as an apex. Each of these cones is in turn refined by intersecting it with some circles of radius r centered on specific points of P obtained by the ε-nets. This creates a constant number "slices". Using these slices, we are able to create subproblems: For a given slice, we consider only the segments of S intersecting this slice. Moreover, we consider only the points of P such that the circle of radius r centered on them intersects this slice. The use of ε-nets allows us to guarantee that only a constant fraction of the points and a fraction of the segments belong to each subproblem. Moreover, a careful selection of the slices guarantees that a segment of S (or a point of P) belongs to at most one of the subproblems. These creates disjoint subproblems such that the main decision problem has a positive solution only if one of the subproblems has a positive solution. This allows us to define a recursive procedure having $O(\log\omega)$ depth and such that the size of each level of the recursion is $O(n+m)$. This leads to the $O((n+m)\log\omega)$-time algorithm to solve the $(P,S)_r$-decision problem.

In Section 6.4, we use the technique presented in Section 3.2 to transform a decision algorithm into an optimization algorithm. This technique requires an efficient algorithm to partition the problem into smaller subproblems, where the global solution is the minimum among the subproblems solutions. Using techniques to those in Section 6.3, we are able to partition the problem in linear time and use the technique presented in Section 3.2 to solve the (P, S)-optimization problem in $O((n+m)\log\omega)$ time.

6.2 Preliminaries

In this section, we provide some of the definitions that will be used throughout this chapter.

6.2.1 P-circles

Given a subset X of the plane, the interior and convex hull of X are denoted by $\textsc{int}(X)$ and $\textsc{ch}(X)$, respectively. A point x is *enclosed* by a circle C if $x \in \textsc{ch}(C)$; otherwise we say that x is *excluded* by C. An *X-circle* is a circle that encloses every point of X.

Let P be a set of n points in the plane. Given a point $x \in \mathbb{R}^2$, let $\bigcirc_r(x)$ be the circle with radius r and center on x. Given $W \subseteq P$, let $\Lambda_r(W) = \cap_{p\in W}\textsc{ch}(\bigcirc_r(p))$, i.e., the intersection of every disk of radius r with center at a point of W. Notice that $\Lambda_r(W)$ is a convex set whose boundary is composed of circular arcs each with the same curvature.

A point $p \in W$ *contributes* to $\Lambda_r(W)$ if there is an arc of the circle $\bigcirc_r(p)$ on the boundary of $\Lambda_r(W)$. We refer to this arc as the *contribution of p* to $\Lambda_r(W)$ and we denote it by $\alpha_r(p, W)$. As the curvature of all circles is the same, a point contributes with at most one arc to the boundary of $\Lambda_r(W)$ as shown in the following lemma.

Lemma 6.2.1. *Let $W \subseteq P$. If a point $p \in W$ contributes to $\Lambda_r(W)$, then there is exactly one arc $\alpha_r(p, W)$ on the boundary of $\Lambda_r(W)$ contained in $\bigcirc_r(p)$.*

Proof. Let x, y be two points on $\bigcirc_r(p) \cap \partial\Lambda_r(W)$. The shortest arc joining x with y along $\bigcirc_r(p)$ is contained in $\Lambda_r(W)$; otherwise, there would be a point $q \in W$ such that $\bigcirc_r(q)$ contains x and y but excludes another point lying on α. However, this

can only happen if $\bigcirc_r(q)$ has a radius greater than $\bigcirc_r(p)$. Therefore $\bigcirc_r(p) \cap \Lambda_r(W)$ is connected and consists of at most one arc, namely $\alpha_r(p, W)$. □

Given two subsets X and Y of the plane, let $B_X(Y)$ be the minimum X-circle with center on Y and let $b_X(Y)$ be its center.

If $X = P$, we let $\rho(Y)$ denote the radius of $B_P(Y)$, i.e., $\rho(Y)$ is the radius of the minimum P-circle with center on Y. When referring to a single point x, let $B_P(x) = B_P(\{x\})$ and let $\rho(x)$ be the radius of $B_P(x)$. Let C_P be the minimum P-circle, c_P be its center and r_P be its radius.

Observation 6.2.2. *Given a point $x \in \mathbb{R}^2$ and a real number $r \geq r_P$, $\rho(x) \leq r$ if and only if $x \in \Lambda_r(P)$. Moreover, $\rho(x) = r$ if and only if x lies on the boundary of $\Lambda_r(P)$.*

Proposition 6.2.3. *Let x and y be two points in $\mathbb{R}^2$. If z is a point contained in the open segment (x, y), then $B_P(z) \subseteq B_P(x) \cup B_P(y)$. Moreover, $\rho(z) < \max\{\rho(x), \rho(y)\}$.*

Proof. Let a and b be the two points of intersection between $B_P(x)$ and $B_P(y)$ and let C_z be the circle with center at z passing through a and b. It is clear that $B_P(x) \cap B_P(y) \subseteq C_z$ and since $P \subset B_P(x) \cap B_P(y)$, we infer that C_z is a P-circle, therefore $B_P(z) \subseteq C_z$. Furthermore, since $C_z \subseteq B_P(x) \cup B_P(y)$ by construction, we conclude that $B_P(z) \subseteq B_P(x) \cup B_P(y)$ and therefore, $\rho(z) < \max\{\rho(x), \rho(y)\}$. □

Proposition 6.2.4. *If S is a set of segments in the plane, then $b_P(S)$ is visible from c_P. That is, the open segment joining $b_P(S)$ with c_P intersects no segment of S.*

Proof. Let w be the open segment joining $b_P(S)$ with c_P. Note that $B_P(S)$ and C_P intersect since both are P-circles. From Proposition 6.2.3, we know that for every z in w, $\rho(z) < \max\{\rho(S), \rho(c_P)\} = \rho(S)$. Therefore, no segment of S intersects w as otherwise this intersection would be the center of a P-circle with smaller radius than $B_P(S)$—a contradiction with the minimality of $B_P(S)$. In other words, $b_P(S)$ must be visible from c_P. □

6.3 Solving the decision problem on a set of segments

Let S be a set of m segments and let $r > r_P$. In this section we present an $O((n + m)\log\omega)$ time algorithm to solve the $(P, S)_r$-decision problem for the given radius r, where $\omega = \min\{n, m\}$.

Notice that by Observation 6.2.2, if we could compute $\Lambda_r(P)$, we could decide if there is a P-circle of radius r with center on S by checking if there is a segment of S that intersects $\Lambda_r(P)$. However, we cannot compute $\Lambda_r(P)$ explicitly as this requires $\Omega(n \log n)$ time. Thus, we approximate it using ε-nets and use it to split both S and P into a constant number of subsets each representing a subproblem of smaller size. Using Divide-and-conquer we determine if there is an intersection between S and $\Lambda_r(P)$ by solving the decision problem recursively for each of the subproblems. The algorithm runs in $O(\min\{\log n, \log m\})$ phases and on each of them we spend $O(n + m)$ time. Further details are required to bound the size of the subproblems on each level of the recursion.

6.3.1 The algorithm

Initially compute the minimum P-circle C_P, its center c_P and its radius r_P in $O(n)$ time [70]. In $O(m)$ time we can verify if c_P lies on a segment of S. If it does, then C_P is the minimum P-circle with center on S. Otherwise, as we assume from now on, the radius of $B_P(S)$ is greater than r_P.

Consider a family of convex sets $\mathcal{G}$ defined as follows. A set $G \in \mathcal{G}$ is the intersection of $A_1 \cap \ldots \cap A_6$, where each A_i is either the interior of a circle or an open halfplane supported by a straight line (A_i may be equal to A_j for some $i \neq j$).

Fix a constant $0 < \varepsilon \leq 1$ and consider the set system $(S, \mathcal{G}_S)$ induced by $\mathcal{G}$ on S. As the VC-dimension of this range space is finite by Theorem 3.1.2, we can compute an ε-net N_S of $(S, \mathcal{G}_S)$ of size $O(1)$ in $O(m)$ time using Theorem 3.1.3. That is, each set $G \in \mathcal{G}$ that intersects more than εm segments of S intersects at least one segment of N_S.

For each segment s of N_S, compute $B_P(s)$ in $O(n)$ time [70] and mark at most two points of P that uniquely define this circle by lying on its boundary. Let r_{min} be the radius of the minimum circle among the P-circles computed in the previous step.

If $r_{min} \leq r$, then there is a positive answer to the $(P,S)_r$-decision problem and the decision algorithm finishes. Otherwise, let $M_P \subset P$ be the set of marked points and note that $|M_P| \leq 2|N_S| = O(1)$.

By the minimality of r_{min}, the interior of $\Lambda_{r_{min}}(M_P)$ intersects no segment of N_S. Since we assumed that $r_{min} > r$, we know that $\Lambda_r(M_P) \subset \Lambda_{r_{min}}(M_P)$ and hence $\Lambda_r(M_P)$ intersects no segment of N_S.

We refine this intersection using another ε-net. Let $\mathcal{C} = \{\bigcirc_r(p) : p \in P\}$ be the set of circles of radius r centered at the points of P. Consider the set system $(\mathcal{C}, \mathcal{G}_\mathcal{C})$ induced by $\mathcal{G}$ on $\mathcal{C}$. Compute an ε-net N_P of this set system in $O(n)$ time using Theorem 3.1.3. That is, if a convex set $G \in \mathcal{G}$ intersects more than εn circles of $\mathcal{C}$, then G intersects at least one circle of N_P.

Let $W_P = \{p \in P : \bigcirc_r(p) \in N_P\}$ i.e., W_P is the subset of P defining N_P where $|W_P| = O(1)$. Notice that for every $p \in W_P$, $\Lambda_r(W_P)$ is enclosed by $\bigcirc_r(p)$, i.e. the circle $\bigcirc_r(p)$ does not intersect the interior of $\Lambda_r(W_P)$. Let $P^+ = M_P \cup W_P$. As $\Lambda_r(P^+)$ is contained in both $\Lambda_r(M_P)$ and $\Lambda_r(W_P)$, we observe the following.

Lemma 6.3.1. *No segment of N_S and no circle of N_P intersects the interior of $\Lambda_r(P^+)$.*

Because $r > r_P$, c_P is enclosed by $\bigcirc_r(p)$ for every $p \in P$, i.e., c_P lies in the interior of $\Lambda_r(P^+)$. Our idea is to partition $\Lambda_r(P^+)$ by connecting c_P with the vertices of $\Lambda_r(P^+)$. This operation produces "slices" of $\Lambda_r(P^+)$ which are convex sets that belong to $\mathcal{G}$ and do not intersect either N_S or N_P. Therefore, each "slice" is intersected by few segments of S and few circles from $\mathcal{C}$. This is the idea behind our Divide-and-conquer algorithm. However, we would like to go even further and guarantee that each segment of S (and each circle in $\mathcal{C}$) appears only in one slice.

To achieve this, we construct a set of points $\Pi \supset P^+$ of constant size such that Π retains the properties of P^+, but whose "slices" allow us to partition both S and $\mathcal{C}$. We detail the construction of Π below.

6.3.2 Constructing slices

Because $\Lambda_r(P^+)$ is a convex set whose boundary is composed of circular arcs, we can think of it as described by the sequence of vertices in clockwise order along its

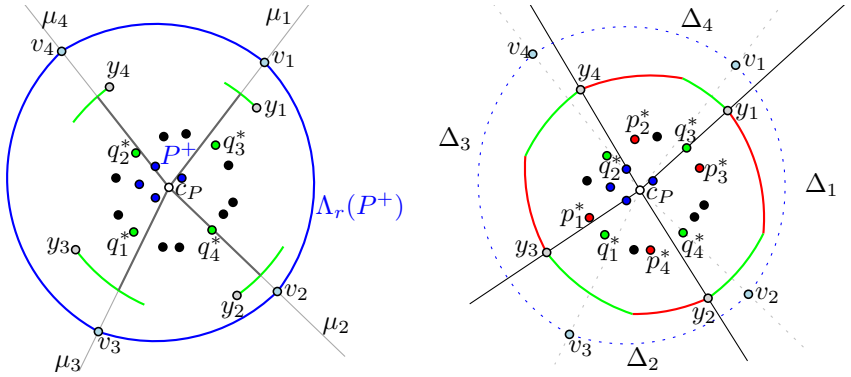

Figure 6.1: Left: The set P^+ and the convex set $\Lambda_r(P^+)$ with vertices $v_1, \ldots, v_4$ are depicted in blue. In green the points $q_1^*, \ldots, q_4^*$ such that the $\bigcirc_r(q_i^*)$ is the first circle of $\mathcal{C}$ intersected by the ray μ_i going from c_P towards v_i. Right: The set $P^* = \{q_1^*, p_1^* \ldots, q_4^*, p_4^*\}$ such that the set $Y = \{y_1, \ldots, y_4\}$ is contained in the boundary of $\Lambda_r(P^*)$. For every $1 \leq i \leq 4$, no circle in $\mathcal{C}$ intersects the open segment (c_P, y_i). The rays from c_P towards each y_i split the plane into cones $\Delta_1, \ldots, \Delta_4$.

boundary. Assume that $\Lambda_r(P^+) = (v_1, \ldots, v_k)$ where $k = O(1)$. Recall that c_P lies in the interior of $\Lambda_r(P^+)$. For each $1 \leq i \leq k$, let μ_i be the ray with apex c_P that passes through v_i. Let q_i^* be a point of P such that $\bigcirc_r(q_i^*)$ is the first circle of $\mathcal{C}$ intersected by μ_i when going from c_P towards v_i.

As $O(1)$ rays are considered, we can compute the set $\{q_1^*, \ldots, q_k^*\}$ in $O(n)$ time. By construction, each q_i^* contributes to $\Lambda_r(P)$ as no circle in $\mathcal{C}$ intersects the portion of μ_i between c_P and its intersection with $\bigcirc_r(q_i^*)$.

For every $1 \leq i \leq k$, we can compute in $O(n)$ time the arc $\alpha_r(q_i^*, P)$ by intersecting $\bigcirc_r(q_i^*)$ with every other circle in $\mathcal{C}$. Let y_i be the most clockwise endpoint of $\alpha_r(q_i^*, P)$ and let $Y = \{y_1, \ldots, y_k\}$; see Fig. 6.1.

Notice that each y_i is the intersection of $\bigcirc_r(q_i^*)$ and $\bigcirc_r(p_i^*)$ for some point $p_i^* \in P$. Let $P^* = \{q_1^*, p_1^*, \ldots, q_k^*, p_k^*\}$ and notice that for every $1 \leq i \leq k$, p_i^* and q_i^* contribute to $\Lambda_r(P)$ and y_i is the point where their contributions intersect; we observe the following.

Observation 6.3.2. *Let X be a set of points such that $P^* \subseteq X \subseteq P$. Every point of Y lies on the boundary of $\Lambda_r(X)$.*

Consider the k rays with apex at c_P that pass through $y_1, \ldots, y_k$. These rays split the plane into k cones $\mathcal{D} = \{\Delta_1, \ldots, \Delta_k\}$ such that the boundary of Δ_i contains y_i

and y_{i+1}; see Fig. 6.1. By Observation 6.3.2 and by convexity, $\text{CH}(Y)$ is contained in $\Lambda_r(P) \subset \Lambda_r(P^*)$. If there is a segment of S intersecting $\text{CH}(Y)$, then the $(P,S)_r$-decision problem has a positive answer. Since we can test every segment of S for intersection with $\text{CH}(Y)$ in $O(m)$ time, we assume from now on that no segment of S intersects $\text{CH}(Y)$. Under this assumption, we obtain the following.

Lemma 6.3.3. *Let X be a set of points such that $P^* \subseteq X \subseteq P$. For every $s \in S$, if s intersects $\Lambda_r(X)$, then $s \cap \Lambda_r(X)$ is contained in exactly one cone of $\mathcal{D}$.*

Proof. Since every point of Y lies on the boundary of $\Lambda_r(X)$ by Observation 6.3.2, the difference $\overline{Y} = \Lambda_r(X) \setminus \text{CH}(Y)$ consists of k disjoint components. Let x and y be two points on different components of $\overline{Y}$. By convexity of $\Lambda_r(X)$, the segment $[x,y]$ must intersect $\text{CH}(Y)$. Therefore, there is no segment that intersects two or more components of $\overline{Y}$ without intersecting $\text{CH}(Y)$. Finally note that each disjoint component of $\overline{Y}$ is completely contained in some cone of $\mathcal{D}$. □

The following lemma shows that not only the segments can be classified using the cones in $\mathcal{D}$, but also the points of P that contribute to $\Lambda_r(P)$.

Lemma 6.3.4. *For any point $p \in P$, if p contributes to $\Lambda_r(P)$, then its contribution $\alpha_r(p,P)$ is contained in exactly one cone of $\mathcal{D}$.*

Proof. If $p \in P \setminus P^*$ contributes to $\Lambda_r(P)$ then $\bigcirc_r(p)$ must intersect $\Lambda_r(P^*)$ and this happens only if $\bigcirc_r(p)$ excludes some vertex v of $\Lambda_r(P^*)$ such that $v \notin Y$. Notice that at most one vertex of $\Lambda_r(P^*)$ lies inside each cone Δ_i (p_i^* may be equal to q_i^* and hence, there may be no vertex of $\Lambda_r(P^*)$ inside Δ_i). Let $y_i, y_{i+1} \in Y$ be the two neighbors of v along the boundary of $\Lambda_r(P^*)$. Since $\bigcirc_r(p)$ excludes v, $\bigcirc_r(p)$ intersects $\Lambda_r(P^*)$ but must enclose both y_i and y_{i+1}. Therefore, there is one arc β of $\bigcirc_r(p)$ that intersects $\Lambda_r(P^*)$ and β is completely contained in Δ_i. Furthermore, by Lemma 6.2.1, β is the only intersection between $\bigcirc_r(p)$ and $\Lambda_r(P^*)$; that is, $\alpha_r(p,P) \subseteq \beta$ is completely contained in Δ_i. By construction, the contribution of each point in P^* to $\Lambda_r(P)$ is contained in exactly one cone of $\mathcal{D}$. □

Let $\Pi = P^+ \cup P^*$ and notice that $|\Pi| = O(1)$. Intuitively, Π inherits the properties from both P^+ and P^*. For every $1 \leq i \leq k$, let $R_i = \Delta_i \cap \Lambda_r(\Pi)$ be a Π-*slice* of $\Lambda_r(\Pi)$. Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ be the set of Π-slices.

Since at most four points of Π contribute to $\Lambda_r(\Pi)$ inside each cone of $\mathcal{D}$ (two from P^* and at most two from P^+), we infer that each Π-slice of $\mathcal{R}$ belongs to $\mathcal{G}$.

By Lemma 6.3.1, the interior of each region R_i intersects no segment of N_S and no circle of N_P. Thus, as N_S and N_P are both ε-nets, we obtain the following result.

Corollary 6.3.5. *For each $1 \leq i \leq k$, at most εm segments of S intersect the region R_i and at most εn circles of $\mathcal{C}$ intersect R_i.*

6.3.3 Divide-and-conquer

The idea is to use Divide-and-conquer using Corollary 6.3.5. That is, we split both P and S into k subsets according to their intersection with the elements of $\mathcal{R}$, each pair of which represents a subproblem. Finally, we will prove that the $(P, S)_r$-decision problem has a positive answer if and only if some subproblem has a positive answer. We first described how to partition the set S.

Lemma 6.3.6. *We can compute sets $S_1, \ldots, S_k \subset S$ in $O(m)$ time such that $\sum_{i=1}^{k} |S_i| \leq m$ and $|S_i| < \varepsilon m$. Moreover, S_i contains all segments of S that intersect $\Lambda_r(\Pi)$ inside R_i.*

Proof. Note that segments that do not intersect $\Lambda_r(\Pi)$ can be discarded as they cannot contain a point being the center of a P-circle of radius r.

Let S_i be the set of segments of S that intersect R_i. The construction of $S_1, \ldots, S_k$ can be performed in $O(m)$ time since the size of R_i is constant. By Lemma 6.3.3, a segment of S belongs to at most one element of the partition and hence, $\sum_{i=1}^{k} m_i \leq m$. By Corollary 6.3.5, for any $1 \leq i \leq k$ at most εm segments of S intersect R_i. Consequently, $|S_i| < \varepsilon m$. □

In a similar way, we describe how to partition the set of points P.

Lemma 6.3.7. *We can compute sets $P_1, \ldots P_k \subset P$ in $O(n)$ time such that $p_i^*, q_{i+1}^* \in P_i$, $|P_i| < \varepsilon n$, $\sum_{i=1}^{k} |P_i| \leq n$ and $\Lambda_r(P) = \Lambda_r(P_1 \cup \ldots \cup P_k)$. Moreover, if a point $p \in P_i$ contributes to $\Lambda_r(P)$, then $\alpha_r(p, P)$ intersects R_i.*

Proof. Let $P_i = \{p \in P : \bigcirc_r(p)$ intersects $R_i \setminus Y\}$ (we remove Y since y_{i+1} belongs to both R_i and R_{i+1} leading some points to belong to more than one P_i). This partition

of P can be computed in $O(n)$ time as the size of R_i is constant. For each point $p \in P$ that contributes to $\Lambda_r(P)$, $\bigcirc_r(p)$ has to intersect $\Lambda_r(\Pi)$ and hence, $\alpha_r(p, P)$ intersects at least one region of $\mathcal{R}$. By Lemma 6.3.4, a point of P contributes to $\Lambda_r(\Pi) \subset \Lambda_r(P^*)$ inside only one cone of $\mathcal{D}$. Thus, a point of P belongs to at most one of the computed subsets. Moreover, by Corollary 6.3.5, at most εn circles of $\mathcal{C}$ intersect R_i, i.e., $|P_i| < \varepsilon n$. By construction, p_i^* and q_{i+1}^* have their contribution to $\Lambda_r(P)$ contained in R_i and hence, both belong to P_i. □

The following theorem states that the partitions of P and S induce subproblems in such a way that the solution to the decision problem can be retrieved from the subproblems' solutions.

Theorem 6.3.8. *The $(P, S)_r$-decision problem has positive answer if and only if there is a P_i-circle of radius r with center on S_i, for some $1 \leq i \leq k$.*

Proof. $\rightarrow$) Let C be a P-circle of radius r with center c lying on a segment $s \in S$. Since the cones of $\mathcal{D}$ partition the plane, c belongs to some cone Δ_i for some $1 \leq i \leq k$. Thus, s intersects the cone Δ_i. By Observation 6.2.2, s intersects $\Lambda_r(P) \subset \Lambda_r(\Pi)$. Consequently, by Lemma 6.3.6, s belongs to S_i. Assume that c lies on the arc $\alpha_r(p, P)$ being the contribution of some point $p \in P$. Since c belongs to Δ_i and lies on the boundary of $\Lambda_r(P) \subset \Lambda_r(\Pi)$, $\bigcirc_r(p)$ intersects $\Lambda_r(\Pi) \cap \Delta_i = R_i$. Thus, by Lemma 6.3.7, p belongs to P_i. That is, there is a P_i-circle of radius r with center on S_i.
$\leftarrow$) Let C be a P_i-circle of radius r with center c lying on a segment s of S_i. Suppose that C is not a P-circle, i.e., assume that there is a point $p \in P$ such that p is not enclosed by C. Thus, $\bigcirc_r(p)$ does not enclose c. We claim that $c \in R_i$; if this claim is true, then as c_P is enclosed by $\bigcirc_r(p)$, $\bigcirc_r(p)$ must intersect R_i in order to exclude c. However, $\bigcirc_r(p)$ intersects R_i if and only if p belongs to P_i—a contradiction since every point of P_i is enclosed by C.

We proceed to prove our claim ($c \in R_i$). By Lemma 6.3.7, we know that P_i contains the points p_i^* and q_{i+1}^* of P^*. Let $L = \text{CH}(\bigcirc_r(p_i^*)) \cap \text{CH}(\bigcirc_r(q_{i+1}^*))$ and notice that the points y_i and y_{i+1} of Y lie on the boundary of L. Therefore, the segment $[y_i, y_{i+1}]$ splits L into two convex sets L^+ and L^-. As both y_i and y_{i+1} belong to Δ_i, either L^+ or L^- is completely contained in Δ_i. As $P^* \subset \Pi$ and $p_i^*, q_{i+1}^* \in P_i$,

both $\Lambda_r(P_i)$ and $\Lambda_r(\Pi)$ are contained in L. Since $s \in S_i$, by definition s intersects $\Lambda_r(\Pi)$ inside Δ_i and hence, s intersects L inside Δ_i; assume *wlog* that s intersects L^+. Therefore, no point of s can lie inside $\Lambda_r(P_i)\backslash\Delta_i$ as it would be in L^- and by convexity, s would intersect the segment $[y_i, y_{i+1}]$; however, we assumed that no segment of S intersects $\textsc{ch}(Y)$. As c lies in Δ_i, $c \in R_i$ since $\Lambda_r(P_i) \cap \Delta_i \subset \Lambda_r(\Pi) \cap \Delta_i = R_i$. □

By Lemmas 6.3.6 and 6.3.7, in $O(n + m)$ time we can either give a positive answer to the decision algorithm, or compute sets $P_1, \ldots, P_k$ and $S_1, \ldots, S_k$ in order to define k decision subproblems each stated as follows: Decide if there is a P_i-circle of radius r with center on S_i. Because Theorem 6.3.8 allows us to solve each subproblem independently, we proceed until we find a positive answer on some branch of the recursion, or until either P_i or S_i reaches $O(1)$ size and can be solved in linear time. Since $|S_i| < \varepsilon m$ and $|P_i| < \varepsilon n$ by Lemmas 6.3.6 and 6.3.7, the number of recursion steps needed is $O(\min\{\log n, \log m\})$. Furthermore, by Lemmas 6.3.6 and 6.3.7, the size of all subproblems at the i-th level of the recursion is bounded above by $n + m$.

Lemma 6.3.9. *Given sets P of n points and S of m segments and $r > 0$, the $(P, S)_r$-decision problem can be solved in $O((n + m) \log \omega)$ time, where $\omega = \min\{n, m\}$.*

6.4 Converting decision to optimization

In the previous section we showed an algorithm for the $(P, S)_r$-decision problem. However, our main objective is to solve its optimization version. To do that, we use Lemma 3.2.1. This technique requires an efficient algorithm to partition the problem into smaller subproblems, where the global solution is the minimum among the subproblems solutions. By presenting an $O(n + m)$-time partition algorithm, we obtain a randomized algorithm for the (P, S)-optimization problem having an expected running time of $O((n+m) \log \omega)$, where $\omega = \min\{n, m\}$. As the partition of the plane into cones used in the previous section has no correlation with the structure of $\Lambda_r(P)$ as r changes, the partition of P used in this section requires a different approach. However, the partition of S is very similar. We present the statement of the main result and will spend the rest of the section proving it.

Lemma 6.4.1. *We can compute sets $P'_1, \ldots, P'_h \subset P$ and $S'_1, \ldots, S'_h \subset S$ in $O(n+m)$ time such that $|P'_i| < \varepsilon n$, $|S'_i| < \varepsilon m$ and $B_P(S)$ is the circle of minimum radius in $\{B_{P'_1}(S'_1), \ldots, B_{P'_h}(S'_h)\}$.*

We introduce some notation that will help us prove our result. The farthest-point Voronoi diagram of a point set $X \subset \mathbb{R}^2$ can be seen as a tree with n unbounded edges and is denoted by $\mathcal{V}(X)$ (see [35] for further information on this diagram). For any point x of X, let $R(x)$ be the farthest-point Voronoi cell associated with x in $\mathcal{V}(X)$, i.e., the region of the plane containing the points that are farther from x than from any other point of X.

Observation 6.4.2. *Let $r > r_P$. For every point $p \in P$ that contributes to $\Lambda_r(P)$, a point x lies on $\alpha_r(p, P)$ if and only if x lies in the Voronoi cell of p in $\mathcal{V}(P)$.*

Given any subset of $\mathbb{R}^2$, it can be *embedded* into $\mathbb{R}^3$ by identifying $\mathbb{R}^2$ with the plane $Z_0 = \{(x, y, z) \in \mathbb{R}^3 : z = 0\}$. To prove Lemma 6.4.1, we start by partitioning P using a different ε-net defined in $\mathbb{R}^3$.

Given a point p in Z_0, let γ_p be the boundary of the 3-dimensional cone, lying above p, with apex at p and 45° slope with respect to the plane Z_0. That is $\gamma_p = \{(x, y, z) \in \mathbb{R}^3 : (x - p_x)^2 + (y - p_y)^2 = z^2\}$. Let γ_p^+ be the open set containing all the points lying above γ_p.

Embed P into $\mathbb{R}^3$ and let $\Gamma = \{\gamma_p : p \in P\}$. Recall that the furthest-point Voronoi diagram of P is the upper envelope of Γ when projected to the plane Z_0. That is, a point x in the plane Z_0 belongs to $R(p)$ if and only if γ_p is the last cone intersected by a ray shooting upwards, orthogonally to the plane Z_0, from x.

To do the partition, we consider the following set system. Consider the family of convex sets $\mathcal{H}$ defined as follows. A convex region H belongs to $\mathcal{H}$ if for some $x \in Z_0$, $H = h_1 \cap h_2 \cap h_3 \cap \gamma_x^+$, where each h_i is a halfspace supported by a halfplane orthogonal to Z_0.

Consider the set system $(\Gamma, \mathcal{H}_\Gamma)$ induced by $\mathcal{H}$ on Γ. Since each element of G and of $\mathcal{H}$ is a semialgebraic set of constant description, this set system has finite VC-dimension by Theorem 3.1.2.

Fix a sufficiently small constant ε and compute an ε-net N_P of $(\Gamma, \mathcal{H}_\Gamma)$ in $O(n)$ time such that $|N_P| = O(1)$ (see Theorem 3.1.3). Let $P^+ = \{p \in P : \gamma_p \in N_P\}$, i.e.,

P^+ is the constant size subset of P defining N_P. Compute the upper envelope of N_P, denoted by $\mathcal{U}$, in constant time and let $\mathcal{U}^+$ be the set of points in $\mathbb{R}^3$ lying strictly above $\mathcal{U}$.

Observation 6.4.3. *The interior of $\mathcal{U}^+$ intersects no cone of N_P.*

Given a point x in $\mathbb{R}^3$, let ℓ_x be the vertical line passing through x in 3D-space. Given a set $Y \subset \mathbb{R}^2$, the *vertical lifting* of Y is the set $VL(Y) = \cup_{x \in Y} \ell_x$ obtained after embedding Y into $\mathbb{R}^3$. For any set that partitions the plane into k disjoint maximal components, its vertical lifting partitions $\mathbb{R}^3$ into the same number of maximal connected components.

Compute the farthest-point Voronoi diagram of P^+, denoted by $\mathcal{V}(P^+)$, in $O(1)$ time. Notice that $\mathcal{V}(P^+)$ is the projection on Z_0 of $\mathcal{U}$. Complete the planar graph $\mathcal{V}(P^+)$ to a triangulation $\mathcal{T}$ in $O(1)$ time by adding edges greedily (some triangles have the point at ∞ as a vertex); see Fig. 6.4. Embed $\mathcal{T}$ in $\mathbb{R}^3$ and compute its vertical lifting $VL(\mathcal{T})$ in constant time. Since $\mathcal{T}$ partitions the plane, $VL(\mathcal{T})$ partitions $\mathbb{R}^3$ into $O(1)$ solid prisms, each defined by the intersection of two or three halfspaces. Finally, intersect each of these prisms with $\mathcal{U}^+$ to obtain a family of convex regions $\mathcal{F} = \{F_1, \ldots, F_t\}$ for some $t \in O(1)$.

Since each region F_i projects on Z_0 inside exactly one Voronoi cell $R(p_i)$ of $\mathcal{V}(P^+)$ for some $p_i \in P^+$, we can uniquely associate the point p_i with F_i (p_i may be equal to p_j for $i \neq j$) and hence $F_i \in \mathcal{H}$. Moreover, the cone γ_{p_i} is the "floor" of the region F_i. By Observation 6.4.3 no region of $\mathcal{F}$ intersects a cone of the ε-net N_P. By the properties of ε-nets, we obtain the following result.

Observation 6.4.4. *For each $1 \leq i \leq t$, at most εn cones of Γ intersect F_i.*

We now compute an ε-net of S that will allow us to complete the partition. Using the set of ranges $\mathcal{G}_S$ defined in Section 6.3, compute in $O(n)$ time an ε-net N_S of $(S, \mathcal{G}_S)$. For each segment s of N_S, compute $B_P(s)$ in $O(n)$ time [70] and mark the points of P that lie on the boundary of this circle. Let $r_{\min}$ be the radius of the minimum circle among the computed P-circles and let P^* be the set of marked points. Therefore, the interior of $\Lambda_{r_{\min}}(P^*)$ intersects no segment of N_S. Let $(v_1, \ldots, v_k)$ be the set of vertices of $\Lambda_{r_{\min}}(P^*)$ given in clockwise order. Triangulate $\Lambda_{r_{\min}}(P^*)$ by

joining c_P with every vertex of $\Lambda_{r_{\min}}(P^*)$ and let $\mathcal{A} \subset \mathcal{G}$ be the obtained set of k *slices* (two sides are straight-line segments and one an arc of a circle). Therefore, no segment of N_S intersects the interior of a slice in $\mathcal{A}$. Thus, at most εm segments of S intersect each slice in $\mathcal{A}$.

Intersect each region of $\mathcal{F}$ with the vertical lifting of each slice in $\mathcal{A}$ and let $\mathcal{Y} = \{Y_1, \ldots, Y_h\}$ be the set containing all the obtained intersections. That is, each Y_j is the intersection of some region F_i and the vertical lifting of some slice of $\mathcal{A}$. Since $|\mathcal{F}| = O(1)$ and $|\mathcal{A}| = O(1)$, $\mathcal{Y}$ can be computed in constant time.

We are ready to provide the proof of Lemma 6.4.1.

Proof of Lemma 6.4.1. For every $1 \leq i \leq h$, let $P'_i = \{p \in P : \gamma_p \cap Y_i \neq \emptyset\}$ and note that P'_i can be computed in $O(n)$ time. Since $Y_i \subset F$ for some $F \in \mathcal{F}$, by Observation 6.4.4, $|P'_i| < \varepsilon n$.

Let $S'_i = \{s \in S : VL(s) \cap Y_i \neq \emptyset\}$ and note that S'_i can be computed in $O(m)$ time. Since the orthogonal projection of Y_i onto Z_0 is contained in some slice $A \in \mathcal{A}$ and the interior of A intersects no segment of N_S, $|S'_i| < \varepsilon m$. Let r^* be the radius of $B_P(S)$. Since $b_P(S)$ lies in $\Lambda_{r^*}(P)$ and $\Lambda_{r^*}(P) \subset \Lambda_r(P) \subset \Lambda_r(P^*)$, the point $b_P(S)$ lies inside $\Lambda_r(P^*)$.

Let $s^* \in S$ be the segment where $b_P(S)$ lies and let $p \in P$ be a point on the boundary of $B_P(S)$. We claim that s^* and p belong to the same subproblem, i.e., s^* and p belong to S'_j and P'_j, respectively, for some $1 \leq j \leq k$. Notice that if this claim is true, then all the points of P through which $B_P(S)$ passes belong to P'_j. That is, $B_{P'_j}(S'_j)$ and $B_P(S)$ are defined as the circles with center on s^* passing through the same set of points, i.e., $B_{P'_j}(S'_j) = B_P(S)$. Thus, by computing $B_{P'_i}(S'_i)$ for each $1 \leq i \leq k$, the minimum P-circle with center on S can be obtained by choosing the minimum among $B_{P'_1}(S'_1), \ldots, B_{P'_h}(S'_j)$.

We proceed now to prove that s^* and p belong to the same subproblem. Since $b_P(S)$ lies in $\Lambda_r(P^*)$, $b_P(S)$ lies inside the projection of Y_j for some $1 \leq j \leq h$. i.e, $s^* \in S'_j$. Consider the ray σ shooting upwards (perpendicular to Z_0) from $b_P(S)$. Since p lies on the boundary of $B_P(S)$, p is farther away from $b_P(S)$ than any other point of P. That is, γ_p is the last cone intersected by σ (there may be other cones intersected at the same time). Therefore, γ_p intersects Y_j and consequently $p \in P'_j$. $\square$

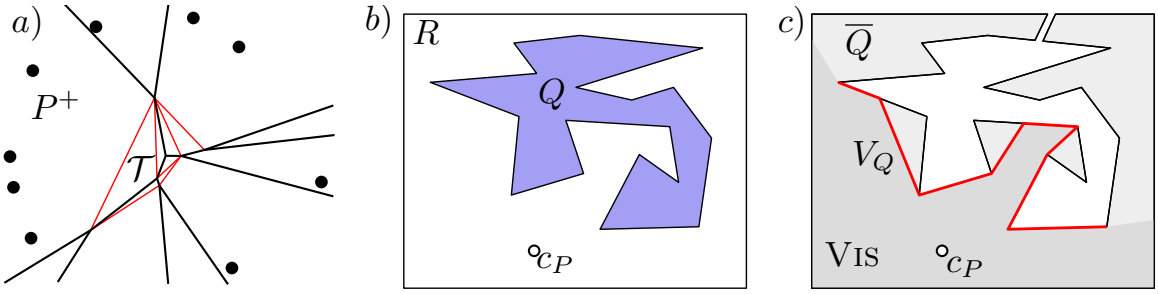

Figure 6.2: *a*) The farthest-point Voronoi diagram of P^+ extended to a triangulation $\mathcal{T}$ by adding the red edges. *b*) Simple polygon Q and the rectangle R containing both c_P and Q. By removing Q from R we obtain a polygon with one hole. *c*) After connecting this hole with the boundary we obtain a simple polygon $\overline{Q}$ containing c_P in its interior. In red the polygonal chain V_Q obtained after computing the visibility polygon Vis of $\overline{Q}$ from c_P and removing the edges adjacent to the boundary of R.

By Lemmas 6.3.9 and 6.4.1, we can use Lemma 3.2.1 to obtain the following result.

Theorem 6.4.5. *Given a set P of n points and a set S of m segments in the plane, the (P, S)-optimization problem can be solved in expected $O((n+m)\log\omega)$ time where $\omega = \min\{n, m\}$.*

6.5 Constraining to a simple polygon

Let $Q = (v_1, \ldots, v_m)$ be a simple polygon. In this section we present an algorithm to find the minimum P-circle with center in Q running in $O(m+n\log n)$ time. To achieve this running time, we use the underlying structure given by the order of the vertices along the boundary of Q. We present an algorithm for the $(P, Q)_r$-decision problem running in $O(n+m)$ time after computing the farthest-point Voronoi diagram of P (see Section 3.4). Therefore, by Lemma 6.4.1 we can use Lemma 3.2.1 to obtain an algorithm for the (P, Q)-optimization problem running in expected $O(m + n\log n)$ time. This results improve upon Theorem 6.4.5 whenever $m > n$.

6.5.1 Star-shaped regions

A polygonal chain N is *star-shaped* if there exists a set of points called its *kernel* such that every point on this chain is visible from every point x in its kernel, i.e., the

open segment joining x with every point on N does not intersect N. A key result to improve the algorithm presented in Section 6.3 is the following.

Lemma 6.5.1. *Let $r > r_P$. Given a star-shaped chain N of m vertices with c_P in its kernel and $\Lambda_r(P)$ as the sequence of the h vertices along its boundary, we can decide if N intersects $\Lambda_r(P)$ in $O(h+m)$ time.*

Proof. Let $N = (v_1, \ldots, v_m)$, let $(b_1, \ldots, b_h)$ be the sequence of vertices along the boundary of $\Lambda_r(P)$ and assume that both sequences are given in counterclockwise order. Given a point x in the plane, let $\theta(x)$ be the angle $0 \leq \theta(x) < 2\pi$ that the ray emanating from c_P and passing through x makes with the positive x-axis. Let e_i denote the edge $[v_i, v_{i+1}]$ and let α_j denote the arc (b_j, b_{j+1}). Assume *wlog* that $\theta(v_1) = 0$ and assume that α_1 is the arc intersected by the ray emanating from c_P that passes through v_1. Otherwise, relabel the vertices of $\Lambda_r(P)$ in $O(h)$ time. If at some point during the execution the algorithm detects an intersection, finish and report a positive answer to the intersection problem. The algorithm works as follows.

Step 0: If v_1 lies inside $\Lambda_r(P)$ finish and report that N intersects $\Lambda_r(P)$. Otherwise let $i = j = 1$ and proceed with Step 1.

Step 1: Test the edge e_j and the arc α_i for intersection. If $\theta(v_{j+1}) \leq \theta(b_{i+1})$ proceed and test the next edge of N. Let $j \leftarrow j+1$ and repeat Step 1. Otherwise, if $\theta(v_{j+1}) > \theta(b_{i+1})$ proceed with the next arc of $\Lambda_r(P)$. Let $i \leftarrow i+1$ and repeat Step 1.

If at some point $j > k$ (we have no edges of N left to test) finish and report that no intersection exists. On each step of the algorithm we advance and consider the next arc or the next edge either on $\Lambda_r(P)$ or on N, respectively. Therefore, this algorithm has an $O(h+m)$ running time. By walking the two sequence of vertices in clockwise order, we test every edge e on N only with the arcs on the boundary of $\Lambda_r(P)$ that intersect the triangle $T = CH(e \cup \{c_P\})$. Therefore, we considered every possible intersection between the boundary of $\Lambda_r(P)$ and N. □

6.5.2 The decision problem on a simple polygon

We will use the farthest-point Voronoi diagram to speed up the decision algorithm. Start by computing C_P and its center c_P. If c_P lies in the interior of Q, then C_P is the trivial solution of the optimization problem. Thus, we will assume from now on that c_P does not belong to Q. Compute $\mathcal{V}(P)$ in $O(n \log n)$ time using Theorem 3.4.6. Assume that each edge of $\mathcal{V}(P)$ has pointers to the points of P defining it.

Lemma 6.5.2. *After computing the farthest-point Voronoi diagram of P in $O(n \log n)$ time, we can compute $\Lambda_r(P)$ in $O(n)$ time for any given radius $r > r_P$.*

Proof. By Observation 6.2.2, for each point x on the boundary of $\Lambda_r(P)$, $\rho(x) = r$. Moreover, as long as x belongs to $R(p)$ for some $p \in P$, x must lie on $\alpha_r(p, P)$ by Observation 6.4.2. Consequently, the vertices of $\Lambda_r(P)$ are the points on the tree $\mathcal{V}(P)$ such that their value under ρ is equal to r.

Thus, if we determine all vertices of $\Lambda_r(P)$ in clockwise order, we can construct $\Lambda_r(P)$ by adding the arcs of fixed curvature between them in $O(n)$ time. Let e be an edge of $\mathcal{V}(P)$ and notice that it is contained in the bisector of two points p and q of P. If x is a point lying on e, then $\rho(x) = d(x, p) = d(x, q)$. As we assumed that e has pointers to both p and q, in $O(1)$ time we can determine if there is a point x lying on e such that $\rho(x) = r$. By walking along an Eulerian tour of $\mathcal{V}(P)$ and testing every edge on it, we can compute the set of vertices of $\Lambda_r(P)$ in counterclockwise order in $O(n)$ time. □

Given $r > r_P$, compute $\Lambda_r(P)$ in $O(n)$ time using Lemma 6.5.2. Our objective is to decide if there is an edge of Q that intersects $\Lambda_r(P)$. In this section we show how to do it in $O(m + n)$ time.

Let R be a sufficiently large rectangle to enclose Q and c_P in its interior. Note that R can be computed in $O(m)$ time. Let $\overline{Q} = R - int(Q)$ which is a simple polygon with one hole. Shoot a ray σ from c_P that intersects the interior of the hole. Let x and x' be the two intersections lying farther from c_P between σ and the boundary of $\overline{Q}$ (one of them on the boundary of R). Remove the hole from $\overline{Q}$ by joining x and x' with a sufficiently small corridor. This way, $\overline{Q}$ becomes a simple polygon without

holes that can be computed in $O(m)$ time. Moreover, c_P lies in the interior of $\overline{Q}$; see Fig. 6.4.

Compute the visibility polygon Vis of $\overline{Q}$ from the point c_P in $O(m)$ time using the algorithm from Joe and Simpson [58]. Finally, let V_Q be the polygonal chain obtained by removing the edges of the boundary of Vis that have an endpoint lying on the boundary of the rectangle R (there may be none). Since every point in the boundary of $\Lambda_r(P)$ is visible from c_P, $\Lambda_r(P)$ intersects Q if and only if $\Lambda_r(P)$ intersects V_Q. As V_Q is star-shaped and has c_P in its kernel, by Lemma 6.5.1 we can decide if V_Q interests $\Lambda_r(P)$ in $O(n+m)$ time. Therefore, we can solve the $(P,Q)_r$-decision problem in linear time. By considering the set of segments on the boundary of Q, we can use Lemma 6.4.1 to construct $O(1)$ subproblems so that the solution of the (P,Q)-optimization problem is the minimum among the subproblems solutions. Consequently, we can use Lemma 3.2.1 to obtain the following result.

Theorem 6.5.3. *Given a set P of n points and a simple polygon Q of m vertices, the (P,Q)-optimization problem can be solved in expected $O(m+n\log n)$ time.*

Notice that the bottleneck of this algorithm is the construction of the farthest-point Voronoi diagram. Therefore, if P is the set of vertices of a convex polygon, we can compute its farthest-point Voronoi diagram in linear time using Theorem 3.4.6.

Corollary 6.5.4. *Let N and Q be two convex polygons on n and m vertices, respectively. The minimum enclosing circle of N, whose center is constrained to lie in Q, can be found in expected $\Theta(m+n)$ time.*

6.6 Lower bounds

In this section,we prove lower bounds for the decision problems: We show inputs where the decision problem is equivalent to answering a membership query in a set with "many" disjoint components. We then use Theorem 3.3.1 to obtain lower bounds for any decision algorithm that solves this membership problem.

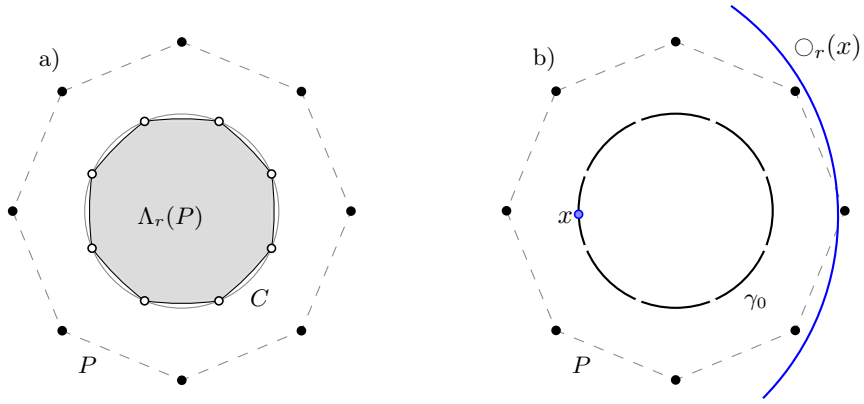

Figure 6.3: *a*) The construction presented in Section 6.6.1. A set P representing the vertices of a regular n-gon. The region $\Lambda_r(P)$ is enclosed by C and intersects it at n points being the n vertices of $\Lambda_r(P)$. *b*) The removal of the set $\Lambda_r(P)$ from C splits it into n disconnected components that define the set φ_0. No point on φ_0 supports a P-circle of radius r.

6.6.1 Lower bounds when constraining to a set of points

Let $r > 0$. In this section, we construct a set of points P such that for any point set M (with certain constraints), the $(P, M)_r$-decision problem has a lower bound of $\Omega(m \log n)$.

Let r' be a number such that $0 < r' < r$. Let P be the set of vertices of a regular n-gon circumscribed on a circle of radius r'. Because $r > r'$, $\Lambda_r(P)$ is a non-empty convex region whose boundary is composed of circular arcs. Given a point set M, the $(P, M)_r$-decision problem has an affirmative answer if and only if there is a point of M lying in $\Lambda_r(P)$ (see Observation 6.2.2).

Let C be the circumcircle of the vertices of $\Lambda_r(P)$. Partition this circle into $\varphi_1 = C \cap \Lambda_r(P)$ and $\varphi_0 = C - \varphi_1$. Because φ_1 consists of exactly n points being the vertices of $\Lambda_r(P)$, φ_0 consists of n disconnected open arcs all lying outside of $\Lambda_r(P)$. Moreover, a point on C supports a P-circle of radius r if and only if it lies on φ_1; see Fig. 6.3(a).

Theorem 6.6.1. *Let P be a set of n points and a let M be a set of m points (m segments or m lines). Given a radius r, the $(P, M)_r$-decision problem has a lower bound of $\Omega(m \log n)$ in the algebraic decision-tree model.*

Proof. Consider the restriction of the $(P, M)_r$-decision problem where M is constrained to be a subset of C. Notice that any lower bound for this restricted problem is also a lower bound for the general decision problem.

Consider the space of all multi-sets of m points constrained to lie on C. Notice that an input of m points can also be seen as a point in $\mathbb{R}^{2m}$. That is, the space of inputs for the restricted decision problem defines a subset C^m of $\mathbb{R}^{2m}$. Moreover, C^m can be partitioned into to subsets, the "yes" and the "no" sets (with respect to the $(P, M)_r$-decision problem), where the "no" region is equal to $\gamma_0 = \varphi_0^m$. That is, a point $(x_1, y_1, x_2, y_2, \ldots, x_m, y_m)$ lies in the "no" region γ_0 if for every index $1 \leq j \leq m$, the point (x_j, y_j) lies inside $\varphi_0 \subset C$.

Because φ_0 contains n disjoint components, $\gamma_0 = \varphi_0^m$ contains $O(n^m)$ disjoint components being the product-space of m copies of φ_0. Recall that the $(P, M)_r$-decision problem is equivalent to the membership problem in φ_0^m. Therefore, by Theorem 3.3.1 we obtain a lower bound of $\Omega(m \log n)$ for every decision algorithm in the algebraic decision-tree model. Notice that this lower bound implies a lower bound for the decision problem where the center is constrained to lie on a set of segments. This proof can be slightly modified to work when the P-circle is constrained to have its center on a set of lines by constraining each line of the input to be tangent to C. □

6.6.2 Lower bound when constraining to a simple polygon

Let $r > 0$. In this section, we construct a simple m-gon Q such that for a any input P on n points, the $(P, Q)_r$-decision problem has a lower bound of $\Omega(m + n \log n)$.

Let $r_\varepsilon = r + \varepsilon$ for some $\varepsilon > 0$ to be specified later. Let $N = \{p_1, \ldots, p_n\}$ be the set of vertices of a regular n-gon whose circumcircle C has radius r and center on c. Because r_ε is greater than r, the radius of C, $\Lambda_{r_\varepsilon}(N)$ is non-empty.

Consider the middle points of every arc along $\Lambda_{r_\varepsilon}(N)$ and label them so that m_i is the middle point on the arc opposite to p_i. Let C' be a circle with center on c and radius $2r$. For each $1 \leq i \leq n$, let q_i be the point of intersection between C' and the ray from c to p_i. Let Q' be a star-shaped polygon with vertex set $\{m_1, \ldots, m_n\} \cup \{q_1, \ldots, q_n\}$ where edges connect consecutive vertices in the radial order around c; see

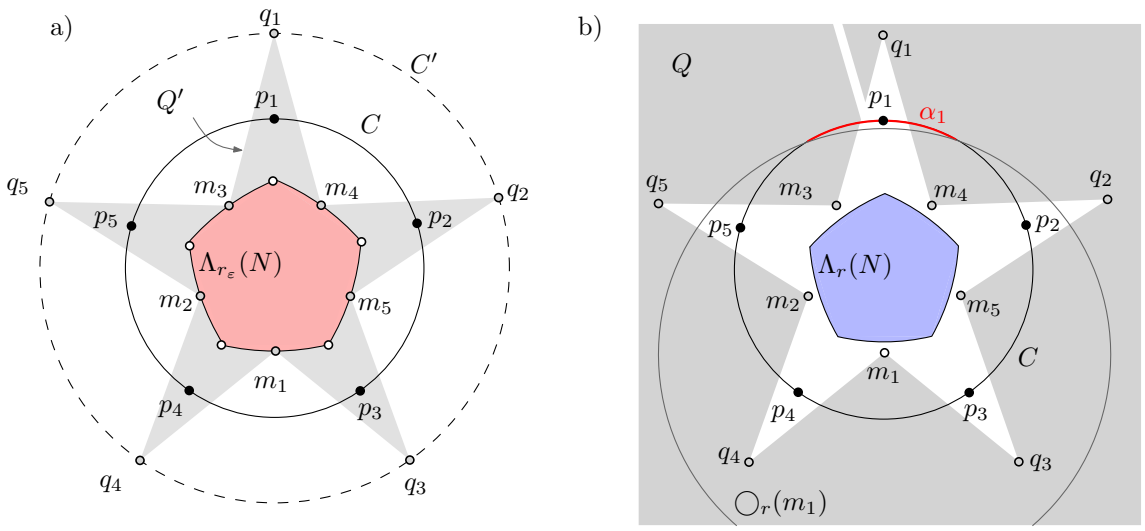

Figure 6.4: *a*) The construction of the star-shaped polygon Q' used in Section 6.6.2. *b*) The polygon Q constructed in Section 6.6.2 being disjoint from $\Lambda_{r_\varepsilon}(P)$. The arc α_6 (in red) is the arc of C excluded by $\bigcirc_{r_\varepsilon}(q_6)$. Moreover, α_6 contains exactly one point (say p_1) of P.

Fig. 6.4(a).

Let R be a sufficiently large rectangle to enclose Q' and let $Q = R \setminus \text{INT}(Q')$. Remove the star-shaped hole of Q by connecting the boundary of R with an edge of Q using a small corridor; see Fig. 6.4(b) for an illustration.

Observation 6.6.2. *The points $m_1, \ldots, m_n$ are the only points of intersection between Q and $\Lambda_{r_\varepsilon}(N)$.*

Consider the restriction of the $(P, Q)_r$-decision problem, where every point of P is constrained to lie on C. Note that any lower bound for this restricted problem is also a lower bound for the general problem. We show now a lower bound of $\Omega(m+n\log n)$ for this restricted problem.

Recall that every input of n points constrained to lie on C can be seen as a point in $C^n \subset \mathbb{R}^{2n}$ and vice versa. Formally, for each point $X = (x_1, y_1, \ldots, x_n, y_n) \in C^n$ let $P_X = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be the set of n points contained in C that corresponds to X.

Let $\gamma_0 \subset C^n$ such that $X \in \gamma_0$ if and only if the $(P_X, Q)_r$-decision problem has a negative answer. Let $\gamma_1 = C^n - \gamma_0$ be the set defining the positive answers.

Because $r < r_\varepsilon$, Observation 6.6.2 implies that $\Lambda_r(N)$ does not intersect Q, i.e.,

$N \in \gamma_0$. Notice that every permutation of the same input of n points induces a different point in $\mathbb{R}^{2n}$. That is, a set of n points in the plane can be represented by $n!$ different points in $\mathbb{R}^{2n}$. Therefore, every one of the $n!$ points representing N in $\mathbb{R}^{2n}$ lies in γ_0.

We claim that each of these $n!$ points lie in different connected components of γ_0. The idea is that any continuous transformation between two of these permutations will reach a state in which the point it represents becomes part of γ_1. A formal proof is provided below.

Theorem 6.6.3. *Let P be a set of n points and let Q be a simple polygon on m vertices such that $m \geq n$. Given a radius r, the $(P, Q)_r$-decision problem has a lower bound of $\Omega(m + n \log n)$ in the algebraic decision-tree model.*

Proof. Since there are $n!$ permutations of N and each of them belongs to a different connected component in γ_0, we conclude that γ_0 contains at least $n!$ disjoint connected components. Because the $(P, M)_r$-decision problem is equivalent to the membership problem in γ_0, by Theorem 3.3.1 we obtain a lower bound of $\Omega(n \log n)$ for the restricted $(P, M)_r$-decision problem in the algebraic decision-tree model. To obtain this lower bound, the m-gon Q needs to have at least n vertices, i.e., $m \geq n$.

Finally, notice that any decision algorithm has also a lower bound of $\Omega(m)$ since every vertex of Q has to be considered. Otherwise, an adversary could perturb a vertex so that the solution switches from a negative to a positive answer without affecting the execution of the algorithm. □

This theorem provides the same lower bound for the $(P, S)_r$-decision problem, where S is a set of m segments. Furthermore, this proof can be easily modified to bound the $(P, M)_r$-decision problem where M is a set of m points by considering the vertices of Q with even indices as the points of M.

Many disjoint components with negative answer

In this section, we show that the $n!$ points in C^n that represent the different permutations of N lie in $n!$ disconnected components.

Note that every point of N lies inside $\bigcirc_r(m_i)$ except for p_i, i.e., there is a connected portion of C excluded by $\bigcirc_r(m_i)$. For every $1 \leq i \leq n$, let α_i be the arc of C excluded by the circle $\bigcirc_r(m_i)$ where p_i lies on α_i. By letting ε sufficiently small, α_i is disjoint from α_j for any $i \neq j$. Moreover, every point lying on $C \setminus \alpha_i$ is enclosed by $\bigcirc_r(m_i)$.

Observation 6.6.4. *Let X be a point in C^n. If the set P_X contains no point lying on the arc α_i, for some $1 \leq i \leq n$, then $X \in \gamma_1$.*

Proof. If P_X has no point lying on α_i for some $1 \leq i \leq n$, then there exists $m_i \in Q$ such that $\bigcirc_r(m_i)$ is a P_X-circle of radius r, i.e., P_X belongs to γ_1. □

Let $\mathcal{X}_N$ be the set of $n!$ points of C^n such that for each $X \in \mathcal{X}_N$, $P_X = N$, i.e., the $n!$ points that correspond to the permutations of N. We are now ready to complete the proof of Theorem 6.6.3 by showing that each two points of $\mathcal{X}_N$ lie in different connected components of γ_0.

Lemma 6.6.5. *Let X_0 and X_1 be two points in $\mathcal{X}_N$. Any continuous path contained in C^n joining X_0 with X_1 intersects γ_1.*

Proof. Let $X(t)$ be a continuous curve contained in C^n joining X_0 with X_1, $t \in [0, 1]$. That is, $X(0) = X_0$, $X(1) = X_1$ and for every $t \in [0, 1]$, $X(t)$ is a set of n points contained in C^n. Notice that for each point $p \in X_0$, $P(t)$ induces a continuous curve $\sigma_p(t)$ contained in C such that $\sigma_p(0) = p$ and $\sigma_p(1) \in X_1$. Since X_0 and X_1 are different permutations of P, there exists a point in X_0 that needs to change position during the continuous transition. In order to do that, the points changing position will move outside the arc α_i where it originally lied on and will reach a different arc α_j at the end of the transition, $i \neq j$. Let t_0 be the first time when a point of $P(t)$ abandons its arc and assume that α_j is the abandoned arc at time t_0. Therefore, by choosing ε sufficiently small, at time t_0 no point of $P(t_0)$ lies on α_j. Thus, by Observation 6.6.4, $P(t_0)$ belongs to γ_1. □

6.6.3 Another lower bound when constraining to sets of points

Let A and B be two sets of m and n real numbers in $[0, 1]$, respectively, such that $m \leq n$ and A is sorted in increasing order. The *A-B-subset problem* asks if A is a subset of B.

In this section, we show that any A-B-subset problem can be reduced in linear time to a $(P,Q)_r$-decision problem for some simple polygon Q. Hence, any lower bound for the A-B-subset problem is a lower bound for the $(P,Q)_r$-decision problem. We show that the A-B-subset problem has a lower bound of $\Omega(n \log m)$ which yields the same lower bound for the $(P,Q)_r$-decision problem.

The reduction

We start by showing the connection between the A-B-subset problem and the $(P,Q)_r$-decision problem.

Lemma 6.6.6. *Any A-B-subset problem can be reduced in linear time into a $(P,M)_r$-decision problem for a set M of m points (or a $(P,Q)_r$-decision problem for a simple m-gon Q).*

Proof. Assume that $A = \{a_1, \ldots, a_m\}$ and that $B = \{b_1, \ldots, b_n\}$. Let C be a circle with center on $c = (0,0)$ and radius r. For any point w on C, let $\theta(w)$ be the angle that the ray with apex on c passing through w forms with the positive x-axis. For each $a_i \in A$, let q_i be the point on C such that $\theta(q_i) = \pi a_i$. Let $M = \{q_1, \ldots, q_n\}$. For each $q_i \in M$, let w_i be the point lying opposite to q_i on the circle C, that is the point on C such that $\theta(w_i) = \pi + \theta(q_i)$. Therefore, $W = \{w_1, \ldots, w_n\}$ is a set of n points lying on C (actually on the lower half of the circle while the points of M lie on the upper half).

For each $b_i \in B$, let p_i be the point on C such that $\theta(p_i) = \pi + b_i\pi$ and let $P = \{p_1, \ldots, p_n\}$. We claim that the radius of $B_P(M)$ is equal to $2r$ if and only if $A \subseteq B$. Note that $2r$ is an upper bound for the radius $B_P(M)$ since the circle $\bigcirc_{2r}(q_i)$ is a P-circle for every $q_i \in M$. Moreover, the radius of $B_P(M)$ is exactly $2r$ if and only if there is a point in P lying at distance $2r$ from every point in M, i.e., if and only P contains every point of W. However, a point $w_i \in W$ belongs to P if and only if there is a point $p_j \in P$ such that $\pi + \pi a_i = \pi + \theta(q_i) = \theta(w_i) = \theta(p_j) = \pi + b_j\pi$, i.e., $w_i \in P$ if and only if there is $b_j \in B$ such that $a_i = b_j$. Since $w_i \in P$ if and only if $a_i \in A$, we conclude that $B_P(M)$ has radius $2r$ if and only if $A \subseteq B$.

An analogous proof follows for simple polygons by considering a serrated simple polygon Q having, alternately, one vertex coinciding with a point in M and the next

one lying outside of C. Since A is sorted in increasing order, the construction of Q can be performed in linear time. □

By Lemma 6.6.6, any lower bound for the A-B-subset problem is a lower bound for the $(P, M)_r$-decision problem (or the $(P, Q)_r$-decision problem, where Q is a simple polygon). In fact, the lower bound we prove for the A-B-subset problem considers arbitrary sets of real numbers. Furthermore, the lower bound holds when A is given as a fixed sorted set prior to the design of the algorithm.

6.6.4 The A-B-subset problem

Let $A = \{a_1, \ldots, a_m\}$ be a sorted set of m distinct real numbers and think of it as a point in $\mathbb{R}^m$. Note that any input set of n real numbers can be represented by a point in $\mathbb{R}^n$. Let γ_1 be the subspace of $\mathbb{R}^n$ containing all points representing a set B such that $A \subseteq B$, i.e., the "yes" region.

An *A-constraint* is an equation of the form $(x_i = a_j)$ for some $1 \leq i \leq n, 1 \leq j \leq m$. Two A-constraints $(x_i = a_j)$ and $(x_h = a_k)$ are compatible if $i \neq h$ (j may be equal to k). A point $X \in \mathbb{R}^n$ *satisfies* an A-constraint $(x_i = a_j)$ if its i-th coordinate is equal to a_j. A set φ of pairwise compatible A-constraints is *complete* if it contains exactly one A-constraint of the form $(x_j = a_i)$ for every $a_i \in A$, i.e., it contains exactly m A-constraints, one for each element of A. Given a complete set φ of A-constraints, let $K_\varphi = \{X \in \mathbb{R}^n : X \text{ satisfies every } A\text{-constraint in } \varphi\}$.

Notice that $dim(K_\varphi) = n - m$ and $codim(K_\varphi) = m$. Moreover, if a point B belongs to K_φ, then $A \subseteq B$, i.e., every point in K_φ belongs to γ_1. Additionally, if $A \subseteq B$, then B belongs to some $K_{\varphi'}$ for some complete set φ' of A-constraints. Therefore, if we let $\mathscr{A} = \{K_\varphi : \varphi \text{ is a complete set of } A\text{-constraints}\}$, then $\gamma_1 = \cup\mathscr{A}$. Note that $K_\varphi \cap K_{\varphi'} = K_{\varphi \cup \varphi'}$ since the elements in $K_\varphi \cap K_{\varphi'}$ have to satisfy both sets of constraints.

Consider the poset induced by the intersection semilattice of $\mathscr{A}$, denoted by $\mathscr{L}_\mathscr{A}$, ordered by the reverse inclusion and let $\hat{0} = \mathbb{R}^n$ denote the minimum of this poset. Since the elements of $\mathscr{L}_\mathscr{A}$ are the intersection of the elements of $\mathscr{A}$, it holds that for every $K \in \mathscr{L}_\mathscr{A}$, $K = K_\varphi$ for some set φ of A-constraints such that φ contains a complete set of A-constraints ($|\varphi| \geq m$).

Consider the Möebius function μ defined on the elements of $\mathscr{L}_{\mathscr{A}}$. Recall that $\mu(\hat{0}) = \mu(\hat{0}, \hat{0}) = 1$ and for any $K \in \mathscr{L}_{\mathscr{A}}$ such that $\hat{0} \neq K$, $\mu(K) = \mu(\hat{0}, K) = -\sum_{K \subsetneq G} \mu(G)$.

Lemma 6.6.7. *For every $K_\varphi \in \mathscr{L}_{\mathscr{A}}$ such that $K_\varphi \neq \hat{0}$, $\mu(K_\varphi) = (-1)^{codim(K_\varphi)-m-1}$.*

Proof. We prove it by induction on the co-dimension of K_φ. For the base case, consider a set K_φ such that φ is a complete set of m A-constraints, i.e., $K_\varphi \in \mathscr{A}$. Since $\hat{0}$ is the only element in $\mathscr{L}_{\mathscr{A}}$ that constains K_φ, $\mu(K_\varphi) = -\mu(\hat{0}) = -1 = (-1)^{m-m-1}$.

Let $K_\varphi \in \mathscr{A}$ such that $|\varphi| = k > m$, i.e., $codim(K_\varphi) = k$. By definition,

$$\mu(K_\varphi) = -\left(\sum_{K_\varphi \subsetneq K_\beta} \mu(K_\beta) \right).$$

Notice that for every $K_\beta \in \mathscr{L}_{\mathscr{A}}$, $K_\varphi \subsetneq K_\beta$ if and only if β is a set of A-constraints such that $\beta \subsetneq \varphi$ and $|\beta| \geq m$. In other words, by dropping some A-constraints of φ (but not more that $k - m$) we obtain a new affine space containing K_φ (we have to drop at least one constraint to get a proper superset). For every $1 \leq i \leq m$, let φ_i be the subset of φ containing all the A-constraints of the form $(x_j = a_i)$, i.e., the set of constraints associated with the item $a_i \in A$. That is, $\varphi = \varphi_1 \cup \ldots \cup \varphi_m$. Consequently, $K_\varphi \subsetneq K_\beta$ if and only if $\beta = \cup_{i=1}^{m} \beta_i$ where $\beta_i \subseteq \varphi_i$, $|\beta_i| \geq 1$ and not every β_i is equal to φ_i (i.e., $\sum_{j=1}^{m} |\beta_j| < \sum_{j=1}^{m} |\varphi_j|$). Using this notion, we obtain that

$$\mu(K_\varphi) = -\left(\mu(\hat{0}) + \sum_{\substack{\beta_i \subseteq \varphi_i, |\beta_i| \geq 1 \\ 1 \leq i \leq m \\ \sum |\beta_j| < \sum |\varphi_j|}} \mu(K_{\beta_1 \cup \ldots \cup \beta_m}) \right).$$

However, $codim(K_{\beta_1 \cup \ldots \cup \beta_m}) = \sum_{j=1}^{m} |\beta_j|$. Therefore, by induction hypothesis $\mu(K_{\beta_1 \cup \ldots \cup \beta_m}) = (-1)^{\sum_{j=1}^{m} |\beta_j| - m - 1} = (-1)^{|\beta_1|} \ldots (-1)^{|\beta_m|} (-1)^{-m-1}$. Using this fact and rearranging the terms,

$$\mu(K_\varphi) = -\left(\mu(\hat{0}) + (-1)^{-m-1} \sum_{\substack{\beta_i \subseteq \varphi_i, |\beta_i| \geq 1 \\ 1 \leq i \leq m \\ \sum |\beta_j| < \sum |\varphi_j|}} (-1)^{|\beta_1|} \dots (-1)^{|\beta_m|} \right).$$

In order to simplify this expression, we would like to drop the constraint in the summation given by $\sum_{j=1}^{m} |\beta_j| < \sum_{j=1}^{m} |\varphi_j|$. To do this, notice that

$$(-1)^{\sum_{i=1}^{m} |\varphi_i|} + \sum_{\substack{\beta_i \subseteq \varphi_i, |\beta_i| \geq 1 \\ 1 \leq i \leq m \\ \sum |\beta_j| < \sum |\varphi_j|}} (-1)^{|\beta_1|} \dots (-1)^{|\beta_m|} = \sum_{\substack{\beta_i \subseteq \varphi_i, |\beta_i| \geq 1 \\ 1 \leq i \leq m}} (-1)^{|\beta_1|} \dots (-1)^{|\beta_m|}.$$

Let $z = (-1)^{\sum_{i=1}^{m} |\varphi_i| - m - 1} = (-1)^{k-m-1}$. Therefore, if we add and subtract the term z inside the brackets, then we complete the summation and are left with one negative z.

$$\mu(K_\varphi) = -\left(\mu(\hat{0}) - z + (-1)^{-m-1} \sum_{\substack{\beta_i \subseteq \varphi_i, |\beta_i| \geq 1 \\ 1 \leq i \leq m}} (-1)^{|\beta_1|} \dots (-1)^{|\beta_m|} \right).$$

Since the term $(-1)^{|\beta_i|}$ depends only on the summation on i and since $\mu(\hat{0}) = 1$, we can rearrange the terms as follows:

$$\mu(K_\varphi) = -\left(1 - z + (-1)^{-m-1} \sum_{\substack{\beta_1 \subseteq \varphi_1 \\ |\beta_1| \geq 1}} (-1)^{|\beta_1|} \sum_{\substack{\beta_2 \subseteq \varphi_2 \\ |\beta_2| \geq 1}} (-1)^{|\beta_2|} \dots \sum_{\substack{\beta_m \subseteq \varphi_m \\ |\beta_m| \geq 1}} (-1)^{|\beta_m|} \right).$$

Notice that for any $1 < j \leq |\varphi_i|$, there are $\binom{|\varphi_i|}{j}$ ways to choose j A-constraints from φ_i. Moreover, since $\sum_{j=0}^{r} \binom{r}{j} (-1)^j = 0$ for any integer r, we have that

$$\sum_{\substack{\beta_i \subseteq \varphi_i \\ |\beta_i| \geq 1}} (-1)^{|\beta_i|} = \sum_{j=1}^{|\varphi_i|} \binom{|\varphi_i|}{j} (-1)^j = -1.$$

Therefore, $\mu(K_\varphi) = -(1 - z + (-1)^{m-m-1}) = -(1 - z - 1) = z = (-1)^{k-m-1}$. $\square$

Lemma 6.6.8. *Let n, m be two integers such that $n \geq m$. For any $A \in \mathbb{R}^m$ such that A is given in sorted order, there is a lower bound of $\Omega(n \log m)$ in the algebraic decision-tree model for the A-B-subset problem given any set $B \in \mathbb{R}^n$.*

Proof. Let $k = \lfloor n/m \rfloor$. We claim that there are $\Omega(\frac{n!}{(k!)^{m-1}(n-km)!})$ sets in $\mathscr{L}_{\mathscr{A}}$ with non-zero value under μ. Assuming this claim and using result (16) of [91], we obtain a lower bound of $\Omega(\log(\frac{n!}{(k!)^{m-1}(n-km)!}))$ for the membership problem in $\cup\mathscr{A} = \gamma_1$. Moreover, we have that

$$\log\left(\frac{n!}{(k!)^{m-1}(n-km)!}\right) = n\log n - (m-1)k\log k - (n-km)\log(n-km).$$

Since $k = \lfloor n/m \rfloor$, $n - km \leq k$ and hence $(n-km)\log(n-km) \leq k \log k$. Thus,

$$\log\left(\frac{n!}{(k!)^{m-1}(n-km)!}\right) \geq n\log n - mk\log k \geq n\log n - m(n/m)\log(n/m) = n\log m.$$

Consequently, we attain an $\Omega(n \log m)$ lower bound for the A-B-subset problem in the algebraic decision-tree model.

To prove our claim, we exhibit $\frac{n!}{(k!)^{m-1}(n-km)!}$ distinct elements of $\mathscr{L}_{\mathscr{A}}$ with non-zero value under μ. An *index sequence* is a sequence $I = (i_1, \ldots, i_m)$ of m distinct indices such that $1 \leq i_j \leq n$ for every $j \in \{1 \ldots, m\}$. Given an index sequence $I = (i_1, \ldots, i_m)$, let $\varphi(I) = \{(x_{i_j} = a_j) :$ the index i_j appears in $I\}$ be the complete set of A-constraints induced by I (i.e., the j-th index on the sequence defines an A-constraint associated with a_j). Moreover, if I and I' are disjoint index sequences (no index belongs to both sequences), then $\varphi(I) \cup \varphi(I')$ is a compatible set of A-constraints. Recall that $A \subseteq B$ if and only if there exists an index sequence I such that B, seen as a point in $\mathbb{R}^n$, belongs to $K_{\varphi(I)}$, i.e., if the point B satisfies all the constraints induced by I.

Given a set of pairwise disjoint index sequences $\mathcal{I} = \{I_1, \ldots, I_t\} \neq \emptyset$, let $\varphi_{\mathcal{I}} = \cup_{i=1}^t \varphi(I_i)$. Notice that $K_{\varphi_{\mathcal{I}}} = \cap_{i=1}^t K_{\varphi(I_i)}$ and hence $K_{\varphi_{\mathcal{I}}} \in \mathscr{L}_{\mathscr{A}}$. Let $\mathcal{I} = \{I_1, \ldots, I_k\}$ be a set of $k = \lfloor n/m \rfloor$ pairwise disjoint index sequences. Notice that $K_{\varphi_{\mathcal{I}}}$ has co-dimension km since every A-constraint restricts a different coordinate of every point in $K_{\varphi_{\mathcal{I}}}$. Additionally, since $K_{\varphi_{\mathcal{I}}} \in \mathscr{L}_{\mathscr{A}}$, by Lemma 6.6.7 we know that $\mu(K_{\varphi_{\mathcal{I}}}) =$

$(-1)^{km-m-1} \neq 0$. Therefore, every set of k pairwise disjoint index sequences induces a different element of $\mathscr{L}_{\mathscr{A}}$ with non-zero value under μ.

Using simple counting arguments, we can compute the number of k pairwise disjoint index sequences, and hence give a lower bound on the number of elements in $\mathscr{L}_{\mathscr{A}}$. Note that there are $\binom{n}{k}$ ways to choose the first element of each of the k index sequences (i.e., the elements that induce a constraint of the form $(x_j = a_1)$). Once the first elements are chosen, there are $\binom{n-k}{k}$ ways to choose the second element and so on. In this way, we obtain that there are $\binom{n}{k}\binom{n-k}{k}\binom{n-2k}{k}\ldots\binom{n-(m-1)k}{k} = \frac{n!}{(k!)^{m-1}(n-km)!}$ sets of k pairwise index sequences, each inducing a different element of $\mathscr{L}_{\mathscr{A}}$ with non-zero value under μ as claimed. □

Corollary 6.6.9. *Given a set P of n points and a simple polygon Q on m vertices (or a set of m segments or a set of m points), the $(P,Q)_r$-decision problem has a lower bound of $\Omega(n \log m)$ in the algebraic decision-tree model.*

Chapter 7

Geodesic Center

Let P be a simple polygon with n vertices. Given two points x, y in P (either on the boundary or in the interior), the *geodesic path* $\pi(x, y)$ is the shortest path contained in P connecting x with y. If the straight-line segment connecting x with y is contained in P, then $\pi(x, y)$ is a straight-line segment. Otherwise, $\pi(x, y)$ is a polygonal chain whose vertices (other than its endpoints) are reflex vertices of P. We refer the reader to [73] for more information on geodesic paths.

The *geodesic distance* between x and y, denoted by $|\pi(x, y)|$, is the sum of the Euclidean lengths of each segment in $\pi(x, y)$. Throughout this chapter, when referring to the distance between two points in P, we mean the geodesic distance between them.

Given a point $x \in P$, a (geodesic) *farthest neighbor* of x is a point $f_P(x)$ (or simply $f(x)$) of P whose geodesic distance to x is maximized. To ease the description, we assume that each vertex of P has a unique farthest neighbor. This *general position* condition was also assumed by Aronov et al. [7] and can be obtained by applying a slight perturbation to the positions of the vertices [43].

Let $F_P(x)$ be the function that maps each $x \in P$ to the distance to a farthest neighbor of x (i.e., $F_P(x) = |\pi(x, f(x))|$). A point $c_P \in P$ that minimizes $F_P(x)$ is called the *geodesic center* of P. Similarly, a point $s \in P$ that maximizes $F_P(x)$ (together with $f(s)$) is called a *geodesic diametral pair* and their distance is known as the *geodesic diameter*. Asano and Toussaint [9] showed that the geodesic center is unique (whereas it is easy to see that several geodesic diametral pairs may exist).

In this chapter we show how to compute the geodesic center of P in $O(n)$ time.

7.1 Outline

In order to compute the geodesic center, c_P, Pollack et al. [83] introduced a linear time *chord-oracle*. A *chord* of a polygon is an edge joining two non-adjacent vertices such

that the open segment connecting them is contained in the interior of the polygon.

Given a chord C that splits P into two sub-polygons, the oracle determines which sub-polygon contains c_P. Combining this operation with an efficient search on a triangulation of P, Pollack et al. narrow the search for c_P to be within a triangle (and afterwards find the center using optimization techniques). Their approach however, does not allow them to reduce the complexity of the problem in each iteration, and hence it runs in $\Theta(n \log n)$ time.

The general approach of our algorithm described in Section 7.6 is similar: partition P into $O(1)$ cells, use an oracle to determine which cell contains c_P, and recurse within the cell. Our approach differs, however, in two important aspects that allow us to speed-up the algorithm. First, we do not use the chords of a triangulation of P to partition the problem into cells. We use instead a cutting of a suitable set of chords. Secondly, we compute a set Σ of $O(n)$ functions, each defined in a triangular domain contained in P, such that their upper envelope, $\phi(x)$, coincides with $F_P(x)$. Thus, we can "ignore" the polygon P and focus only on finding the minimum of the function $\phi(x)$.

The search itself uses ε-nets and cutting techniques, which guarantee that both the size of the cell containing c_P and the number of functions of Σ defined in it decrease by a constant fraction (and thus lead to an overall linear-time algorithm). However, this search has two stopping conditions, (1) reach a subproblem of constant-size complexity, or (2) find a triangle containing c_P. In the latter case, we show that $\phi(x)$ is a convex function when restricted to this triangle. Therefore, to find its minimum we can use standard optimization techniques based on cuttings in $\mathbb{R}^3$. This procedure is described in Section 7.7.

The key to this approach lies in the computation of the functions of Σ and their triangular domains. Each function $g(x)$ of Σ is defined in a triangular domain $\triangle$ contained in P and is associated with a particular vertex w of P. Intuitively speaking, $g(x)$ maps points in $\triangle$ to their (geodesic) distance to w. We guarantee that, for each point $x \in P$, there is one function g defined in a triangle containing x, such that $g(x) = F_P(x)$. To compute these triangles and their corresponding functions, we proceed as follows.

In Section 7.2, we use the matrix search technique introduced by Hershberger and Suri [55] to decompose the boundary of P, denoted by ∂P, into connected edge-disjoint chains. Each chain is defined by either (1) a consecutive list of vertices that have the same farthest neighbor v (we say that v is *marked* if it has such a chain associated to it), or (2) an edge whose endpoints have different farthest neighbors (such edge is called a *transition edge*).

In Section 7.3, we consider each transition edge ab of ∂P independently and compute its *hourglass*. Intuitively, the hourglass of ab, H_{ab}, is the region of P between two chains, the edge ab and the chain of ∂P that contains the farthest neighbors of all points in ab. Inspired by a result of Suri [88], we show that the sum of the complexities of all hourglasses defined on a transition edge is $O(n)$. (The *combinatorial complexity*, or simply *complexity* of a geometric object, is the total number of vertices and edges that define it.) In addition, we provide a new technique to compute all these hourglasses in linear time.

While the hourglasses cover a big part of P, to complete the coverage we need to consider *funnels* from each marked vertex. Intuitively, the funnel of a marked vertex v is the region of P that contains all the paths from v to every point of ∂P that is farther from v than from any other vertex of P. In Section 7.4, we prove that the total complexity of all the funnels of all marked vertices is $O(n)$. Moreover, we provide an algorithm to compute all these funnels in linear time.

In Section 7.5 we show how to compute the functions in Σ and their respective triangles. We distinguish two cases: (1) Inside each hourglass H_{ab} of a transition edge, we use a technique introduced by Aronov et al. [7] that uses the shortest-path trees of a and b in H_{ab} to construct $O(|H_{ab}|)$ triangles with their respective functions (for more information on shortest-path trees refer to [51]). (2) Inside the funnel of each marked vertex v, we compute triangles that encode the distance from v. Moreover, we guarantee that these triangles cover every point of P whose farthest neighbor is v. Overall, we compute the $O(n)$ functions of Σ in linear time.

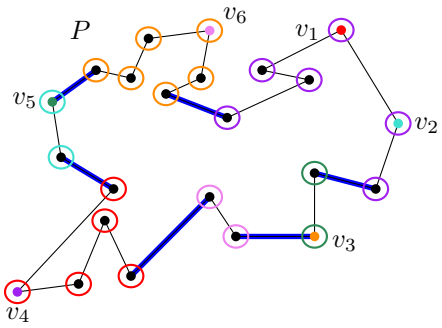

Figure 7.1: Each vertex of the boundary of P is associated with a farthest neighbor which is then marked. The boundary is then decomposed into vertex-disjoint chains, each associated with a marked vertex, joined by transition edges (blue) whose endpoints have different farthest neighbors.

7.2 Decomposing the boundary

In this section, we decompose the boundary of P into chains of consecutive vertices that share the same farthest neighbor and edges of P whose endpoints have distinct farthest neighbors.

Using a result from Hershberger and Suri [55], in $O(n)$ time we can compute the farthest neighbor of each vertex of P. Recall that the farthest neighbor of each vertex of P is always a convex vertex of P [9] and is unique by our general position assumption. The (geodesic farthest-point) *Voronoi region* of a vertex v of P is the set of points $R(v) = \{x \in P : F_P(x) = |\pi(x, v)|\}$ (including boundary points).

We mark the vertices of P that are farthest neighbors of at least one vertex of P. Let M denote the set of marked vertices of P (clearly this set can be computed in $O(n)$ time after applying the result of Hershberger and Suri). In other words, M contains all vertices of P whose Voronoi region contains at least one vertex of P.

Given a vertex v of P, the vertices of P whose farthest neighbor is v appear contiguously along ∂P [7]. Therefore, after computing all these farthest neighbors, we effectively split the boundary into subchains, each associated with a different vertex of M; see Figure 7.1.

Given two points x and y on ∂P, let $\partial P(x, y)$ be the polygonal chain that starts at x and follows the boundary of P clockwise until reaching y. We say that three (non-empty) disjoint sets A, B and C contained in ∂P are in *clockwise order* if $B \subset \partial P(a, c)$

for any $a \in A$ and any $c \in C$. (To ease notation, we say that three points $x, y, z \in \partial P$ are in clockwise order if $\{x\}, \{y\}$ and $\{z\}$ are in clockwise order).

Let a and b be the endpoints of an edge of ∂P such that b is the clockwise neighbor of a along ∂P and $f(a) \neq f(b)$. Recall that we have computed $f(a)$ and $f(b)$ in the previous step and note that $a, b, f(a), f(b)$ are in clockwise order. For any vertex $v \in \partial P$ such that $f(a), v, f(b)$ are in clockwise order, we know that there cannot be a vertex u of P such that $f(u) = v$. As proved by Aronov et al. [7, Corollary 2.7.4], if there is a point x on ∂P whose farthest neighbor is v, then x must lie on the open segment (a, b). In other words, the Voronoi region $R(v)$ restricted to ∂P is contained in (a, b).

7.3 Hourglasses

For any polygonal chain $C = \partial P(p_0, p_k)$, the *hourglass* of C, denoted by H_C, is the simple polygon contained in P bounded by C, $\pi(p_k, f(p_0))$, $\partial P(f(p_0), f(p_k))$ and $\pi(f(p_k), p_0)$; see Figure 7.2. We call C and $\partial P(f(p_0), f(p_k))$ the *top* and *bottom* chains of H_C, respectively, while $\pi(p_k, f(p_0))$ and $\pi(f(p_k), p_0)$ are referred to as the *walls* of H_C.

We say that the hourglass H_C is *open* if its walls are vertex-disjoint. We say that C is a *transition chain* if $f(p_0) \neq f(p_k)$ and neither $f(p_0)$ nor $f(p_k)$ are interior vertices of C. In particular, if an edge ab of ∂P is a transition chain, we say that it is a *transition edge* (see Figure 7.2).

Lemma 7.3.1. *[Restatement of Lemma 3.1.3 of [7]] If C is a transition chain of ∂P, then the hourglass H_C is an open hourglass.*

Note that by Lemma 7.3.1, the hourglass of each transition chain is open. In the remainder of this chapter, all the hourglasses considered are defined by a transition chain. That is, they are open and their top and bottom chains are edge-disjoint.

The following lemma is depicted in Figure 7.2 and is a direct consequence of the Ordering Lemma proved by Aronov et al. [7, Corollary 2.7.4].

Lemma 7.3.2. *Let C_1, C_2, C_3 be three edge-disjoint transition chains of ∂P in clockwise order. Then, the bottom chains of H_{C_1}, H_{C_2} and H_{C_3} are also edge-disjoint and*

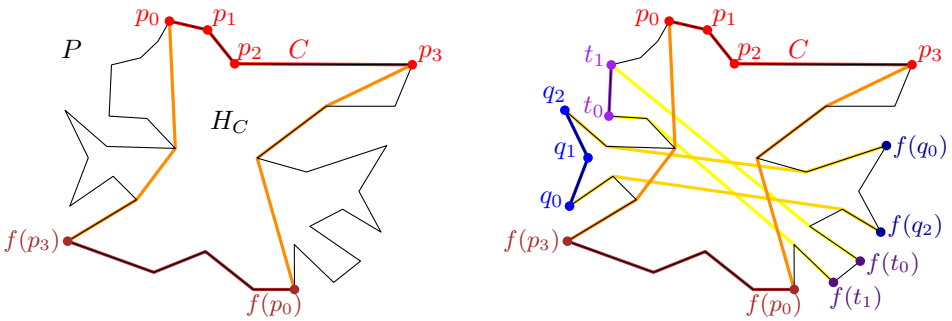

Figure 7.2: Given two edge-disjoint transition chains, their hourglasses are open and the bottom chains of their hourglasses are also edge-disjoint. Moreover, these bottom chains appear in the same cyclic order as the top chains along ∂P.

are in clockwise order.

Let γ be a geodesic path joining two points on the boundary of P. We say that γ *separates* two points x_1 and x_2 of ∂P if the points of $X = \{x_1, x_2\}$ and the endpoints of γ alternate along the boundary of P. We also say that γ separates x_1 and x_2 if either of them coincides with an endpoint of γ. We say that a geodesic path γ *separates* an hourglass H if it separates the points of its top chain from those of its bottom chain.

Lemma 7.3.3. *Let $C_1, \ldots, C_r$ be edge-disjoint transition chains of ∂P. Then, there is a set of $t \leq 10$ geodesic paths $\gamma_1, \ldots, \gamma_t$ with endpoints on ∂P such that for each $1 \leq i \leq r$ there exists $1 \leq j \leq t$ such that γ_j separates H_{C_i}. Moreover, this set can be computed in $O(n)$ time.*

Proof. Aronov et al. showed that there exist four vertices $v_1, \ldots, v_4$ of P and geodesic paths $\pi(v_1, v_2), \pi(v_2, v_3), \pi(v_3, v_4)$ such that for any point $x \in \partial P$, one of these paths separates x from $f(x)$ [7, Lemma 2.7.6]. Moreover, they show how to compute this set in $O(n)$ time.

Let $\Gamma = \{\pi(v_i, v_j) : 1 \leq i < j \leq 4\}$ and note that $v_1, \ldots, v_4$ split the boundary of P into at most four connected components. If a chain C_i is completely contained in one of these components, then one path of Γ separates the top and bottom chain of H_{C_i}. Otherwise, some vertex v_j is an interior vertex of C_i. However, because the chains $C_1, \ldots, C_r$ are edge-disjoint, there are at most four chains in this situation. For each chain C_i containing a vertex v_j, we add the geodesic path connecting the

endpoints of C_i to Γ. Therefore, Γ consists of at most 10 geodesic paths and each hourglass H_{C_i} has its top and bottom chain separated by some path of Γ. Since only $O(1)$ additional paths are computed, this can be done in linear time. □

Lemma 7.3.4. *[Restatement of Lemma 3.4.3 of [7]] Let $C_1, \ldots, C_r$ be a set of edge-disjoint transition chains of ∂P in clockwise order. Then each chord of P appears in $O(1)$ hourglasses among $H_{C_1}, \ldots, H_{C_r}$.*

Proof. Note that chords can appear only on walls of hourglasses. Because hourglasses are open, any chord must be an edge on exactly one wall of each of these hourglasses. Assume, for the sake of contradiction, that there exist two points $s, t \in P$ whose chord st is in three hourglasses H_{C_i}, H_{C_j} and H_{C_k} (for some $1 \leq i < j < k \leq r$) such that s is visited before t when going from the top chain to the bottom one along the walls of the three hourglasses. Let s_i and t_i be the points in the in the top and bottom chains of H_{C_i}, respectively, such that $\pi(s_i, t_i)$ is the wall of H_{C_i} that contains st (analogously, we define s_k and t_k).

Because C_i, C_j, C_k are in clockwise order, Lemma 7.3.2 implies that the bottom chains of C_i, C_j and C_k are also in clockwise order. Therefore, C_j lies between s_i and s_k and the bottom chain of H_{C_j} lies between t_i and t_k. That is, for each $x \in C_j$ and each y in the bottom chain of H_{C_j}, the geodesic path $\pi(x, y)$ is "sandwiched" between the paths $\pi(s_i, t_i)$ and $\pi(s_k, t_k)$. In particular, $\pi(x, y)$ contains st for each pair of points in the top and bottom chains of H_{C_j}. However, this implies that the hourglass H_{C_j} is not open—a contradiction that comes from assuming that st lies in the wall of three open hourglasses, when this wall is traversed from the top chain to the bottom chain. Analogous arguments can be used to bound the total number of walls that contain the edge st (when traversed in any direction) to $O(1)$. □

Given two points x and y in P, let $\#[x, y]$ denote the number of edges in the path $\pi(x, y)$.

Lemma 7.3.5. *Let x, u, y, v be four vertices of P in clockwise order. Given the shortest-path trees T_x and T_y of x and y in P, respectively, such that T_x and T_y can answer lowest common ancestor (LCA) queries in $O(1)$ time, we can compute the*

path $\pi(u, v)$ in $O(\#[u, v])$ time. Moreover, all edges of $\pi(u, v)$, except perhaps one, belong to $T_x \cup T_y$.

Proof. Let X (resp. Y) be the set containing the LCA in T_x (resp. T_y) of u, y, and of v, y (resp. u, x and x, y). Note that the points of $X \cup Y$ lie on the path $\pi(x, y)$ and can be computed in $O(1)$ time by hypothesis; see Figure 7.3. Moreover, using LCA queries, we can decide their order along the path $\pi(x, y)$ when traversing it from x to y. (Both X and Y could consist of a single vertex in some degenerate situations.) Two cases arise, either (1) there is a vertex $x^* \in X$ lying after a vertex $y^* \in Y$ along $\pi(x, y)$ or (2), all vertices of X^* lie before those of Y^* along $\pi(x, y)$.

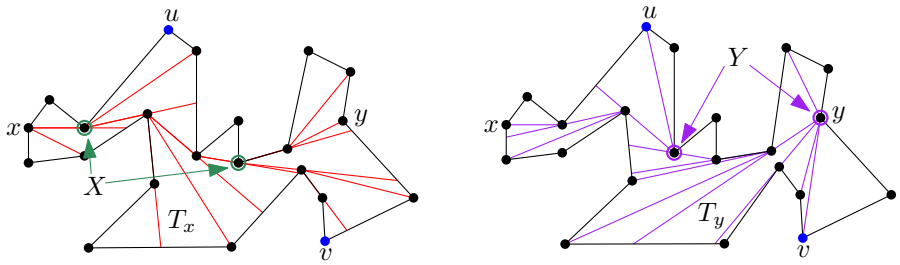

Figure 7.3: The shortest-path trees T_x and T_y are depicted as well as the set X and Y as defined in the proof of Lemma 7.3.5.

Case 1. If there is a vertex $x^* \in X$ lying after a vertex $y^* \in Y$ along $\pi(x, y)$, then the path $\pi(u, v)$ contains the path $\pi(y^*, x^*)$; see Figure 7.4 (top). Since $x^* \in \pi(u, v)$, $\pi(u, v)$ can be partitioned into two paths $\pi(u, x^*)$ and $\pi(x^*, v)$. Note that y^* lies either on $\pi(u, x^*)$ or on $\pi(x^*, v)$. Assume without loss of generality that y^* lies on $\pi(u, x^*)$; otherwise swap the roles of u and v.

Because y^* lies on $\pi(u, x^*)$, we conclude that $\pi(u, v)$ is the concatenation of the paths $\pi(u, y^*)$, $\pi(y^*, x^*)$ and $\pi(x^*, v)$. Furthermore, these three paths are contained in $T_x \cup T_y$ as $\pi(u, x^*) \subseteq \pi(y, u)$, $\pi(v, y^*) \subseteq \pi(x, v)$, and $\pi(x^*, y^*) \subseteq \pi(x, y)$. Therefore, $\pi(u, v)$ can be computed in time proportional to the number of its edges by traversing the corresponding shortest-path trees.

Case 2. In this case the vertices of X appear before the vertices of Y along $\pi(x, y)$. Let x' (resp. y') be the vertex of X (resp. Y) closest to x (resp. y). Notice that x' is in fact the LCA in T_x of u and v as u, y and v are in clockwise order. Analogously, y' is the LCA in T_y of u and v.

Let u' be the last vertex of $\pi(u,x)$ that is also in $\pi(u,y)$ (maybe u itself). Therefore, $\pi(u',x)$ and $\pi(u',y)$ coincide only at u'. Note that u' can be constructed by walking from u' towards x until the path towards y diverges. Thus, u' can be computed in $O(\#[u,u'])$ time. Define v' analogously and compute it in $O(\#[v,v'])$ time.

Because $u' \in \pi(u,x) \cap \pi(u,v)$ by definition and since y, v and x are in clockwise order, the path $\pi(u,v)$ contains u'. Analogously, we know the path $\pi(u,v)$ contains v'. Therefore, the path $\pi(u,v)$ is the union of $\pi(u,u'), \pi(u',v')$ and $\pi(v',v)$. Because $\pi(u,u')$ and $\pi(v',v)$ can be computed in time proportional to the number of their edges, it suffices to compute $\pi(u',v')$ in $O(\#[u',v'])$ time.

Let P' be the polygon bounded by the geodesic paths $\pi(x',u'), \pi(u',y'), \pi(y',v')$ and $\pi(v',x')$. By the definitions of x', y', u' and v', and since the vertices of X appear before the vertices of Y along $\pi(x,y)$, we know the four paths bounding P' are pairwise interior-disjoint, i.e., P' is a simple polygon; see Figure 7.4 (bottom). We claim that P' is a pseudo-quadrilateral with only four convex vertices being x', u', y' and v'. To show this, recall that each path bounding P' is a shortest path. Thus, the path $\pi(x',u')$ contains only reflex vertices of P'; otherwise, the shortest path from x' to u' restricted to P' would be shorter than $\pi(x',u')$ which is a contradiction as $P' \subseteq P$. Thus, each interior vertex of the path $\pi(x',u')$ is a reflex vertex of P'. The same argument holds for each of the other three paths bounding P'. Therefore, P' is a simple pseudo-quadrilateral with x', u', y' and v' as its four convex vertices.

Consequently, the shortest path from u' to v' can have at most one diagonal edge connecting distinct reflex chains of P'. Since the rest of the points in $\pi(u',v')$ lie on the boundary of P' and because each edge of P' is an edge of $T_x \cup T_y$, we conclude all edges of $\pi(u,v)$, except perhaps one, belong to $T_x \cup T_y$. In fact, this was already shown by Suri [88, Lemma 4].

We want to find the common tangent from either $\pi(u',x')$ or $\pi(u',y')$ to either $\pi(v',x')$ or $\pi(v',y')$ that belongs to the shortest path $\pi(u',v')$. Assume that the desired tangent lies between the paths $\pi(u',x')$ and $\pi(v',y')$, we consider the other three possible cases later. Since these paths consist only of reflex vertices, the problem can be reduced to finding the common tangent of two convex polygons. By slightly modifying the linear time algorithm to compute this tangent, we can make it run in

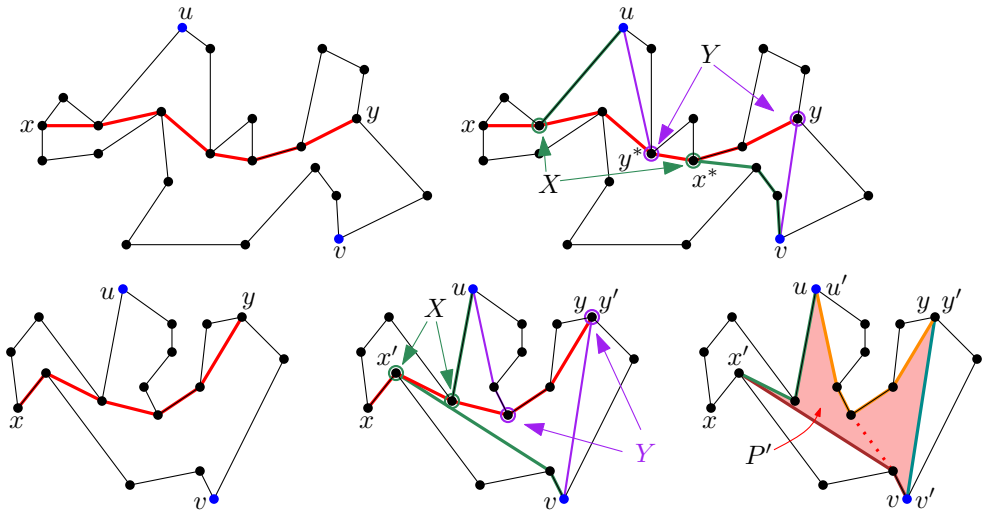

Figure 7.4: (top) Case 1 of the proof of Lemma 7.3.5 where the path $\pi(u, v)$ contains a portion of the path $\pi(x, y)$. (bottom) Case 2 of the proof of Lemma 7.3.5 where the path $\pi(u, v)$ has exactly one edge being the tangent of the paths $\pi(u', y')$ and $\pi(v', x')$.

$O(\#[u', v'])$ time.

Since we do not know if the tangent lies between the paths $\pi(u', x')$ and $\pi(v', y')$, we process the four pairs of paths in parallel and stop when finding the desired tangent. Therefore, we can compute $\pi(u', v')$ in $O(\#[u', v'])$ time and consequently, we can compute $\pi(u, v)$ in $O(\#[u, v])$ time. □

Lemma 7.3.6. *Let P be a simple polygon with n vertices. Given k disjoint transition chains $C_1, \ldots, C_k$ of ∂P, it holds that*

$$\sum_{i=1}^{k} |H_{C_i}| = O(n).$$

Proof. Because the given transition chains are disjoint, Lemma 7.3.2 implies that the bottom chains of their respective hourglasses are also disjoint. Therefore, the sum of the complexities of all the top and bottom chains of these hourglasses is $O(n)$. To bound the complexity of their walls we use Lemma 7.3.4. Since no chord is used more than a constant number of times, it suffices to show that the total number of chords used by all these hourglasses is $O(n)$.

To prove this, we use Lemma 7.3.3 to construct $O(1)$ paths $\gamma_1, \ldots, \gamma_t$ such that for each $1 \leq i \leq k$, there is a path γ_j that separates the top and bottom chains of

H_{C_i}. For each $1 \leq j \leq t$, let

$$\mathcal{H}^j = \{H_{C_i} : \text{the top and bottom chain of } H_{C_i} \text{ are separated by } \gamma_j\}.$$

Since the complexity of the shortest-path trees of the endpoints of γ_j is $O(n)$ [51], and from the fact that the chains $C_1, \ldots, C_k$ are disjoint, Lemma 7.3.5 implies that the total number of edges in all the hourglasses of $\mathcal{H}^j$ is $O(n)$. Moreover, because each of these edges appears in $O(1)$ hourglasses among $C_1, \ldots, C_k$, we conclude that

$$\sum_{H \in \mathcal{H}^j} |H| = O(n).$$

Since we have only $O(1)$ separating paths, our result follows. □

7.3.1 Building hourglasses

Let E be the set of transition edges of ∂P. Given a transition edge $ab \in E$, we say that H_{ab} is a *transition hourglass*. In order to construct the triangle cover of P, we construct the transition hourglass of each transition edge of E. By Lemma 7.3.6, we know that $\sum_{ab \in E} |H_{ab}| = O(n)$. Therefore, our aim is to compute the cover in time proportional to the size of H_{ab}.

By Lemma 7.3.3 we can compute a set Γ of $O(1)$ paths such that for any transition edge ab, the transition hourglass H_{ab} is separated by one (or more) paths in this set. For each endpoint of the $O(1)$ paths of Γ, we compute its shortest-path tree in linear time [28, 51]. In addition, we preprocess these trees in linear time to support constant time LCA queries [54]. Both computations need linear time per endpoint and use $O(n)$ space. Since we perform this process for a constant number of endpoints, overall this preprocessing takes $O(n)$ time.

Let $\gamma \in \Gamma$ be a separating path. Note that γ separates the boundary of P into two chains S_γ and S'_γ such that $S_\gamma \cup S'_\gamma = \partial P$. Let $\mathcal{H}(\gamma)$ be the set of transition hourglasses separated by γ whose transition edge is contained in S_γ (whenever an hourglass is separated by more than one path, we pick one arbitrarily). Note that we can classify all transition hourglasses into the sets $\mathcal{H}(\gamma)$ in $O(n)$ time (since $|\Gamma| = O(1)$).

Lemma 7.3.7. *Given a separating path $\gamma \in \Gamma$, it holds that $\sum_{H \in \mathcal{H}(\gamma)} |H| = O(n)$. Moreover, we can compute all transition hourglasses of $\mathcal{H}(\gamma)$ in $O(n)$ time.*

Proof. Since all transition hourglasses in $\mathcal{H}$ are defined from disjoint transition edges, Lemma 7.3.6 implies that $\sum_{H\in\mathcal{H}(\gamma)} |H| = O(n)$.

By construction, each wall of these hourglasses consists of a (geodesic) path that connects a point in S_γ with a point in S'_γ. Let $u \in S_\gamma$ and $v \in S'_\gamma$ be two vertices such that $\pi(u, v)$ is the wall of a hourglass in $\mathcal{H}(\gamma)$. Because LCA queries can be answered in $O(1)$ time, Lemma 7.3.5 allows us to compute this path in $O(\#[u, v])$ time. Therefore, we can compute all hourglasses of $\mathcal{H}(\gamma)$ in $O(\sum_{H\in\mathcal{H}(\gamma)} |H| + n) = O(n)$ time. □

Because $|\Gamma| = O(1)$ and since every transition hourglass is separated by at least one path in Γ, we obtain the following result.

Corollary 7.3.8. *The total complexity of the transition hourglasses of all transition edges of P is $O(n)$. Moreover, all these hourglasses can be constructed in $O(n)$ time.*

7.4 Funnels

Let $C = (p_0, \ldots, p_k)$ be a chain of ∂P and let v be a vertex of P not in C. The *funnel* of v to C, denoted by $S_v(C)$, is the simple polygon bounded by C, $\pi(p_k, v)$ and $\pi(v, p_0)$; see Figure 7.5 (a). Note that the paths $\pi(v, p_k)$ and $\pi(v, p_0)$ may coincide for a while before splitting into disjoint chains. We call C the *main chain* of $S_v(C)$ while $\pi(p_k, v)$ and $\pi(v, p_0)$ are referred to as the *walls* of the funnel. See Lee and Preparata [64] or Guibas et al. [51] for more details on funnels.

A subset $R \subset P$ is *geodesically convex* if for every $x, y \in R$, the path $\pi(x, y)$ is contained in R. This funnel $S_v(C)$ is then the minimum geodesically convex set that contains v and C.

Given two points $x, y \in P$, the (geodesic) *bisector* of x and y is the set of points contained in P that are equidistant from x and y. This bisector is a curve, contained in P, that consists of straight-line segments and hyperbolic arcs. Moreover, this curve intersects ∂P only at its endpoints [6, Lemma 3.22].

Lemma 7.4.1. *Let v be a vertex of P and let C be a transition chain such that $R(v) \cap \partial P \subset C$ and $v \notin C$. Then $R(v)$ is contained in the funnel $S_v(C)$.*

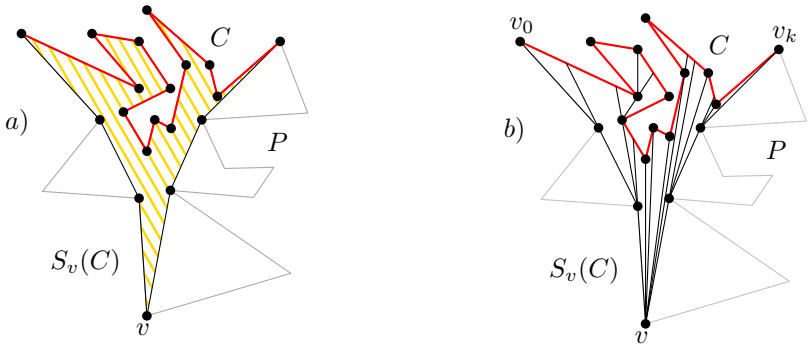

Figure 7.5: *a*) The funnel $S_v(C)$ of a vertex v and a chain C contained in ∂P are depicted. *b*) The decomposition of $S_v(C)$ into apexed triangles produced by the shortest-path map of v.

Proof. Let a and b be the endpoints of C such that $a, b, f(a)$ and $f(b)$ are in clockwise order. Because $R(v) \cap \partial P \subset C$, we know that $f(a), v$ and $f(b)$ are in clockwise order by Lemma 7.3.2. Moreover, we also know that $|\pi(a, v)| < |\pi(a, f(a))|$ and $|\pi(b, v)| < |\pi(b, f(b))|$ as $a, b \notin R(v)$.

Assume for a contradiction that there is a point of $R(v)$ lying outside of $S_v(C)$. Since $R(v) \cap \partial P \subset C$ and by continuity, the boundaries of $R(v)$ and $S_v(C)$ must intersect at a point w not in ∂P. Recall that the boundary of $S_v(C)$ is exactly the union of C and the paths $\pi(v, a)$ and $\pi(v, b)$. Therefore, w lies in either $\pi(v, a)$ or $\pi(v, b)$. Assume without loss of generality that $w \in \pi(v, b)$, the case where w lies in $\pi(v, a)$ is analogous. Because $w \in \pi(v, b)$, we know that $|\pi(b, w)|+|\pi(w, v)| = |\pi(b, v)|$. Moreover, since $w \in R(v)$, we know that $|\pi(w, f(b))| \leq |\pi(w, v)|$.

By the triangle inequality and since $w \notin \partial P$, we get that

$$|\pi(b, f(b))| \leq |\pi(b, w)| + |\pi(w, f(b))| \leq |\pi(b, w)| + |\pi(w, v)| = |\pi(b, v)|.$$

However, we knew that $|\pi(b, v)| < |\pi(b, f(b))|$—contradicting the assumption that $R(v)$ is not contained in $S_v(C)$. □

7.4.1 Funnels of marked vertices

Recall that for each marked vertex $v \in M$, we know at least of one vertex on ∂P such that v is its farthest neighbor.

For any marked vertex v, let $u_1, \ldots, u_{k-1}$ be the vertices of P such that $v = f(u_i)$ and assume that $u_1, \ldots, u_{k-1}$ are in clockwise order. Let u_0 and u_k be the neighbors of u_1 and u_{k-1} other than u_2 and u_{k-2}, respectively. Note that both u_0u_1 and $u_{k-1}u_k$ are transition edges of P. Let $C_v = (u_0, \ldots, u_k)$ and consider the funnel $S_v(C_v)$ defined by v.

Lemma 7.4.2. *Let x be a point in P. If $f(x) = v$ for some marked vertex $v \in M$, then $x \in S_v(C_v)$.*

Proof. Since $f(u_0) \neq f(u_k)$, C_v is a transition chain. Moreover, C_v contains $R(v) \cap \partial P$ by definition. Therefore, Lemma 7.4.1 implies that $R(v) \subset S_v(C_v)$. Since $v = f(x)$, we know that $x \in R(v)$ and hence that $x \in S_v(C_v)$. □

We focus now on computing the funnels defined by the marked vertices of P.

Lemma 7.4.3. *Given a marked vertex v such that $C_v = (u_0, \ldots, u_k)$, it holds that $|S_v(C_v)| = O(k + |H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$. Moreover, $S_v(C_v)$ can be computed in $O(|S_v(C_v)|)$ time.*

Proof. Since $H_{u_0u_1}$ and $H_{u_{k-1}u_k}$ are transition hourglasses, we can assume that they have been computed using Corollary 7.3.8.

Because $v = f(u_1) = f(u_{k-1})$, we know that v is a vertex of both $H_{u_0u_1}$ and $H_{u_{k-1}u_k}$. By definition, we have $\pi(v, u_0) \subset H_{u_0u_1}$ and $\pi(v, u_k) \subset H_{u_{k-1}u_k}$. Thus, we can compute both paths $\pi(v, u_0)$ and $\pi(v, u_k)$ in $O(|H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$ time [51] and their complexities are $O(|H_{u_0u_1}|)$ and $O(|H_{u_{k-1}u_k}|)$, respectively. So, overall, the funnel $S_v(C_v)$ has total complexity $O(k + |H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$ and can be constructed in linear time in its size. □

Recall that, by Corollary 7.3.8, the total sum of the complexities of all transition hourglasses is $O(n)$. Therefore, Lemma 7.4.3 implies that

$$\sum_{v \in M} |S_v(C_v)| = O\left(n + \sum_{ab \in E} |H_{ab}|\right) = O(n).$$

Because the funnel of each marked vertex can be computed in linear time in its size, we obtain the following result.

Corollary 7.4.4. *The total complexity of the funnels of all marked vertices of P is $O(n)$. Moreover, all these funnels can be constructed in $O(n)$ time.*

7.5 Covering the polygon with apexed triangles

An *apexed triangle* $\triangle = (a, b, c)$ with *apex* a is a triangle contained in P with an associated distance function $g_\triangle(x)$, called the *apex function* of $\triangle$, such that (1) a is a vertex of P, (2) there is an edge of ∂P containing both b and c, and (3) there is a vertex w of P, called the *definer* of $\triangle$, such that

$$g_\triangle(x) = \begin{cases} -\infty & \text{if } x \notin \triangle, \\ |xa| + |\pi(a, w)| = |\pi(x, w)| & \text{if } x \in \triangle. \end{cases}$$

Intuitively, $\triangle$ bounds a constant complexity region where the geodesic distance function from w is explicitly stored by our algorithm.

In this section, we show how to find a set of $O(n)$ apexed triangles of P such that the upper envelope of their apex functions coincides with $F_P(x)$. To this end, we first decompose the transition hourglasses into apexed triangles that encode all the geodesic distance information inside them. For each marked vertex $v \in M$ we construct a funnel that contains the Voronoi region of v. We then decompose this funnel into apexed triangles that encode the distance from v.

The same approach was already used by Pollack et al. in [83, Section 3]. Given a segment contained in the interior of P, they show how to compute a linear number of apexed triangles such that $F_P(x)$ coincides with the upper envelope of the corresponding apex functions in the given segment.

While the construction we follow is analogous, we use it in each transition hourglass H_{ab} instead of the full polygon P. Therefore, we have to specify what is the relation between the upper envelope of the computed functions and $F_P(x)$. We will show that the upper envelope of the apex functions computed in H_{ab} coincides with $F_P(x)$ inside the Voronoi region $R(v)$ of every vertex v in the bottom chain of H_{ab}.

7.5.1 Inside a transition hourglass

Let ab be a transition edge of P such that b is the clockwise neighbor of a along ∂P. Let B_{ab} denote the open bottom chain of H_{ab}. As noticed above, a point on ∂P can be farthest from a vertex in B_{ab} only if it lies in the open segment ab. That is, if v is a vertex of B_{ab} such that $R(v) \neq \emptyset$, then $R(v) \cap \partial P \subset ab$.

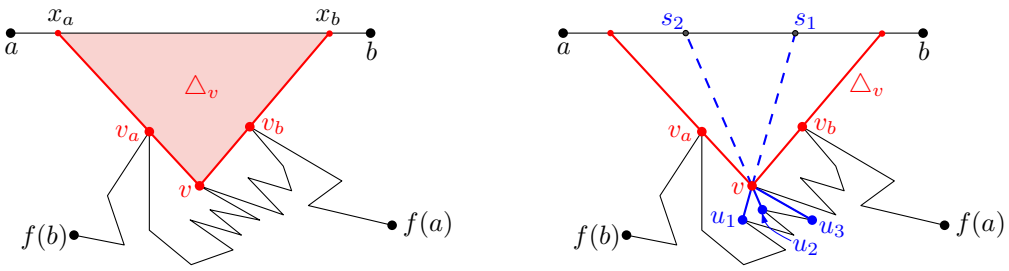

Figure 7.6: (left) A vertex v visible from the segment ab lying on the bottom chain of H_{ab}, and the triangle $\triangle_v$ that contains the portion of ab visible from v. (right) The children u_1 and u_2 of v are visible from ab while u_3 is not. The triangle $\triangle_v$ is split into apexed triangles by the rays going from u_1 and u_2 to v.

In fact, not only this Voronoi region is inside H_{ab} when restricted to the boundary of P, but we can further bound its location and show that $R(v) \subset H_{ab}$. The next result follows trivially from Lemma 7.4.1.

Corollary 7.5.1. *Let v be a vertex of B_{ab}. Then $R(v) \subset H_{ab}$.*

Our objective is to compute $O(|H_{ab}|)$ apexed triangles contained in H_{ab}, each with its distance function, such that the upper envelope of these apex functions coincides with $F_P(x)$ restricted to H_{ab} inside the Voronoi region of every vertex in B_{ab}.

Let T_a and T_b be the shortest-path trees in H_{ab} rooted at a and b, respectively. We can compute these trees in $O(|H_{ab}|)$ time [51]. For each vertex v such that $f(a), v$ and $f(b)$ are in clockwise order, let v_a and v_b be the neighbors of v in the paths $\pi(v, a)$ and $\pi(v, b)$, respectively. We say that a vertex v is *visible* from ab if $v_a \neq v_b$. Note that if $v_a \neq a$, then both v_a and v_b lie inside the triangle $\triangle(a, b, v)$. Thus, if vertex v is visible, then the extension of the segments vv_a and vv_b must intersect the top segment ab at points x_a and x_b, respectively. Therefore, for each visible vertex v, we obtain a triangle $\triangle_v = \triangle(v, x_a, x_b)$ as shown in Figure 7.6.

We further split $\triangle_v$ into a series of triangles with apex at v as follows: Let u be a child of v in either T_a or T_b. As noted by Pollack et al. [83], v can be of three types, either (1) u is not visible from ab (and is hence a child of v in both T_a and T_b); or (2) u is visible from ab, is a child of v only in T_b, and v_bvu is a left turn; or (3) u is visible from ab, is a child of v only in T_a, and v_avu is a right turn.

Let $u_1, \ldots, u_{k-1}$ be the children of v of type (2) sorted in clockwise order around v. Let $c(v)$ be the maximum distance from v to any invisible vertex in the subtrees of

T_a and T_b rooted at v; if no such vertex exists, then $c(v) = 0$. Define a function $d_l(v)$ on each vertex v of H_{ab} in a recursive fashion as follows: If v is invisible from ab, then $d_l(v) = c(v)$. Otherwise, let $d_l(v)$ be the maximum of $c(v)$ and $\max\{d_l(u_i) + |u_i v| : u_i$ is a child of v of type (2)$\}$. Symmetrically, we define a function $d_r(v)$ using the children of type (3) of v.

For each $1 \leq i \leq k-1$, extend the segment $u_i v$ past v until it intersects ab at a point s_i. Let s_0 and s_k be the intersections of the extensions of vv_a and vv_b with the segment ab. We define then k apexed triangles contained in $\triangle_v$ as follows. For each $0 \leq i \leq k-1$, consider the triangle $\triangle(s_i, v, s_{i+1})$ whose associated apexed (left) function is

$$f_i(x) = \begin{cases} |xv| + \max_{j>i}\{c(v), |vu_j| + d_l(u_j)\}, & \text{if } x \in \triangle(s_i, v, s_{i+1}), \\ -\infty, & \text{otherwise.} \end{cases}$$

In a symmetric manner, we define a set of apexed triangles induced by the type (3) children of v and their respective apexed (right) functions.

Let $g_1, \ldots, g_r$ and $\triangle_1, \ldots, \triangle_r$ respectively be an enumeration of all the generated apex functions and apexed triangles such that g_i is defined in the triangle $\triangle_i$. Because each function is determined uniquely by a pair of adjacent vertices in T_a or in T_b, and since these trees have $O(|H_{ab}|)$ vertices each, we conclude that $r = O(|H_{ab}|)$.

Note that for each $1 \leq i \leq r$, the apexed triangle $\triangle_i$ has two vertices on the segment ab and a third vertex, say a_i, being its apex such that for each $x \in \triangle_i$, $g_i(x) = |\pi(x, w_i)|$ for some vertex w_i of H_{ab}. Recall that w_i is called the definer of $\triangle_i$.

Lemma 7.5.2. *Given a transition edge ab of P, we can compute a set $\mathcal{A}_{ab}$ of $O(|H_{ab}|)$ apexed triangles in $O(|H_{ab}|)$ time with the property that for any point $p \in P$ such that $f(p) \in B_{ab}$, there is an apexed triangle $\triangle \in \mathcal{A}_{ab}$ with apex function g and definer equal to $f(p)$ such that*

1. *$p \in \triangle$ and*
2. *$g(p) = F_P(p)$.*

Proof. Let $p \in P$ such that $f(p) \in B_{ab}$ and let $\mathcal{A}_{ab}$ be the set of apexed triangles computed in H_{ab} by the above procedure. We claim that p lies inside an apexed

triangle of $\mathcal{A}_{ab}$. To show this, note that since $p \in R(f(p))$, Corollary 7.5.1 implies that $p \in H_{ab}$. Consider the path $\pi(p, f(p))$ and let v be the successor of p along this path. Because $p \in R(f(p))$, the ray shooting from v towards p stays within $R(f(p))$ until hitting the boundary of P as shown by Aronov et al. [7, Lemma 2.6.6]. Consequently, this ray must intersect the segment ab which implies that v is visible from ab. Therefore, p lies inside $\triangle_v$ and hence it is contained in one of the apex triangles defined by v during the construction of $\mathcal{A}_{ab}$. That is, there is an apexed triangle $\triangle \in \mathcal{A}_{ab}$ with apex v and definer $f(p)$ that contains p.

Since the apex function of $\triangle$ encodes the geodesic distance to a vertex of P and because $F_P(x)$ is the upper envelope of all the geodesic functions, we know that $g(p) \leq F_P(p)$.

To prove the other inequality, note that if $v = f(p)$, then trivially $g(p) = |pv| + |\pi(v, w)| \geq |pv| = |\pi(p, f(p))| = F_P(p)$. Otherwise, let z be the next vertex after v in the path $\pi(p, f(p))$. Three cases arise:

(a) If z is invisible from ab, then so is $f(p)$ and hence,

$$|\pi(p, f(p))| = |pv| + |\pi(v, f(p))| \leq |pv| + c(v) \leq g(p).$$

(b) If z is a child of type (2), then z plays the role of some child u_j of v in the notation used during the construction. In this case:

$$|\pi(p, f(p))| = |pv| + |vz| + |\pi(z, f(p))| \leq |pv| + |vu_j| + d_l(u_j) \leq g(p).$$

(c) If z is a child of type (3), then analogous arguments hold using the (right) distance d_r.

Therefore, regardless of the case $F_P(p) = |\pi(p, f(p))| \leq g(p)$.

To bound the running time, note that the recursive functions d_l, d_r and c can be computed in $O(|T_a| + |T_b|)$ time. Then, for each vertex visible from ab, we can process it in time proportional to its degree in T_a and T_b. Because the sum of the degrees of all vertices in T_a and T_b is $O(|T_a| + |T_b|)$ and from the fact that both $|T_a|$ and $|T_b|$ are $O(|H_{ab}|)$, we conclude that the total running time to construct $\mathcal{A}_{ab}$ is $O(|H_{ab}|)$. □

In other words, Lemma 7.5.2 says that no information on farthest neighbors is lost if within H_{ab} we consider only the functions of $\mathcal{A}_{ab}$. In the next section we use a

similar approach to construct a set of apexed triangles (and their corresponding apex functions) to encode the distance from the vertices of M.

7.5.2 Inside the funnels of marked vertices

We now proceed to split a given funnel into $O(|S_v(C_v)|)$ apexed triangles that encode the distance function from v. To this end, we use the algorithm described by Guibas et al. [51, Section 2] to compute the shortest-path map of v in $S_v(C_v)$ in $O(|S_v(C_v)|)$ time. This algorithm produces a partition of $S_v(C_v)$ into $O(|S_v(C_v)|)$ interior-disjoint triangles with vertices on ∂P, such that each triangle consists of all points in $S_v(C_v)$ whose shortest path to v consists of the same sequence of vertices; see Figure 7.5 (b). Let $\triangle$ be a triangle in this partition and let a be its apex, i.e., the first vertex found along each path $\pi(x, v)$, where $x \in \triangle$. We define the apex function $g_{\triangle}(x)$ of $\triangle$ as follows:

$$g_{\triangle}(x) = \begin{cases} |xa| + |\pi(a, v)| & \text{if } x \in \triangle, \\ -\infty & \text{otherwise.} \end{cases}$$

Notice that for each $x \in \triangle$, $g_{\triangle}(x) = |\pi(x, v)|$.

Lemma 7.5.3. *The shortest-path map of v in $S_v(C_v)$ can be computed in $O(|S_v(C_v)|)$ time and produces $O(|S_v(C_v)|)$ interior-disjoint apexed triangles such that their union covers $S_v(C_v)$. Moreover, for each point $x \in R(v)$, there is an apexed triangle $\triangle$ with apex function $g(x)$ such that (1) $x \in \triangle$ and (2) $g(x) = F_P(x)$.*

Proof. The above procedure splits $S_v(C_v)$ into $O(|S_v(C_v)|)$ apexed triangles, such that the apex function in each of them is defined as the geodesic distance to v. By Lemma 7.4.2, if $x \in R(v)$, then $x \in S_v(C_v)$. Therefore, there is an apexed triangle $\triangle$ with apex function $g(x)$ such that $x \in \triangle$ and $g(x) = |\pi(x, v)| = F_P(x)$. Thus, we obtain properties (1) and (2). □

7.6 Prune and search

With the tools introduced in the previous sections, we can describe the prune-and-search algorithm to compute the geodesic center. The idea of the algorithm is to

partition P into $O(1)$ cells, determine in which cell of P the center lies and recurse on that cell as a new subproblem with smaller complexity.

We can discard all apexed triangles that do not intersect the new cell containing the center. Using cuttings to produce this partition of P, we can show that both the complexity of the cell containing the center, and the number of apexed triangles that intersect it decrease by a constant fraction in each iteration of the algorithm. This process is then repeated until either of the two objects has constant descriptive size.

Let τ be the set of all apexed triangles computed in previous sections. Corollary 7.3.8 and Lemma 7.5.2 bound the number of apexed triangles constructed inside the transition hourglasses, while Corollary 7.4.4 and Lemma 7.5.3 do so inside the funnels of the marked vertices. We obtain the following.

Corollary 7.6.1. *The set τ consists of $O(n)$ apexed triangles.*

Let $\phi(x)$ be the upper envelope of the apex functions of the triangles in τ (i.e., $\phi(x) = \max\{g(x) : \triangle \in \tau$ and $g(x)$ is the apex function of $\triangle\}$). The following result is a direct consequence of Lemmas 7.5.2 and 7.5.3, and shows that the $O(n)$ apexed triangles of τ not only cover P, but their apex functions suffice to reconstruct the function $F_P(x)$.

Corollary 7.6.2. *The functions $\phi(x)$ and $F_P(x)$ coincide in the domain of points of P, i.e., for each $p \in P$, $\phi(p) = F_P(p)$.*

Given a chord C of P a *half-polygon* of P is one of the two simple polygons into which C splits P. A *k-cell* of P is a simple polygon obtained as the intersection of at most k half-polygons. Because a k-cell is the intersection of geodesically convex sets, it is also geodesically convex.

The recursive algorithm described in this section takes as input a 4-cell (initially equal to P) containing the geodesic center of P and the set of all apexed triangles of τ that intersect it. In each iteration, it produces a new 4-cell of smaller complexity that intersects just a fraction of the apexed triangles and contains the geodesic center of P. By recursing on this new cell, the complexity of the problem is reduced in each iteration by a constant fraction.

Let R be a 4-cell of P (initially equal to P) containing the geodesic center of P and let τ_R be the set of apexed triangles of τ that intersect R. Let $m_{\scriptscriptstyle R} = \max\{|R|, |\tau_R|\}$, where $|R|$ denotes the combinatorial complexity of R. Recall that, by construction of the apexed triangles, for each triangle of τ_R at least one and at most two of its boundary segments are chords of P. Let $\mathcal{C}_R$ be the set containing all chords that belong to the boundary of a triangle of τ_R. Therefore, $|\mathcal{C}_R| \leq 2|\tau_R| \leq 2m_{\scriptscriptstyle R}$.

To construct ε-nets, we need some definitions (for more information on ε-nets refer to [67]). Let φ be the set of all open 4-cells of P. For each $t \in \varphi$, let $\mathcal{C}_R(t) = \{C \in \mathcal{C}_R : C \cap t \neq \emptyset\}$ be the set of chords of $\mathcal{C}_R$ intersected by t. Finally, let $\varphi_{\mathcal{C}_R} = \{\mathcal{C}_R(t) : t \in \varphi\}$ be the family of subsets of $\mathcal{C}_R$ induced by φ. Consider the set system $(\mathcal{C}_R, \varphi_{\mathcal{C}_R})$ (denoted by $(\mathcal{C}_R, \varphi)$ for simplicity) defined by $\mathcal{C}_R$ and φ. The proof of the next lemma is deferred to Section 7.8 for ease of readability.

Lemma 7.6.3. *The set system $(\mathcal{C}_R, \varphi)$ has constant VC-dimension.*

Let $\varepsilon > 0$ be a sufficiently small constant whose exact value will be specified later. Because the VC-dimension of the set system $(\mathcal{C}_R, \varphi)$ is finite by Lemma 7.6.3, and Theorems 3.1.1 and 3.1.3, we can compute an ε-net N of $(\mathcal{C}_R, \varphi)$ in $O(|\mathcal{C}_R|/\varepsilon) = O(m_{\scriptscriptstyle R})$ time [67]. The size of N is $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}) = O(1)$ and its main property is that any 4-cell that does not intersect a chord of N will intersect at most $\varepsilon|\mathcal{C}_R|$ chords of $\mathcal{C}_R$.

Observe that N partitions R into $O(1)$ sub-polygons (not necessarily 4-cells). We further refine this partition to obtain 4-cells. That is, we shoot vertical rays up and down from each endpoint of N, and from the intersection point of any two segments of N, see Figure 7.7. Overall, this partitions R into $O(1)$ 4-cells such that each either (i) is a convex polygon contained in P of at most four vertices, or otherwise (ii) contains some chain of ∂P. Since $|N| = O(1)$, the whole decomposition can be computed in $O(m_{\scriptscriptstyle R})$ time (the intersections between segments of N are done in constant time, and for the ray shooting operations we walk along the boundary of R once).

In order to determine which 4-cell contains the geodesic center of P, we extend each edge of a 4-cell to a chord C. This can be done with two ray-shooting queries (each of which takes $O(m_{\scriptscriptstyle R})$ time). We then use the chord-oracle from Pollack et al. [83, Section 3] to decide which side of C contains $c_{\scriptscriptstyle P}$. The only requirement of this technique is that the function $F_{\scriptscriptstyle P}(x)$ coincides with the upper envelope of the apex

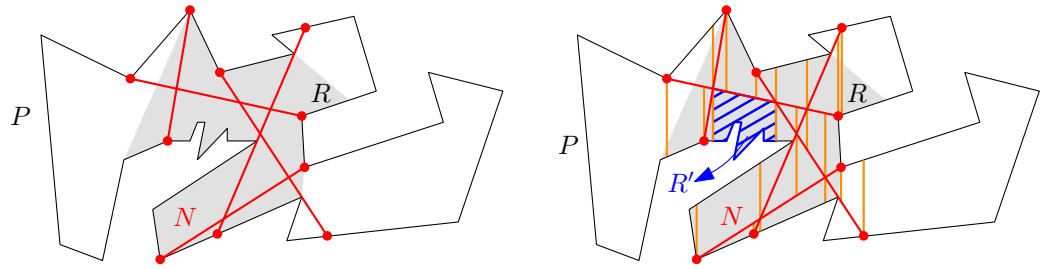

Figure 7.7: The ϵ-net N splits R into $O(1)$ sub-polygons that are further refined into a 4-cell decomposition using $O(1)$ ray-shooting queries from the vertices of the arrangement defined by N.

functions when restricted to C. This holds by Corollary 7.6.2 and from the fact that τ_R consists of all the apexed triangles of τ that intersect R.

Because the chord-oracle described by Pollack et al. [83, Section 3] runs in time linear in the number of functions defined on C, we can decide in total $O(m_R)$ time in which side of C the geodesic center of P lies. Since our decomposition into 4-cells has constant complexity, we need to perform $O(1)$ calls to the oracle before determining the 4-cell R' that contains the geodesic center of P.

Note that the chord-oracle computes the minimum of $F_P(x)$ restricted to the chord before determining the side containing the minimum. In particular, if c_P lies on any chord bounding R', then the chord-oracle will find it. Therefore, we can assume that c_P lies in the interior of R'. Moreover, since N is a ε-net, we know that at most $\varepsilon|\mathcal{C}_R|$ chords of $\mathcal{C}_R$ intersect R'.

Observe that both the complexity of R' and $\tau_{R'}$ decrease by a constant fraction. Indeed, by construction of the cutting at most $2\varepsilon m_R$ apexed triangles can intersect R' (and thus $|\tau_{R'}| \leq 2\varepsilon m_R$).

In order to bound the complexity of R' we use Corollary 7.6.2: function $F_P(x)$ is defined at each point of R', and in particular for each vertex v of R' there must exist an apexed triangle $\Delta \in \tau_{R'}$ such that $v \in \Delta$. By definition of apexed triangles, each such triangle can contain at most three vertices of R'. Combining this fact with the bound of $|\tau_{R'}|$ we obtain that R' has at most $3|\tau_{R'}| \leq 6\varepsilon m_R$ vertices. Thus, if we choose $\varepsilon = 1/12$, we guarantee that both the size of the 4-cell R' and the number of apexed triangles in $\tau_{R'}$ are at most $m_R/2$.

In order to proceed with the algorithm on R' recursively, we need to compute the

set $\tau_{R'}$ with the at most $\varepsilon|\mathcal{C}_R|$ apexed triangles of τ_R that intersect R' (i.e., prune the apexed triangles that do not intersect with R'). For each apexed triangle $\triangle \in \tau_R$, we can determine in constant time if it intersects R' (either one of the endpoints is in $R' \cap \partial P$ or the two boundaries have non-empty intersection in the interior of P). Overall, we need $O(m_R)$ time to compute the at most $\varepsilon|\mathcal{C}_R|$ triangles of τ_R that intersect R'.

By recursing on R', we guarantee that after $O(\log m_R)$ iterations, we reduce the size of either τ_R or R' to constant. In the former case, the minimum of $F_P(x)$ can be found by explicitly constructing function ϕ in $O(1)$ time. In the latter case, we triangulate R' and apply the chord-oracle to determine which triangle will contain c_P. The details needed to find the minimum of $\phi(x)$ inside this triangle are given in the next section.

Lemma 7.6.4. *In $O(n)$ time we can find either the geodesic center of P or a triangle containing the geodesic center.*

7.7 Finding the center within a triangle

In order to complete the algorithm it remains to show how to find the geodesic center of P for the case in which R' is a triangle inside P. If this triangle is in the interior of P, it may happen that several apexed triangles of τ (up to a linear number) fully contain R'. Thus, the pruning technique used in the previous section cannot be further applied. We solve this case with a different approach.

Recall that $\phi(x)$ denotes the upper envelope of the apex functions of the triangles in τ, and the geodesic center is the point that minimizes ϕ. The key observation is that, as with chords, the function $\phi(x)$ restricted to R' is convex.

Let $\tau_{R'} = \{\triangle_1, \triangle_2, \ldots, \triangle_m\}$ be the set of $m = O(n)$ apexed triangles of τ that intersect R'. Let a_i and w_i be the apex and the definer of $\triangle_i$, respectively. Let $g_i(x)$ be the apex function of $\triangle_i$ such that

$$g(x) = \begin{cases} |xa_i| + \kappa_i & \text{if } x \in \triangle_i \\ -\infty & \text{otherwise} \end{cases},$$

where $\kappa_i = |\pi(a_i, w_i)|$ is a constant.

By Corollary 7.6.2, we have $\phi(x) = F_P(x)$. Therefore, the problem of finding the center is equivalent to the following optimization problem in $\mathbb{R}^3$:

(P1). Find a point $(x, r) \in \mathbb{R}^3$ minimizing r subject to $x \in R'$ and

$$g_i(x) \leq r, \text{ for } 1 \leq i \leq m.$$

Thus, we need to find the solution to (P1) to find the geodesic center of P. We use some ideas due to Megiddo in order to simplify the description of (P1) [71].

To simplify the formulas, we square the equation $|xa_i| \leq r - \kappa_i$:

$$\|x\|^2 - 2x \cdot a_i + \|a_i\|^2 = |xa_i|^2 \leq (r - \kappa_i)^2 = r^2 - 2r\kappa_i + \kappa_i^2.$$

And finally for each $1 \leq i \leq m$, we define the function $h_i(x, r)$ as follows:

$$h_i(x, r) = \begin{cases} \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2 & \text{if } x \in \triangle_i \\ -\infty & \text{otherwise} \end{cases}.$$

Therefore, our optimization problem can be reformulated as:

(P2). Find a point $(x, r) \in \mathbb{R}^3$ such that r is minimized subject to $x \in R'$ and

$$h_i(x, r) \leq 0 \text{ and } r > \max\{\kappa_i\}, \text{ for each } i \text{ such that } x \in \triangle_i.$$

Let $h'_i(x, r) = \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2$ be a function defined in the entire plane and let (P2′) be an optimization problem equivalent to (P2) in which every instance of $h_i(x, r)$ is replaced by $h'_i(x, r)$. The optimization (P2′) was studied by Megiddo in [71]. We provide some of the intuition used by Megiddo to solve this problem.

Although the functions $h'_i(x, r)$ are not linear, they all have the same non-linear terms. Therefore, for $i \neq j$, we get that equation $h'_i(x, r) = h'_j(x, r)$ defines a *separating plane*

$$\gamma_{i,j} = \{(x, r) \in \mathbb{R}^3 : 2(\kappa_i - \kappa_j)r - 2(a_i - a_j) \cdot x + \|a_i\|^2 - \|a_j\|^2 - \kappa_i^2 + \kappa_j^2 = 0\}$$

As noted by Megiddo [71], this separating plane has the following property: If the solution (x, r) to (P2′) is known to lie to one side of $\gamma_{i,j}$, then one of the constraints is redundant.

Thus, to solve (P2′) it sufficed to have a *side-decision oracle* to determine in which side of a plane $\gamma_{i,j}$ the solution lies. Megiddo showed how to implement this oracle so that the running time is proportional to the number of constraints [71].

Once we have such an oracle, problem (P2′) can be solved using a prune-and-search approach: pair the functions arbitrarily, and consider the set of $m/2$ separating planes defined by these pairs. For some constant t, compute a $1/t$-cutting in $\mathbb{R}^3$ of the separating planes. A $1/t$-cutting is a partition of the space into $O(t^3) = O(1)$ convex regions each of which is of constant complexity and intersects at most $m/2t$ separating planes. A cutting of planes can be computed in linear time in $\mathbb{R}^3$ for any $t = O(1)$ [67]. After computing the cutting, determine in which of the regions the minimum lies by performing $O(1)$ calls to the side-decision oracle. Because at least $(t-1)m/2t$ separating planes do not intersect this constant complexity region, for each of the corresponding pairs of functions we can discard one of them as redundant. Thereby eliminating a constant fraction of the constraints. By repeating this algorithm recursively we obtain a linear running time.

To solve (P2) we follow a similar approach, but our set of separating planes needs to be extended in order to handle apex functions which are defined only in triangular domains. Note that no vertex of an apexed triangle can lie inside R'.

7.7.1 Optimization problem in a convex domain

In this section we describe our algorithm to solve the optimization problem (P2). We start by pairing the apexed triangles of $\tau_{R'}$ arbitrarily to obtain $m/2$ pairs. By identifying the plane where P lies with the plane $Z_0 = \{(x, y, z) : z = 0\}$, we can embed each apexed triangle in $\mathbb{R}^3$. A *plane-set* is a set consisting of at most five planes in $\mathbb{R}^3$. For each pair of apexed triangles $(\triangle_i, \triangle_j)$ we define its plane-set as follows: For each chord of P bounding either $\triangle_i$ or $\triangle_j$ (at most two chords on each triangle), consider the line extending this chord and the vertical extrusion of this line in $\mathbb{R}^3$, i.e., the plane containing this chord orthogonal to Z_0. The set containing these planes, together with the separating plane $\gamma_{i,j}$, is the plane-set of the pair $(\triangle_i, \triangle_j)$.

Let Γ be the union of all the plane-sets defined by the $m/2$ pairs of apexed triangles. Because the plane-set of each pair $(\triangle_i, \triangle_j)$ consists of at most five planes, at

least one of which is unique to this pair, namely $\gamma_{i,j}$, we infer that $m/2 \leq |\Gamma| \leq 5m/2$.

Using Theorem 3.1.4, compute a $1/t$-cutting of Γ in $O(m)$ time for some constant t to be specified later. Because t is constant, this $1/t$-cutting splits the space into $O(1)$ convex regions, each bounded by a constant number of planes [67]. Using a side-decision algorithm (to be specified later), we can determine the region Q of the cutting that contains the solution to (P2). Because Q is a region of the $1/t$-cutting of Γ, we know that at most $|\Gamma|/t$ planes of Γ intersect Q. In particular, at most $|\Gamma|/t$ plane-sets intersect Q and hence, at least $(t-1)|\Gamma|/t \geq (t-1)m/2t$ plane-sets do not intersect Q.

Let $(\triangle_i, \triangle_j)$ be a pair whose plane-set does not intersect Q. Let Q' be the projection of Q on the plane Z_0. Because the plane-set of this pair does not intersect Q, we know that Q' intersects neither the boundary of $\triangle_i$ nor that of $\triangle_j$. Two cases arise:

Case 1. If either $\triangle_i$ or $\triangle_j$ does not intersect Q', then we know that their apex function is redundant and we can drop the constraint associated with this apexed triangle.

Case 2. If $Q' \subset \triangle_i \cap \triangle_j$, then we need to decide which constraint to drop. To this end, we consider the separating plane $\gamma_{i,j}$. Notice that inside the vertical extrusion of $\triangle_i \cap \triangle_j$ (and hence in Q), the plane $\gamma_{i,j}$ has the property that if we know its side containing the solution to (P2), then one of the constraints can be dropped.

Regardless of the case, if the plane-set of a pair $(\triangle_i, \triangle_j)$ does not intersect Q, then we can drop one of its constraints. Since at least $(t-1)m/2t$ plane-sets do not intersect Q, we can drop at least $(t-1)m/2t$ constraints. By choosing $t = 2$, we are able to drop at least $(t-1)m/2t = m/4$ constraints. Consequently, after $O(m)$ time, we are able to drop a constant fraction of the apexed triangles. By repeating this process recursively, we end up with a constant size problem in which we can compute the upper envelope of the functions explicitly and find the solution to (P2) using exhaustive search. Thus, the running time of this algorithm is bounded by the recurrence $T(m) = T(3m/4) + O(m)$ which solves to $O(m)$. Because $m = O(n)$, we can find the solution to (P2) in $O(n)$ time.

The last detail is the implementation of the side-decision algorithm. Given a

plane γ, we need to determine on which side the solution to (P2) lies. To this end, we solve (P2) restricted to γ, i.e., with the additional constraint of $(x, r) \in \gamma$. This approach was used by Megiddo [71]; the idea is to recurse by reducing the dimension of the problem. Another approach is to use a slight modification of the chord-oracle described by Pollack et al. [83, Section 3].

Once the solution to (P2) restricted to γ is known, we can follow the same idea used by Megiddo [71] to find the side of γ containing the global solution to (P2). Find the apex functions that define the minimum restricted to γ. Since $\phi(x) = F_P(x)$ is locally defined by these functions, we can decide in which side the minimum lies using convexity. We obtain the following result.

Lemma 7.7.1. *Let R' be a convex trapezoid contained in P such that R' contains the geodesic center of P. Given the set of all apexed triangles of τ that intersect R', we can compute the geodesic center of P in $O(n)$ time.*

The following theorem summarizes the result presented in this chapter.

Theorem 7.7.2. *We can compute the geodesic center of any simple polygon P of n vertices in $O(n)$ time.*

7.8 Bounding the VC-dimension

In this section we provide the proof of Lemma 7.6.3. That is, we want to prove that the set system $(\mathcal{C}_R, \varphi)$ has constant VC-dimension. Recall that $\mathcal{C}_R$ is a set of chords of P and φ is the set of all open 4-cells of P.

Let $A \subseteq \mathcal{C}_R$ be a subset of chords. We say that A is *shattered* by φ if each of the subsets of A can be obtained as the intersection of some $Z \in \varphi$ with A, i.e., if for each $\sigma \subseteq A$, there exists $Z \in \varphi$ such that $\sigma = Z \cap A$. The *VC-dimension* of $(\mathcal{C}_R, \varphi)$ is the supremum of the sizes of all finite shattered subsets of $\mathcal{C}_R$.

Let $\mathcal{H} = \{H : H \text{ is a half-polygon of } P\}$. Because each 4-cell of P is the intersection of at most four half-polygons of P, the following result is a direct consequence of Proposition 10.3.3 of [68, Chapter 10].

Lemma 7.8.1. *If $(\mathcal{C}_R, \mathcal{H})$ has VC-dimension d, then $(\mathcal{C}_R, \varphi)$ has VC-dimension $O(d)$.*

Let A be an arbitrary subset of $\mathcal{C}_R$ such that $|A| = \kappa$ for some constant $\kappa > 6$ to be determined later. Recall that if no subset of $\mathcal{C}_R$ of size κ is shattered by $\mathcal{H}$, then the VC-dimension of $(\mathcal{C}_R, \mathcal{H})$ is at most $\kappa - 1$.

By Lemma 7.8.1, it suffices to show that A is not shattered by $\mathcal{H}$ to bound the VC-dimension of $(\mathcal{C}_R, \mathcal{H})$ and hence of $(\mathcal{C}_R, \varphi)$. We spend the remainder of this section proving that A is not shattered by $\mathcal{H}$.

A 6-cell is *strict* if it is defined as the intersection of six half-polygons, none of them redundant, i.e., the removal of any of them modifies the 6-cell.

Lemma 7.8.2. *If there are six chords of A supporting six half-polygons whose intersection defines a strict 6-cell σ, then A is not shattered by $\mathcal{H}$.*

Proof. Let $C_1, \ldots, C_6$ be the chords supporting the six half-polygons whose intersection defines σ. Assume that $C_1, \ldots, C_6$ appear in this order when walking clockwise along the boundary of σ. Note that any half-polygon that intersects C_1, C_3 and C_5 must intersect either C_2, C_4 or C_6. Therefore, the set $\{C_1, C_3, C_5\} \subseteq A$ cannot be obtained as the intersection of some half-polygon $H \in \mathcal{H}$ with A. Consequently A is not shattered by $\mathcal{H}$. □

Given two chords C_1 and C_2 of A, we say that C_1 and C_2 are *separated* if there exists a chord $S \in A$ such that C_1 and C_2 lie in different open half-polygons supported by S. In this case, we say that S *separates* C_1 from C_2.

Note that if A contains two chords C_1 and C_2 that are separated by a chord $S \in A$, then any half-polygon that intersects both C_1 and C_2 must also intersect S. In this case, the subset $\{C_1, C_2\}$ cannot be obtained as the intersection of a half-polygon $H \in \mathcal{H}$ with A, i.e., A is not shattered by $\mathcal{H}$. Therefore, we assume from now on that no two chords of A are separated.

Let G_A be the intersection graph of A, i.e., the graph with vertex set A and an edge between two chords if they intersect. An Erdös-Szekeres type result from Harborth and Möller [53] shows that every arrangement of nine pseudo-lines determines a subarrangement with a hexagonal face. Thus, if G_A has a clique of size nine, then this subset of chords is a set of pseudo-lines. Therefore, it contains a subset of 6 chords that define a strict 6-cell. In this case, Lemma 7.8.2 implies that A is not shattered by $\mathcal{H}$. Consequently, we assume from now on that G_A has no clique of size nine.

Turán's Theorem [90] states that if G_A has no clique of size nine, then it has at most $(7/16)\kappa^2$ edges. Let C_1 be the chord in A with the smallest degree in G_A. Notice that C_1 has degree at most $7\kappa/8$. Therefore, there are at least $\kappa/8$ chords of A that do not intersect C_1. Consider the two half-polygons supported by C_1 and note that one of them, say P', contains at least $\kappa/16$ chords of A that do not intersect C_1. Let A' be the set containing these chords.

Let G'_A be subgraph of G_A induced by A' and let C_2 be the chord of A' with smallest degree in G'_A. Because G'_A has no clique of size nine, we infer that C_2 has degree at most $7|A'|/8$. Repeating the same argument, there is a set A'' of at least $|A'|/8$ chords of A' that intersect neither C_2 nor C_1. Because we assumed that no two chords of A are separated, all chords in A'' must be contained in the half-polygon supported by C_2 that contains C_1. Otherwise, C_1 and some of these chords are separated by C_2.

Let G''_A be the subgraph of G_A induced by A''. Repeating the above procedure recursively on G''_A and A'' four more times, we obtain a set $C_1, \ldots, C_6$ of pairwise disjoint chords such that for each $1 \leq i \leq 6$, $C_1, \ldots, C_{i-1}$ are contained in the same half-polygon supported by C_i. Consequently, the set $\{C_1, \ldots, C_6\}$ bounds a strict 6-cell.

The above process can be applied as long as

$$\left(\frac{1}{16}\right)\left(\frac{1}{8}\right)^4 \kappa \geq 1.$$

That is, as long as $|A| \geq 65,536$ we can always find 6 such chords defining a strict 6-cell. In this case, Lemma 7.8.2 implies that A is not shattered by $\mathcal{H}$. Consequently, no set of 65,536 chord is shattered, i.e., the set system $(\mathcal{C}_R, \mathcal{H})$ has VC-dimension at most 65,535. By Lemma 7.8.1, we obtain the following result which concludes the proof presented in this chapter.

Lemma 7.6.3. *The set system $(\mathcal{C}_R, \varphi)$ has constant VC-dimension.*

7.9 Concluding remarks and future work

In this chapter we studied the complexity of computing the geodesic center of a simple n-gon. We presented a deterministic algorithm that computes this geodesic center in

$O(n)$ time.

Another way to compute the center of a simple polygon is to compute the farthest-point Voronoi diagram of its vertices using the geodesic distance. While the best known algorithm for this task runs in $O(n \log n \log n)$ time [77], no lower bound is known for this instance of the problem. Therefore, it is natural to ask if the farthest-point Voronoi diagram of the set of vertices of a simple polygon can be computed in $O(n)$ time.

To generalize the result presented in this chapter, we ask the following question. Given a set S of m points inside of a simple polygon P with n vertices, how fast can we compute the center of S inside P? That is, the point in P that minimizes the maximum geodesic distance to a point of S. While the (geodesic) farthest-point Voronoi diagram of S provides the answer in $O(n \log \log n + m \log m)$ time, we ask whether it is possible to compute this center in $O(n + m)$ time.

Bibliography

[1] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. In *Proceedings of the ACM symposium on Theory of computing*, pages 39–45, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7.

[2] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4:591–604, September 1989. ISSN 0179-5376.

[3] H. K. Ahn, L. Barba, P. Bose, J.-L. D. Carufel, M. Korman, and E. Oh. A Linear-Time Algorithm for the Geodesic Center of a Simple Polygon. In L. Arge and J. Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–223, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-83-5.

[4] S. Allen, L. Barba, J. Iacono, and S. Langerman. Incremental Voronoi diagrams. In S. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume tbd of *Leibniz International Proceedings in Informatics (LIPIcs)*, page tbd, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[5] G. Aloupis, L. Barba, and S. Langerman. Circle separability queries in logarithmic time. In *Proceedings of the 24th Canadian Conference on Computational Geometry*, CCCG'12, pages 121–125, August 2012.

[6] B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1-4):109–140, 1989.

[7] B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.

[8] B. Aronov, P. Bose, E. Demaine, J. Gudmundsson, J. Iacono, S. Langerman, and M. Smid. Data structures for halfplane proximity queries and incremental voronoi diagrams. In *LATIN 2006: Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 80–92. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-32755-4.

[9] T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.

[10] F. Aurenhammer and R. Klein. Voronoi diagrams. *Handbook of Computational Geometry*, 5:201–290, 2000.

[11] S. W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains. *Discrete & Computational Geometry*, 50(2):306–329, 2013.

[12] S. W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal domain. In *Proceedings of CCCG*, 2014.

[13] S. W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. In *Proceedings of LATIN*, pages 120–131, 2014.

[14] L. Barba. Disk constrained 1-center queries. In *Proceedings of the Canadian Conference on Computational Geometry*, CCCG'12, pages 15–19, 2012.

[15] L. Barba and S. Langerman. Optimal detection of intersections between convex polyhedra. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1641–1654. SIAM, 2015.

[16] L. Barba, P. Bose, and S. Langerman. Optimal algorithms for constrained 1-center problems. In *LATIN 2014: Theoretical Informatics*, pages 84–95. Springer, 2014.

[17] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 80–86, New York, NY, USA, 1983. ACM. ISBN 0-89791-099-0.

[18] J. L. Bentley and J. B. Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[19] T. Biedl and D. F. Wilkinson. Bounded-degree independent sets in planar graphs. *Theory of Computing Systems*, 38(3):253–278, 2005.

[20] V. Bilò, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Algorithms–ESA 2005*, pages 460–471. Springer, 2005.

[21] P. Bose and G. Toussaint. Computing the constrained Euclidean, geodesic and link centre of a simple polygon with applications. In *Computer Graphics International, 1996. Proceedings*, pages 102 –111, jun 1996. doi: 10.1109/CGI.1996.511792.

[22] P. Bose and Q. Wang. Facility location constrained to a polygonal domain. In S. Rajsbaum, editor, *LATIN 2002: Theoretical Informatics*, volume 2286 of *Lecture Notes in Computer Science*, pages 153–164. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43400-9. doi: 10.1007/3-540-45995-2_18.

[23] P. Bose, S. Langerman, and S. Roy. Smallest enclosing circle centered on a query line segment. In *Proceedings of the Canadian Conference on Computational Geometry*, CCCG'08, pages 167–170, 2008.

[24] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22:547–567, 1999. ISSN 0179-5376.

[25] T. M. Chan. More planar two-center algorithms. *Computational Geometry*, 13 (3):189–198, 1999.

[26] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 1196–1202, New York, NY, USA, 2006. ACM. ISBN 0-89871-605-5.

[27] B. Chazelle. A theorem on polygon cutting with applications. In *Proceedings of FOCS*, pages 339–349, 1982.

[28] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

[29] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21:586–591, 1992.

[30] B. Chazelle and D. Dobkin. Detection is easier than computation (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 146–153, 1980.

[31] B. Chazelle and D. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of ACM*, 34(1):1–27, Jan. 1987. ISSN 0004-5411.

[32] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair, and parametric searching. *DCG*, 10:183–196, 1993. ISSN 0179-5376.

[33] Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992.

[34] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[35] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

[36] H. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8: 131–152, 1992.

[37] D. Dobkin and D. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[38] D. Dobkin and D. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6(3):381–392, 1985.

[39] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach. *Automata, Languages and Programming*, pages 400–413, 1990.

[40] D. Dobkin and D. Souvaine. Detecting the intersection of convex objects in the plane. *Computer Aided Geometric Design*, 8(3):181–199, 1991.

[41] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9(6):518–533, 1993.

[42] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *Journal of Algorithms*, 6(2):213–224, 1985.

[43] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[44] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986. ISSN 0179-5376.

[45] J. Erickson. Space-time tradeoffs for emptiness queries. *SIAM Journal on Computing*, 29(6):1968–1996, 2000.

[46] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 434–444, New York, NY, USA, 1988. ACM. ISBN 0-89791-264-0.

[47] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[48] T. F. Gonzalez. Covering a set of points in multidimensional space. *Information Processing Letters*, 40(4):181 – 188, 1991. ISSN 0020-0190.

[49] J. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition*. CRC Press LLC, 2004.

[50] L. Guibas. *Kinetic data structures*. CRC Press, 2004.

[51] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[52] L. Guibas, J. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In G. Schmidt and R. Berghammer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 570 of *Lecture Notes in Computer Science*, pages 113–125. Springer Berlin / Heidelberg, 1992. ISBN 978-3-540-55121-8.

[53] H. Harborth and M. Möller. *The Esther-Klein-problem in the projective plane*. Inst. für Mathematik, TU Braunschweig, 1993.

[54] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[55] J. Hershberger and S. Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

[56] F. Hurtado, V. Sacristan, and G. Toussaint. Some constrained minimax and maximim location problems. *Studies in Locational Analysis*, 15:17–35, 2000.

[57] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[58] B. Joe and R. B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27:458–473, 1987. ISSN 0006-3835.

[59] Y. Ke. An efficient algorithm for link-distance problems. In *Proceedings of SoCG*, pages 69–78, 1989.

[60] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[61] R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989. ISBN 3-540-52055-4.

[62] R. Klein, E. Langetepe, and Z. Nilforoushan. Abstract Voronoi diagrams revisited. *Computational Geometry*, 42(9):885 – 902, 2009. ISSN 0925-7721.

[63] D. T. Lee. On finding the convex hull of a simple polygon. *International Journal of Parallel Programming*, 12(2):87–98, 1983.

[64] D.-T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[65] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, volume 1, pages 602–608, 1998.

[66] J. Matousek. Computing the center of planar point sets. *Discrete and Computational Geometry: papers from the DIMACS special year*, 6:221, 1991.

[67] J. Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50(2):203–208, 1995.

[68] J. Matoušek. *Lectures on Discrete Geometry*, volume 108. Springer New York, 2002.

[69] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete & Computational Geometry*, 10(1):215–232, 1993.

[70] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[71] N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–610, 1989.

[72] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[73] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.

[74] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[75] B. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In *Proceedings of CG*, pages 203–215, 1991.

[76] B. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. *Computational Geometry: Theory and Applications*, 6: 169–194, 1996.

[77] E. Oh, L. Barba, and H.-K. Ahn. The farthest-point geodesic voronoi diagram of points on the boundary of a simple polygon. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51, page 56. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[78] J. O'Rourke. *Computational geometry in C*. Cambridge university press, 1998.

[79] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor. A new linear algorithm for intersecting convex polygons. *Computer Graphics and Image Processing*, 19(4): 384 – 391, 1982.

[80] M. H. Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983.

[81] M. H. Overmars and J. Van Leeuwen. Maintenance of configurations in the plane. *Journal of computer and System Sciences*, 23(2):166–204, 1981.

[82] S. Pettie. Applications of forbidden 01 matrices to search tree and path compression-based data structures. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1457–1467, 2010.

[83] R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(1):611–626, 1989.

[84] M. I. Shamos. Geometric complexity. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, pages 224–233. ACM, 1975.

[85] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pages 208–215. IEEE, 1976.

[86] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[87] S. Suri. *Minimum Link Paths in Polygons and Related Problems*. PhD thesis, Johns Hopkins Univ., 1987.

[88] S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39(2):220–235, 1989.

[89] J. J. Sylvester. A Question in the Geometry of Situation. *Quarterly Journal of Pure and Applied Mathematics*, 1, 1857.

[90] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48(436-452): 137, 1941.

[91] A. C.-C. Yao. Decision tree complexity and Betti numbers. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 615–624, New York, NY, USA, 1994. ACM. ISBN 0-89791-663-8.

[92] G. M. Ziegler. *Lectures on Polytopes*, volume 152. Springer, 1995.