Algorithms for Geometric Facility Location: Centers in a Polygon and Dispersion on a Line

by

Anurag Murty Naredla

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Boris Aronov Professor of Computer Science, New York University
Supervisor(s):	Anna Lubiw Professor, School of Computer Science, University of Waterloo
Internal Member:	Therese Biedl Professor, School of Computer Science, University of Waterloo
Internal Member:	Eric Blais Associate Professor, School of Computer Science, University of Waterloo
Other Member(s):	Stephen Vavasis Professor, Dept. of Combinatorics and Optimization, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis is based on the following papers that I have co-authored:

1. [18]: Dispersion for Intervals: A geometric approach. Joint with Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. *Symposium on Simplicity in Algorithms, 2021.*

2. [71]: The Visibility Center of a Simple Polygon. Joint with Anna Lubiw. *European Symposium on Algorithms, 2021.*

3. The Geodesic Edge Center of a Simple Polygon. Joint with Anna Lubiw. *Symposium on Computational Geometry, 2023.*

Abstract

We study three geometric facility location problems in this thesis.

First, we consider the *dispersion problem in one dimension*. We are given an ordered list of (possibly overlapping) intervals on a line. We wish to choose exactly one point from each interval such that their left to right ordering on the line matches the input order. The aim is to choose the points so that the distance between the closest pair of points is maximized, i.e., they must be socially distanced while respecting the order. We give a new linear-time algorithm for this problem that produces a lexicographically optimal solution. We also consider some generalizations of this problem.

For the next two problems, the domain of interest is a simple polygon with n vertices.

The second problem concerns the *visibility center*. The convention is to think of a polygon as the top view of a building (or art gallery) where the polygon boundary represents opaque walls. Two points in the domain are visible to each other if the line segment joining them does not intersect the polygon exterior. The distance to visibility from a source point to a target point is the minimum geodesic distance from the source to a point in the polygon visible to the target. The question is: Where should a single guard be located within the polygon to minimize the maximum distance to visibility? For m point sites in the polygon, we give an $O((m+n)\log(m+n))$ time algorithm to determine their visibility center.

Finally, we address the problem of locating the *geodesic edge center* of a simple polygon—a point in the polygon that minimizes the maximum geodesic distance to any edge. For a triangle, this point coincides with its incenter. The geodesic edge center is a generalization of the well-studied geodesic center (a point that minimizes the maximum distance to any vertex). Center problems are closely related to farthest Voronoi diagrams, which are well-studied for point sites in the plane, and less well-studied for line segment sites in the plane. When the domain is a polygon rather than the whole plane, only the case of point sites has been addressed—surprisingly, more general sites (with line segments being the simplest example) have been largely ignored. En route to our solution, we revisit, correct, and generalize (sometimes in a non-trivial manner) existing algorithms and structures tailored to work specifically for point sites. We give an optimal linear-time algorithm for finding the geodesic edge center of a simple polygon.

Acknowledgements

To Him for His Constant Guidance.

> There are more things in
> Heaven and Earth, Horatio,
> Than are dreamt of in your
> Philosophy.
>
> *Hamlet*, William Shakespeare

Without my advisor, Anna Lubiw, none of this was remotely possible. Anna is a brilliant mentor, and her guidance helped me at every step. I have learnt from Anna that if I am unable to explain something simply, it is probably because I do not understand it myself... Most importantly, Anna has always been very kind, patient, and supportive (with a Ph.D. student who self-identifies as lethargic).

Thanks also to Therese Biedl for her careful reading of my thesis and the detailed comments. Post-pandemic, I suppose Therese was the person I interacted with most at the university. She was always happy to talk at the algorithms office, and generous with ideas and advice.

I am grateful to Eric Blais for agreeing to be on my thesis committee, and for great advice for the future. It was also a pleasure interacting with him regarding the A & C Seminar. I also thank Stephen Vavasis for being part of my thesis committee, and his careful perusal of my work. Finally, Boris Aronov gave nice and detailed comments on the thesis (along with some humorous ones). Thank you.

It was a lot of fun interacting with Ian Munro and attending his Data Structures course. He always had amazing anecdotes and valuable advice for me. I have learnt a lot from Lap Chi's courses, talks, and style of teaching. I am indebted to Timothy Chan for permitting me to attend some of his courses. I am grateful to the ever-friendly Ahmad for things including, but not limited to, research. I appreciate Heiko's help and understanding during my transition from Waterloo to Bonn. The entire Theory group here has made me feel at home.

I am also ever thankful to Prof V. Prem Pyara (Dayalbagh), Prof. Prem Kalra (IIT-D), Prof. Vijay Natarajan (IISc), and Prof. Sathish Vadhiyar (IISc) for their advice and teaching at various points in my life.

–

Ankit, of course, deserves special mention—he has always been a genuine friend and a constant source of inspiration. I am grateful also to Mohit, Aakriti (and Eshin, and Erisha) and Pallavi, Sahej, Ratan, and Aishwarya—their friendship is invaluable to me.

And to Ben, Werner, Remy, David, Pavle, Jim, our friends at TSG (especially little Arsh), Denis, Karthik, Joel & Ethan, Stephanie, Hong, Amolak, Vijay & Vijay, Harry B., Kevin, Graeme, Nathan, Norm, Dale, Vikram-Kirti, and all members of the Danebury Metal Detecting Club ...

Amrita, Shruti, and Geras were great amigos for innumerable drives on the trusty Crosstrek—both short and long. Ustad Saami, Kishori Amonkar, Radiohead, Andrew Bird, and the Dagar brothers, with able assistance from the Spotify algorithm generously provided us with the background score.

–

And thanks to my family for everything ... To my late Mamma and Ammamma, and to Thathagaru for all their love.

Mom, Dad, and MS—for the support, the laughs, and the love.

Arat for being a friend, a brother, a confidante, and all in all someone I continue to learn a lot from. While I was away from home, he stepped up and took care of Nanna. I truly look up to him. Premsakhi (the newest member of HDHC), for trusting someone who doesn't know what to do with his life for career advice.

Mummy taught me how to read, write, and program—my first teacher. She instilled in me the love of books and knowledge. Thanks for the unwavering love and support. And to the memory of my father—my first hero. It deeply saddens me that he will not read these words ... It was his encouragement that made me consider higher studies. Even in his last days, he would try to understand my research work with great interest and ask me for applications of geometric algorithms to his farming work. I hope to have imbibed at least a small fraction of their integrity, values, and work ethic.

–

All of this would of course be meaningless without my wife, Smriti. This has been quite an adventure, and there is no better person to go on an adventure with! She has had to sacrifice a lot in the past years, and has done so with a smile. The most fun I had in Canada were the concerts, the movies, and the road trips we went on—and the endless discussions about the most obscure of topics.

I cannot find words to express how much all the love and support means to me—and with you I know I don't have to!

Dedication

To Mummy, and the memory of dearest Nanna ...

Table of Contents

List of Figures

Chapter 1

Geometric Facility Location: An Informal Introduction

The point is there ain't no point

No Country for Old Men,
Cormac McCarthy

Facility location problems are among the most fundamental questions about any geometric domain. Such questions are also immensely practical. *"Where should the theme park be built to maximize the number of visitors?"*, a real estate developer might inquire. They might want to locate it far from the competing establishments, yet in a 'central' location so that it is accessible to a large population. An army general might wish to build a military base in a strategic location. The requirements differ greatly from the earlier problem and considerably change its flavor.

Even for the triangle, various 'centers' have been studied since the time of the Ancient Greeks. Some of these include the incenter, the centroid, and the circumcenter. These triangle centers possess properties that allow them to be re-interpreted as solutions to facility location problems. If you seek a location that is "not too far" from the edges of a triangle, the incenter of the triangle is an optimal solution. For the interested reader, the Encyclopedia of Triangle Centers [63] identifies 50,406 triangle centers (as of 6 June 2022).

Moving to the Euclidean plane, James Joseph Sylvester [98] posed the following facility location problem in 1857: given a set of point sites on the plane, find the center of the smallest circle that contains all of them. See Figure 1.1. Equivalently, we can think of

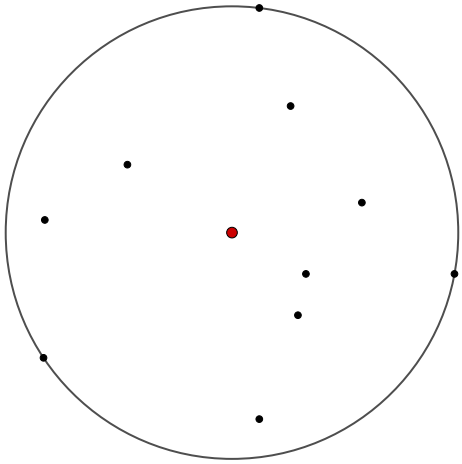


Figure 1.1: Sylvester's problem: Find the smallest circle enclosing a given set of points (in black). The center of this circle (in red) is the point in the plane that minimizes the distance to the farthest point.

this as a problem in facility location: given a set of sites, find a point in the plane that minimizes the maximum distance to a site. In this thesis, we refer to facility location problems with this general flavor (i.e. minimize the maximum distance to an object from an input set) as ***center problems*** (See Chapter 4). In Chapters 5 and 6, we deal with interesting generalizations of Sylvester's problem.

Generalized versions of facility location problems are studied to model real-world applications more precisely. We enumerate a few such generalizations below:

1. **More than one facility**: Of course, there is no reason to restrict ourselves to just one facility—we might want to build several facilities in our geometric domain subject to certain needs. There is an obvious extension of the minimax problems to k-centers instead of one [81, 89]. In higher dimensions, these problems are closely related to clustering problems.

   Another important class of location problems with multiple facilities is the class of *dispersion problems*—here, we want the locations to be 'far apart' in some sense [16, 20, 23]. For instance, we may want to maximize the sum of distances between the facilities (called max-sum dispersion), or we may want to maximize the distance between the closest pair of facilities (called max-min dispersion). Dispersion problems are useful in modeling real-life scenarios where proximity among the facilities is to

be discouraged. It may prove perilous to build chemical factories close to each other, and it is pointless to have rest-stops adjacent to each other on a highway.

2. **Generalized sites**: In Sylvester's problem, the input consists of point sites. What if the sites are not points but segments, circles, or polygons? What is the solution to the minimax problems in these cases? Variants of Sylvester's problem with generalized sites have been studied extensively [17, 60, 27]. In many cases, efficient algorithms make use of the farthest-site Voronoi diagram to arrive at the solution to the center problem [28].

3. **Generalized distances**: Most of the geometric location problems we have discussed perform their optimization using Euclidean distances. The distance between two points is the length of the line segment that connects them. This is not always a faithful representation of reality. A practical and well-researched generalization is to use weighted distances to the point sites—a way to assign *importance* to the site [40, 80].

   A metric relevant to us is the ***geodesic metric*** (or the ***shortest-path metric***) inside a simple polygon. The distance between two points is the length of the shortest path that connects them while staying inside the polygon [66]. Problems that are well understood in the Euclidean metric become very complicated with the change of metric. Relevant examples here include the extension of Sylvester's original problem to the geodesic metric [4, 12, 91].

   Distance to visibility. The distance between two points under the geodesic metric described above may be interpreted as the distance to be covered starting from a source point to *reach* a target point. Another kind of distance pertinent to us is the ***distance to visibility***. The idea is that in some situations it suffices to *see* your target location (instead of reaching it). Visibility is one of the earliest notions studied in computational geometry. Two points are ***visible*** to each other (or ***see*** each other) if the line segment joining them does not intersect any opaque obstacles [88]. Suppose the geometric domain of interest is a simple polygon. For a simple polygon, the boundary is assumed to be an obstacle both to visibility and motion. The (geodesic) distance to visibility is defined in terms of a starting point and a target point to be viewed. It is the length of the shortest path within the polygon that connects the starting point to some point visible from the target point [9].

1.1 Problems Addressed in this Thesis

We give efficient algorithms for three different geometric facility location problems in this thesis.

1. **The dispersion problem in one dimension**: Given n intervals on a line and a specified order, locate a representative point on each interval so that the distance between the closest pair is maximized. The n points must respect the specified order.

   This problem has been studied earlier and has a rather complicated optimal solution [68] that we simplify using a geometric transformation of the problem. An advantage of our approach is that it allows us to solve generalized versions of the problem that are either too complex or unsolvable by the earlier techniques.

   The techniques use elementary ideas from the early days of computational geometry. We show that the dispersion problem is equivalent to finding shortest paths in a geometric domain. Furthermore, we show that the solution provided by our methods guarantees the stronger property of *lexicographic dispersion* (as opposed to the max-min dispersion stated in the problem description). We refer the reader to Chapter 3 for more details.

   Our results. We describe in Chapter 3 an optimal linear-time algorithm for the dispersion problem in one dimension. We also describe methods of solution for some natural variants of the problem. This work was published in Biedl et al. [18].

2. **The visibility center problem**: Given a simple polygon with n vertices and k viewpoints, find a point in the polygon that minimizes the maximum geodesic distance to visibility to any of the viewpoints.

   This problem is a facility location problem that combines two classical threads in computational geometry—motion planning and visibility. Although a very natural notion, we are not aware of prior work that addresses center problems of this type.

   The visibility center problem can be interpreted as a variant of the art gallery problem—the problem of finding the minimum number of static points (guards) in a simple polygon so that each point in the polygon is visible to at least one guard. For the visibility center, we are provided with just one guard. Unlike the art gallery problem, however, the guard is allowed to move.

   Our solution first reduces the visibility center problem to the problem of determining the geodesic center of half-polygons in a simple polygon—a problem we believe to be of independent interest. To solve the latter problem, we need to extend the

techniques of Pollack et al. [91] that were used to solve the geodesic center problem for the vertices of a polygon.

Our results. We describe an $O((n+k)\log(n+k))$ time algorithm for determining the visibility center in Chapter 5. Our solution transforms the original problem to that of determining the geodesic center of $O(k)$ half-polygons in an n-vertex simple polygon. The geodesic center problem is then solved within the same time bounds. This work was published in Lubiw and Naredla [71].

3. **The geodesic edge center problem**: Given a simple polygon with n vertices, find a point in the polygon that minimizes the geodesic distance to the farthest edge.

   This problem generalizes the point sites of the geodesic center problem to sites that are the edges of the polygon. For an acute-angled triangle, the geodesic center is its circumcenter, while the geodesic edge center is the incenter. Thus, we find it rather surprising that only centers of vertices have been studied under the geodesic metric, while edge centers (in fact, centers for any set of non-point sites) are largely ignored.

   The framework used by Ahn et al. [4] for the vertex center carries through for the edge center. However, we must generalize existing structures and algorithms for point sites to the case of edges. This includes a linear-time algorithm for determining the farthest-edge Voronoi diagram restricted to the polygon boundary. We also fix a few gaps in the linear-time algorithm for the geodesic center for vertices.

   Our results. We discuss a linear-time algorithm that locates the edge center of a simple polygon in Chapter 6. Our results from this chapter are currently under review.

Chapter 2

Some Geometric Preliminaries

I hate definitions

Vivian Grey, Benjamin Disraeli

We begin by reviewing a few concepts from computational geometry. These concepts will be used throughout this thesis, especially Chapter 5 and Chapter 6.

A ***simple polygon*** P is a closed region of the plane enclosed by a cycle of straight line segments such that nonadjacent segments do not intersect and adjacent segments intersect only at their common endpoint [99]. See Figure 2.1. The segments are the ***edges*** of the polygon, and the common endpoints between adjacent edges are its ***vertices***. Unless stated otherwise, we will assume that the polygon P has n vertices (and therefore, n edges). A polygon vertex is ***reflex*** if the internal angle at the vertex is greater than π. A partition of the polygon P into triangles is called a ***triangulation*** of P. The triangulation algorithm of Chazelle [24] triangulates any simple polygon in $O(n)$ time.

The edges of the polygon P form the ***boundary*** ∂P and enclose the ***interior*** of the polygon. Points of the plane not lying in the interior or on the boundary of P are ***exterior*** to the polygon. A ***chord*** of P is a line segment joining two points on the polygon boundary that does not intersect its exterior. See Figure 2.2. A chord of a polygon that does not intersect any reflex vertices (except at its endpoints) splits the polygon into two ***half-polygons***. Note that the definition allows any edge of the polygon to be a chord, and one of the half-polygons defined by the edge is empty.

Take any two points a and b that lie either in the interior of P or on its boundary ∂P. The ***geodesic distance*** $d(a, b)$ between a and b is the length of a shortest path

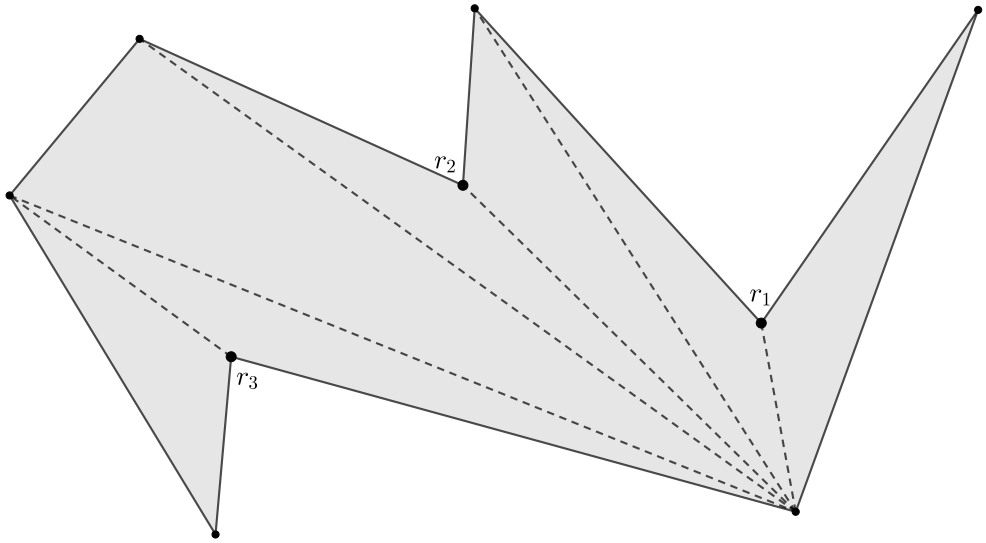


Figure 2.1: A simple polygon and its triangulation. The vertices r_1,r_2, and r_3 are reflex vertices.

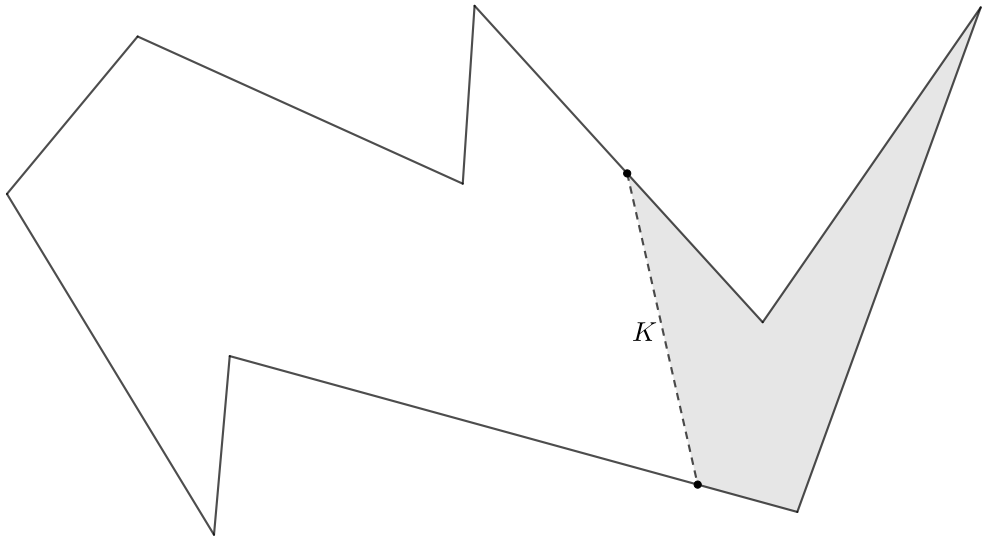


Figure 2.2: A chord K of the polygon splits it into two half-polygons. One of the half-polygons is shaded in this figure.

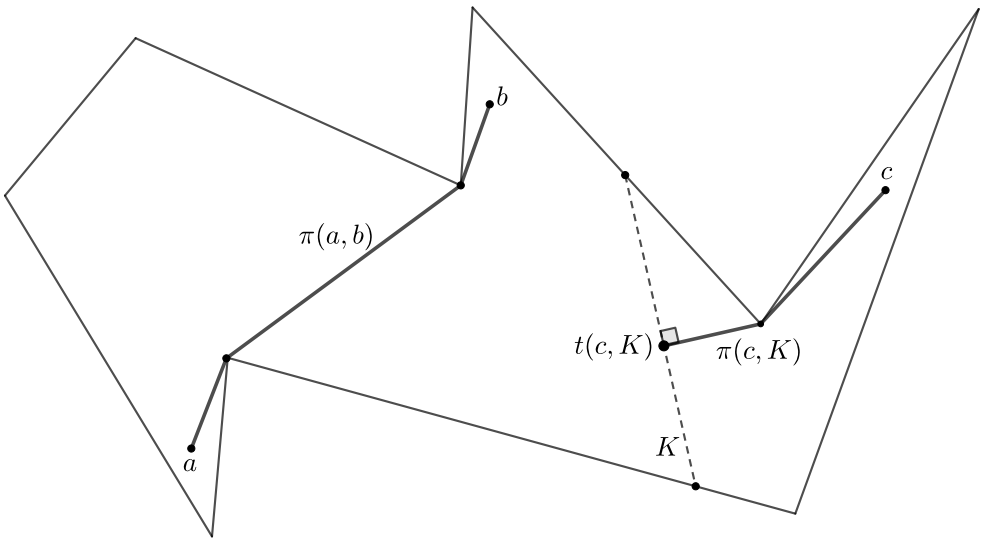


Figure 2.3: The geodesic or shortest path $\pi(a, b)$ between points a and b. Also shown is the shortest path $\pi(c, K)$ between the point c and a chord K. The shortest path joins c to the terminal point $t(c, K)$—the point on K geodesically closest to c.

joining a and b that does not intersect the exterior of P. The ***shortest path*** or ***geodesic*** between the points a and b is actually unique (Lee and Preparata [66]), and is denoted $\pi(a, b)$. It is a polygonal path that bends at reflex vertices of P (Figure 2.3). For a triangulated polygon with n vertices, the geodesic distance between any two points may be determined in $O(\log n)$ time after $O(n)$ time preprocessing, by an algorithm of Guibas and Hershberger [50].

The ***geodesic distance*** $d(a, K)$ between a and chord K is the length of a shortest path joining a to any point on K, such that the path does not intersect the exterior of P. This path is also unique [91] and is denoted $\pi(a, K)$. The endpoint of $\pi(a, K)$ on K is called the ***terminal*** $t(a, K)$ on K with respect to a (Figure 2.3).

From any point a in the polygon, the ***shortest path tree*** to the vertices of P is the union of shortest paths to all the vertices of the polygon. The shortest path tree from any point in a triangulated n-vertex simple polygon can be constructed in $O(n)$ time (Guibas et al. [49]). The shortest path tree from any point $a \in P$ may be used to obtain the ***shortest path map*** with respect to a: a decomposition of the simple polygon into triangular regions such that from all points within any triangle, the shortest path to a is combinatorially the same, i.e. uses the same sequence of vertices. The shortest path tree and the shortest path map to the vertices of the polygon from a point inside it are shown in Figure 2.4. The

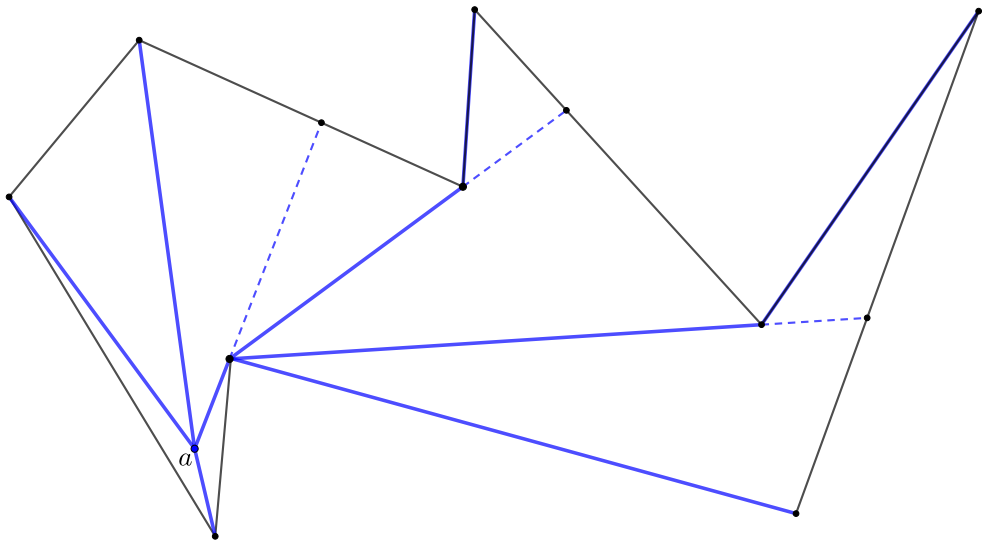


Figure 2.4: The shortest path tree from point a to the vertices of the simple polygon is shown in blue. Some edges of the shortest path tree are extended (dashed) to obtain the shortest path map.

shortest path map for any point in a simple polygon can be constructed in linear time.

A subset Q of P is ***geodesically convex*** if for any two points a and b in Q, the shortest (or "geodesic") path $\pi(a, b)$ in P is contained in Q. A function f defined on P is ***geodesically convex*** if f is convex on every geodesic path $\pi(a, b)$ in P, i.e., for points $x \in \pi(a, b)$, $f(x)$ is a convex function of $d(a, x)$.

A polygon with at least three vertices is called ***weakly simple*** if, for every $\epsilon > 0$, the vertices can be perturbed by at most ϵ to obtain a simple polygon [6]. For a set S in the polygon P, the ***geodesic convex hull*** of S in P is defined as the intersection of all geodesically convex sets in P that contain S.

Figure 2.5 shows the geodesic convex hull of a set of points in a simple polygon.

An important notion in computational geometry is that of visibility. Two points in a simple polygon P are ***visible*** to each other (or ***see*** each other) if the line segment joining them does not intersect the exterior of P. The set of all points visible from a point a in P is the ***visibility polygon*** of a with respect to P (Figure 2.6). Lee [67] gave a linear-time algorithm to construct the visibility polygon of any point in a simple polygon.

Let the set D denote the vertices of P. The graph obtained by drawing line segments between pairs of vertices in D that are visible to each other is called the ***visibility graph***

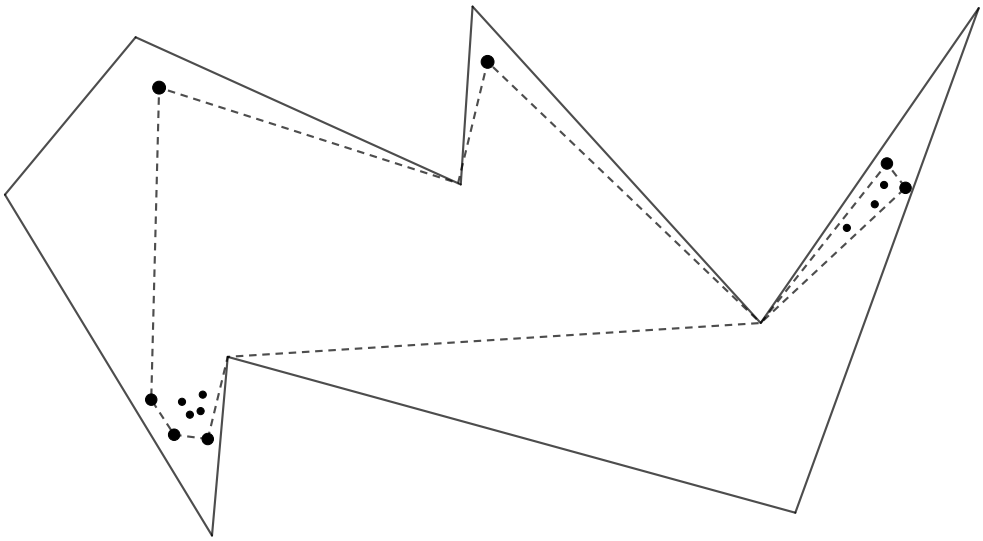


Figure 2.5: The geodesic convex hull of a set of points in a simple polygon.

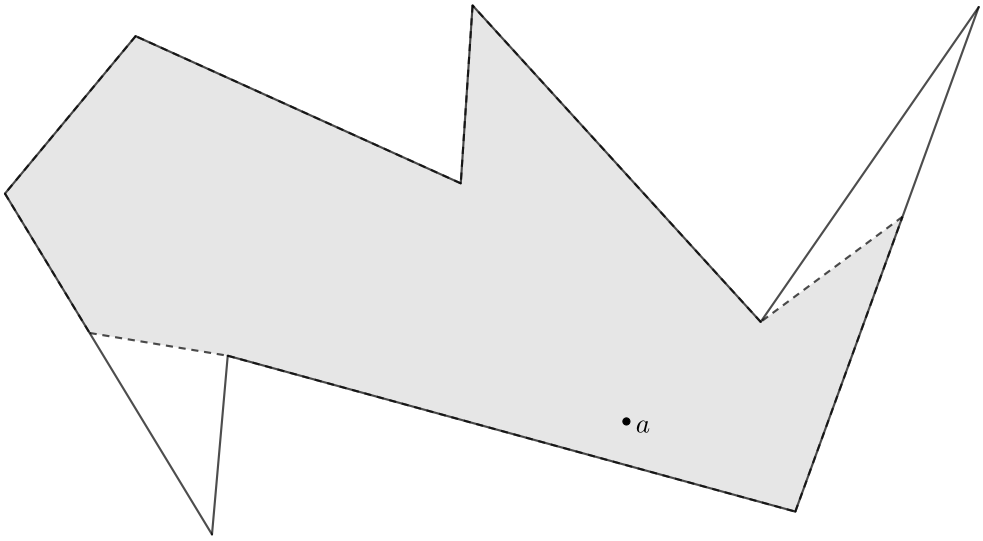


Figure 2.6: The visibility polygon of point a is shaded.

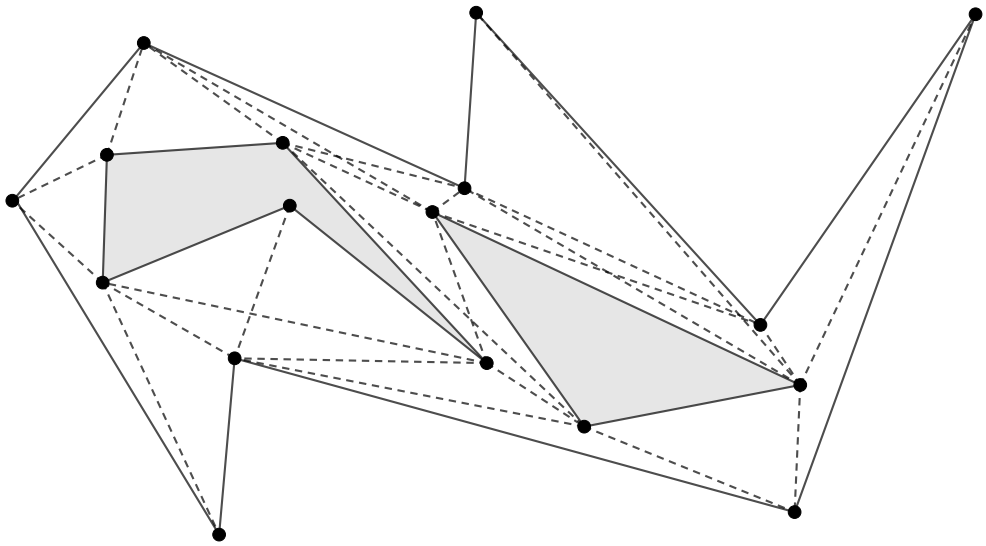


Figure 2.7: A polygon with two holes (shaded). The dashed segments are the segments of the visibility graph.

of the polygon P.

In general, the ***geodesic distance*** between a point a and a point set S in P is defined to be: $\min_{x \in S} d(a, x)$. Given an input set $\mathcal{S}$ of point sets in a polygon and a point x, we can define the ***farthest set*** $\mathcal{F}(x, \mathcal{S})$ as the subset of $\mathcal{S}$ geodesically farthest from x. For a non-empty subset $\mathcal{S}' \subseteq \mathcal{S}$, its ***Voronoi region*** is the set of points in P whose farthest set is $\mathcal{S}'$. A partition of the polygon P into the Voronoi regions of the non-empty subsets of $\mathcal{S}$ is called the ***(geodesic) farthest Voronoi diagram*** for the set $\mathcal{S}$ in the polygon P.

We define a ***polygon with holes*** as a simple polygon Q enclosing simple polygons $H_1, \ldots, H_k$ (the H_i are called ***holes***), such that the boundaries of $Q, H_1, \ldots, H_k$ do not intersect and no hole contains another [88]. Intuitively speaking, the interior of the holes behave like the polygon exterior. The ***boundary*** of the polygon with holes is defined by the edges of Q and the edges of $H_i, 1 \leq i \leq k$. The ***interior*** is the set of points $Q - \partial Q - \cup_{1 \leq i \leq k} H_i$. All other points in the plane make up the ***exterior*** of the polygon with holes.

For polygons with holes, we define a ***shortest path*** $\pi(a, b)$ between two points a and b in the interior or on the boundary as a path of shortest length that joins a and b and does not intersect the exterior. The shortest paths between points for polygons with holes are not necessarily unique. The length of a shortest path is the ***geodesic distance*** $d(a, b)$

between a and b. A ***geodesic*** between two points a and b in the interior or on the boundary is a locally shortest path that joins a and b. Unlike in the case of polygons without holes (i.e., simple polygons), a geodesic is not necessarily a shortest path—although the converse is always true.

Two points in a polygon Q with holes are ***visible*** to each other if the line segment joining them does not intersect the exterior. Let the set D denote the vertices of Q and the holes contained in it. The graph obtained by drawing line segments between pairs of vertices in D that are visible to each other is called the ***visibility graph*** of the polygon with holes. An output sensitive algorithm that takes $O(n \log n + k)$ time (where k is the number of edges in the visibility graph) was given by Ghosh and Mount [48]. Figure 2.7 shows a polygon with two holes and its visibility graph.

Other notions that we defined earlier for simple polygons (triangulation, farthest-site Voronoi diagrams) extend in a similar manner to polygons with holes.

Chapter 3

Dispersion Problems in One Dimension[1]

> I have only made this letter longer because I have not had the time to make it shorter
>
> Blaise Pascal

Suppose you are in a group of people waiting in line, and each of you has been assigned an interval that you must stay inside of. Because of social distancing, you all want to keep as far away from each other as possible, subject to staying in your assigned interval, and keeping your order in the line.

This problem fits into the large class of ***dispersion problems*** that involve choosing 'far apart' points from given sets. There are various notions of 'far apart' but our starting point is the work of Li and Wang [68] who considered the objective of maximizing the distance between the closest pair of points. We will refer to this objective as 'max-min' dispersion.

We introduce a new and stronger goal for dispersion, of which 'max-min' dispersion is a special case. Rather than just maximizing the minimum distance between points, our goal is to maximize the minimum distance, and subject to that, to maximize the second

[1] Based on joint work with Therese Biedl, Anna Lubiw, Peter Dominik Ralbovsky, and Graeme Stroud. Published in the paper 'Dispersion for Intervals: A Geometric Approach' at the Symposium on Simplicity in Algorithms (SOSA), 2021 ([18]).

minimum distance, and so on. In other words, we want the lexicographically maximum vector of distances, formalized as follows:

Lexicographic Dispersion for Ordered Intervals on a Line.

Input: a sequence of intervals $\{I_1, I_2, \ldots, I_n\}$ on the real line, where each interval I_i is given by its left endpoint l_i and right endpoint r_i.
Feasible solution: an ordered set of points $\boldsymbol{p} = (p_1, \ldots, p_n)$ such that $p_i \in I_i$, and $p_1 \leq p_2 \leq \cdots \leq p_n$.
Objective function: lexicographically maximize the ***sorted distance vector*** $\boldsymbol{\delta}(\boldsymbol{p})$, where $\boldsymbol{\delta}(\boldsymbol{p})$ is the list of $n-1$ distances $(p_{i+1} - p_i)$, $i = 1, \ldots, n-1$ sorted from minimum to maximum value.

We call this "lexicographic dispersion" to distinguish from the conventional "max-min" dispersion. The lexicographic maximum is a far more natural and desirable objective function for dispersion problems—if there are two points that are forced to be close together in any solution, we should not give up on spreading the other points as far apart as possible! For example, a max-min solution for dispersion for intervals might not even put the first and last point at the extreme endpoints of their intervals.

Our results. Our main result is a linear-time algorithm for lexicographic dispersion for ordered intervals via a transformation to finding a shortest path in a polygon. The algorithm determines if a feasible solution exists; if so, it computes the solution with optimal lexicographic dispersion. We show that there is a unique "Pareto" (or local) optimal solution, i.e., a solution where no single point can be moved to improve the objective function. Compared to Li and Wang's solution based on a greedy approach, our algorithm is more powerful and much simpler.

In Section 3.3 we apply our geometric approach to lexicographic dispersion for ordered intervals on a closed curve, achieving a simpler linear time algorithm for disjoint intervals, and a new polynomial time algorithm for overlapping intervals.

3.1 Background

Dispersion problems are well-studied in operations research, algorithmic game theory, and computational geometry because of their applications in facility location, and are of rising interest in information retrieval because of applications in "diversified" sampling. An

important case is "max sum diversification"—to select k points from a given set of n points to maximize the sum of the pairwise distances [23, 21, 44]. Another more geometric dispersion problem (related to packing problems) is to select k points from a polygonal region in the plane to maximize the minimum distance between any pair of points [16]. Most of these problems are NP-hard, which means that research is focused on heuristics and approximation algorithms.

The version of dispersion that we are interested in is to find one point from each of n given regions so that the minimum distance between any pair of points is maximized. This was dubbed the "distant representatives" problem by Fiala et al. [45], in a play-on-words on the term "distinct representatives" from Marshall Hall's classic work on bipartite matching [51]. The distant representatives problem is already hard in two dimensions for various natural regions such as discs and squares [45, 16], although there are some constant-factor approximation algorithms [20, 36, 37]. Biedl et al. [19] also give hardness results and approximation algorithms for the distant representatives problem for axis-aligned rectangles in the plane. The related topic of "imprecise points" is about choosing one point from each region (typically small balls) to optimize other aspects of the points (e.g., convex hull, spanning tree) [22, 33, 70, 69].

In one dimension, the distant representatives problem is to find one point from each of a given set of intervals on a line while maximizing the minimum distance between any two points. The decision version—to decide if there is a solution with no two points closer than a given threshold δ—was solved in polynomial time as a scheduling problem by Simons [95]. Simons' scheduling problem was to place disjoint unit jobs in given intervals. To transform the decision version of distant representatives to the scheduling problem, scale so $\delta = 1$, then expand each interval by $1/2$ on each side. The midpoints of the unit jobs provide the desired solution. This was improved to $O(n \log n)$ time by Garey et al. [46].

When the intervals are disjoint, or more generally, when the ordering of the intervals (and their representative points) is specified, we arrive at the ***dispersion problem for ordered intervals*** that is the subject of this chapter. As pointed out by Dumitrescu and Jiang [36] the dispersion problem for ordered intervals can easily be formulated as a linear program, which provides a polynomial time algorithm. Li and Wang [68] gave a linear time algorithm that processes the intervals in order, keeping a tentative solution that may change later on. They establish 9 invariants and prove (over multiple pages) that the invariants are preserved in each of three possible cases that arise when the next interval is added. Li and Wang give details only for the case of disjoint intervals; they remark that their algorithm applies to ordered intervals.

There is a body of research on lexicographic objective functions for location problems

(see [85, 73] for example), but we are not aware of such objective functions being used for dispersion problems. One classic example of lexicographic optimization in geometry is the result that the Delaunay triangulation of a point set in the plane has the lexicographic maximum angle vector [94].

3.2 The Algorithm

In this section we show that, for an input as described above, Algorithm 1 (below) solves the lexicographic dispersion problem for ordered intervals on a line.

Algorithm 1 Lexicographic Dispersion for Ordered Intervals

1. Raise each interval I_i by i units vertically to get segments $I'_i = l'_i r'_i$ with $l'_i = (l_i, i)$ and $r'_i = (r_i, i)$.
2. Construct a simple polygon P by joining the points $l'_1, l'_2, \ldots, l'_n, r'_n, r'_{n-1}, \ldots, r'_1, l'_1$ in order.
3. Find the shortest path γ in P from l'_1 to r'_n. (See Figure 3.1).
4. Declare no solution if the path is not monotone in x.
5. Otherwise output the points p_i where $p'_i = (p_i, i)$ is the intersection of the path with I'_i, $i = 1, \ldots, n$.

The algorithm transforms the dispersion problem for ordered intervals to the problem of finding a shortest path between two points in a simple polygon. See Figure 3.1. Shortest paths in a simple polygon can be found in linear time using the well-known "funnel" algorithm [66] which operates on a triangulated polygon. The simple polygon we construct is monotone—a class of simple polygons for which algorithms tend to be simpler than general simple polygons. In general, triangulating a polygon in linear time requires Chazelle's complicated algorithm [24] but any polygon we construct can be triangulated in a trivial way because it is composed of the trapezoids $l'_i, l'_{i+1}, r'_{i+1}, r'_i$ and each trapezoid can be triangulated by adding one diagonal. The funnel algorithm to find the shortest path is straightforward for the polygons that we construct—see Figure 3.2 for an illustration. Algorithm 1 thus runs in linear time and is very simple to implement.

Theorem 1. *Algorithm 1 solves the lexicographic dispersion problem for ordered intervals.*

Proof. We first note a correspondence between feasible solutions to the dispersion problem and x-monotone paths inside P from the bottom to the top of P. A feasible solution

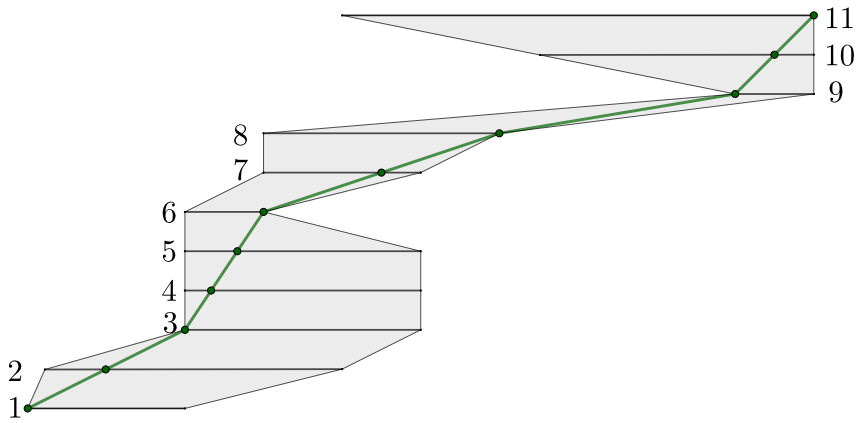


Figure 3.1: The polygon P is obtained by raising the i-th interval by i units. The thick green polygonal line is the shortest path from the left endpoint of the first interval to the right endpoint of the last interval. By Theorem 1, the x-coordinates of the intersections of this path with the raised intervals (green dots) solve the dispersion problem for ordered intervals. The minimum distance between points corresponds to the maximum slope and is realized between intervals 3, 4, 5, 6.

$\boldsymbol{p} = (p_1, \ldots, p_n)$, corresponds to the x-monotone path $\sigma(\boldsymbol{p}) = (p'_1, \ldots, p'_n)$ where $p'_i = (p_i, i)$. This path is contained in P and goes from the bottom to the top. In the other direction, given an x-monotone path σ from the bottom to the top of P, we obtain a feasible solution $\boldsymbol{p}(\sigma)$ by taking the x-coordinate of the first intersection of σ with each interval I'_i. (Observe that there is only one such intersection point if σ is a shortest path.)

Next we show that the algorithm correctly detects whether there is a feasible solution, i.e., we prove that there is a feasible solution to the dispersion problem if and only if the shortest path γ found by the algorithm is x-monotone. One direction is obvious: if γ is x-monotone then the algorithm returns the points $\boldsymbol{p}(\gamma)$ which provide a feasible solution. In the other direction, suppose there is a feasible solution $\boldsymbol{p} = (p_1, \ldots, p_n)$. Then the corresponding path $\sigma(\boldsymbol{p})$ is an x-monotone path from p'_1 to p'_n inside P. Moving p_1 to l_1 and p_n to r_n keeps the path x-monotone and inside P. After this, the path can be converted to the (unique) shortest path γ by applying local shortening moves, which preserve x-monotonicity. Thus γ is x-monotone.

It remains to prove that when the dispersion problem has a feasible solution, the algorithm returns a solution that lexicographically maximizes the sorted distance vector $\boldsymbol{\delta}(\boldsymbol{p})$. Note that altering any feasible solution by moving p_1 to l_1 and moving p_n to r_n improves the sorted distance vector. Thus it is correct to restrict attention to solutions with $p_1 = l_1$

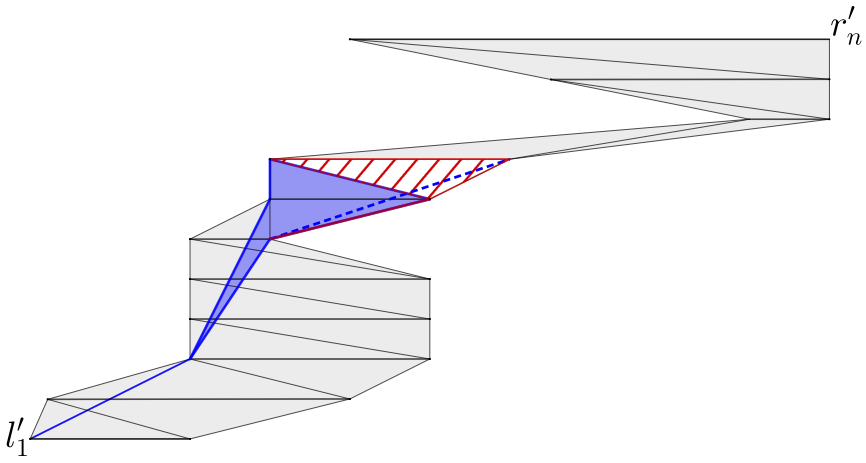


Figure 3.2: The funnel algorithm to find the shortest path from l'_1 to r'_n adds the triangles of P successively while maintaining a *funnel* (colored dark blue). To add the next triangle (red hatching), we update the funnel by backtracking along the side of the funnel until regaining a reflex curve (see the dashed blue edge). Once a polygon vertex leaves the funnel, it never returns, so the total time for updates is linear. After the funnel reaches I'_n, its right side will be the desired path.

and $p_n = r_n$.

We now translate the objective function to the geometric setting. Consider a feasible solution $\boldsymbol{p} = (p_1 = l_1, p_2, \ldots, p_n = r_n)$ and the corresponding path $\sigma(\boldsymbol{p})$. The distance, $p_{i+1} - p_i$, between two successive points corresponds inversely to the slope of the line segment from p'_i to p'_{i+1}, since the slope is $1/(p_{i+1} - p_i)$. Thus the sorted distance vector $\boldsymbol{\delta}(\boldsymbol{p})$ corresponds to the *sorted slope vector* $s(\sigma)$ that lists the $n-1$ slopes $1/(p_{i+1} - p_i)$ sorted from maximum to minimum. Lexicographically maximizing $\boldsymbol{\delta}(\boldsymbol{p})$ corresponds to lexicographically minimizing $s(\sigma)$.

Thus, it suffices to prove that the algorithm returns a solution that lexicographically minimizes $s(\sigma)$. We say that $s(\sigma)$ is *locally minimized* if moving one point p_i does not improve $s(\sigma)$. For the next claim, we assume that σ is directed away from $(l_1, 1)$.

Claim 2. *$s(\sigma)$ is at a local minimum if and only if for each i, path σ either crosses I'_i in a straight line segment, or turns left at the left endpoint (l_i, i), or turns right at the right endpoint (r_i, i).*

Proof. If σ crosses I'_i with a bend that is not of the specified form (as shown in Figure 3.3), then p'_i can be moved along I'_i in the direction of the convex angle $p'_{i-1}p'_i, p'_{i+1}$ to improve

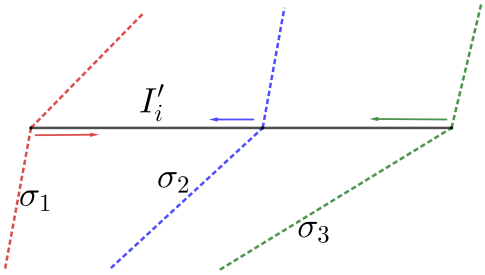


Figure 3.3: Illustration for Claim 2, showing paths that are not locally optimal.

the sorted slope vector. The vector is improved because the larger slope in the vector is reduced while the smaller slope increases. □

Observe that the conditions for locally minimizing $s(\sigma)$ are exactly the same as the conditions for σ to be a locally shortest path inside P from l'_1 to r'_n. Since a locally shortest path in a polygon is unique, $s(\sigma)$ has a unique local minimum. Furthermore, by finding the shortest path in P, the algorithm finds the lexicographic minimum $s(\sigma)$. □

3.3 Variants on Dispersion for Ordered Intervals

In this section we discuss some variants of dispersion for intervals. We first summarize what is known (and not known) about dispersion for unordered intervals on a line—see Section 3.3.1.

After that, we explore two variants for ordered intervals. We first use our geometric approach to give a polynomial time algorithm for lexicographic dispersion for ordered sets where each set consists of multiple intervals—see Section 3.3.2. The motivation for this is the case of intervals on a closed curve (arcs of a circle). Max-min dispersion for *disjoint* intervals on a closed curve was solved previously in linear time by Li and Wang [68]. We use our geometric approach to give a simpler solution for the stronger lexicographic dispersion—see Section 3.3.3.1. There was no previous algorithm for dispersion for ordered *overlapping* intervals on a curve—the issue is that cutting the curve may cut intervals into two pieces. But we can use our results on multiple intervals to solve (in Section 3.3.3.2) the lexicographic dispersion for ordered overlapping intervals on a curve in polynomial time. Section 3.3.4 considers the case where the distances between consecutive pairs of points are weighted.

3.3.1 Unordered Intervals.

Until now, we have focused on a somewhat restricted version of dispersion for intervals where the ordering of intervals is given as part of the input. (Note that if the intervals are disjoint, the ordering is determined.) In this section we mention what is known about optimal dispersion for intervals when the ordering of the intervals is not specified. As noted by Li and Wang [68], if no interval is contained in another, then the optimal ordering of the intervals for max-min dispersion can be achieved by ordering the intervals by their left endpoints, which is justified as follows. Suppose an optimum solution has $p_i < p_{i+1}$ but $l_i > l_{i+1}$. Since interval i is not contained in interval $i+1$, we have $r_i \geq r_{i+1}$. Thus $l_{i+1} < l_i \leq p_i < p_{i+1} \leq r_{i+1} \leq r_i$, and both points are contained in both intervals. Exchange the assignment of these two points to their intervals.

In the general case when the ordering of intervals is not specified and intervals may be nested, then the best algorithm for max-min dispersion that we are aware of is to use the $O(n \log n)$ decision procedure due to Garey et al. [46] (formulated as a scheduling problem) and do a binary search on the possible distance values. This gives a polynomial time (though not strongly polynomial time) algorithm. For an inefficient (although strongly polynomial-time) algorithm, observe that δ can only take a discrete set of values defined by the intervals. Specifically, the minimum δ (maximum slope) is defined by the right endpoint of an interval and the left endpoint of an earlier interval divided by the number of intervals between them. Thus there are only $O(n^3)$ different values that δ can take, and we can use the algorithm of Garey et al. [46] with the possible set of values to find the maximum δ. Also see Biedl et al. [19] for further discussion.

We do not know an algorithm for the lexicographic dispersion problem on general unordered intervals.

3.3.2 Ordered Sets of Multiple Intervals on a Line.

In this section we consider a more general dispersion problem, where we are given ordered sets $S_1, \ldots, S_n$ and each S_i consists of a finite set of intervals on the line. We use our geometric approach to solve this more general lexicographic dispersion problem. Specifically, we give a polynomial time algorithm to find points $p_1 \leq p_2 \cdots \leq p_n$ with $p_i \in S_i$ so as to lexicographically maximize their sorted distance vector.

Even with the weaker max-min objective function, this problem is not amenable to a linear programming formulation—it is not clear how to specify that a point can be in one of several intervals. When the ordering of the S_i's is not specified then the decision version

of the problem is NP-complete, as shown by Simons and Sipser [96], who viewed it as a scheduling problem. We are not aware of any previous algorithm for the ordered version of the problem, but our geometric viewpoint offers a simple solution.

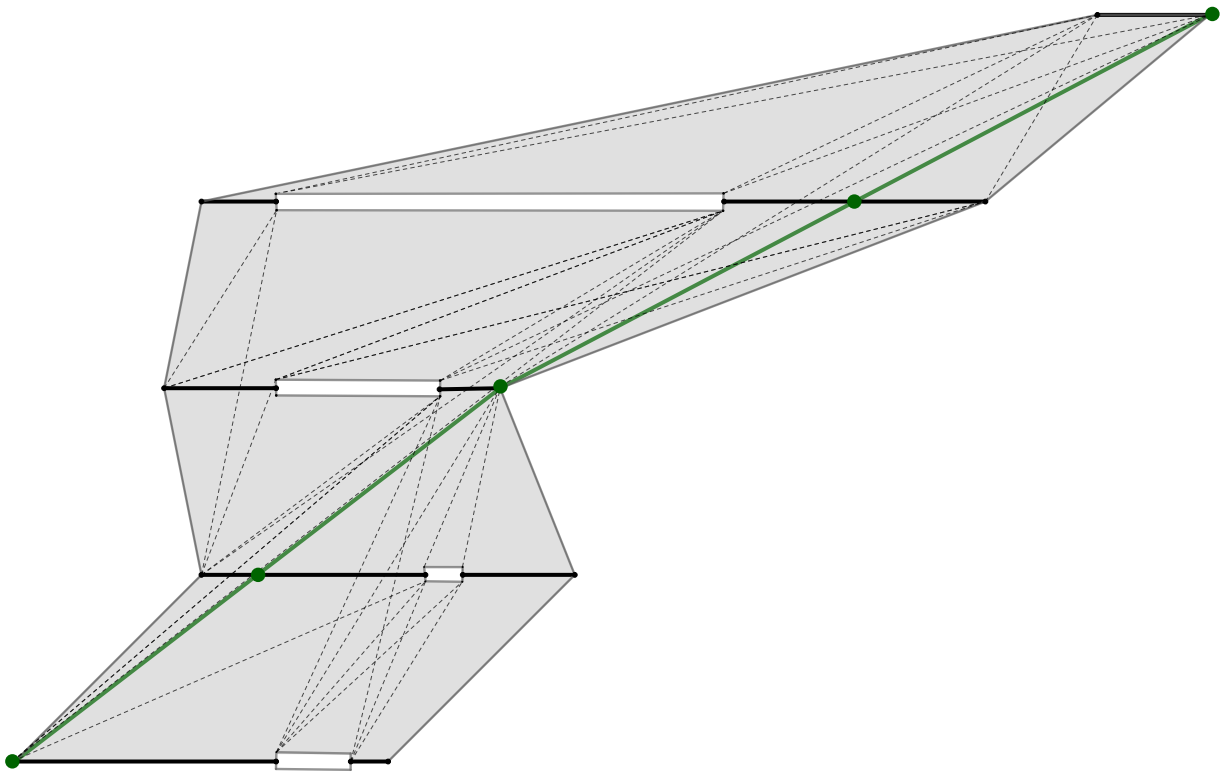


Figure 3.4: Solving lexicographic dispersion for multiple intervals by finding a path of lexicographically minimum slope vector (shown thick green) in the visibility graph (dashed edges) of a polygonal region. Gaps between intervals at the same level are shown as thin rectangles. The intersections of the path with the intervals (green dots) give the solution points.

The idea is to raise each set S_i by i units, and to create a polygonal region R with holes that represent the gaps between the intervals of each S_i. See Figure 3.4. Let l_1 be the leftmost point of S_1, and let r_n be the rightmost point of S_n. We claim that a solution is given by an xy-monotone path γ that lies inside the region R and goes from the point $(l_1, 1)$ to the point (r_n, n). Among all such paths, we want the one that lexicographically minimizes the slope vector of the path.

As before, a path has a locally optimal slope vector if and only if it is a locally shortest

Algorithm 2 Lexicographic Dispersion for Ordered Sets of Intervals

1. Raise the intervals of each set S_i by i units vertically to obtain S'_i
2. Construct a polygonal region R around the segments S'_i (see Figure 3.4)
3. Construct the monotone visibility graph $G = (V, E)$ of R
4. Find a path of lexicographically minimum slope vector in G from source $(l_1, 1)$ (the leftmost point of S'_1) to the rightmost point of S'_n as follows:
5. Initialize the slope vector $s(v)$ for every vertex $v \in V$
6. **while** $V \neq \phi$
7. Pick a source vertex u
8. **for every** edge (u, v)
9. $s'(v) :=$ the slope vector determined by $s(u)$ together with the edge (u, v)
10. **if** $s'(v) < s(v)$ **then** update $s(v)$
11. Remove u from V
12. If a path is found, then return the intersection points of the path with each S'_i
13. Otherwise, return that no solution exists.

path. However, locally shortest paths are not unique in a polygonal region, and the shortest path does not necessarily optimize the slope vector, so we must deal with the objective function more directly.

Our approach is to search for the optimal path in the ***monotone visibility graph*** of the polygonal region. Specifically, we construct a directed graph $G = (V, E)$ on the vertices of R, with a directed edge (u, v) if the line segment uv is contained in the region R and v lies above and to the right of u. Note that G is a DAG (a directed acyclic graph) with $2N$ vertices and $O(N^2)$ edges, where N is the number of intervals in all the S_i's. Graph G can be constructed in time $O(N \log N + k)$ where k is the number of edges in the (entire) visibility graph [48]. We seek a directed path in G from the source vertex $(l_1, 1)$ to (r_n, n) that lexicographically minimizes the slope vector.

We modify the standard linear time algorithm to find optimum paths from the source vertex of a DAG. See Steps 5–11 of Algorithm 2. A vertex v corresponding to an endpoint of an interval of S_i that lies at y-coordinate i, and the slope vector of a path from $(l_1, 1)$ to v has $i - 1$ components. Let $s(v)$ be the (tentative) lexicographically minimum slope vector of a path from $(l_1, 1)$ to v. To initialize, set all $i - 1$ components of $s(v)$ to ∞.

For line 9, suppose that u lies at level i and v lies at level $i + h$. Let s_{uv} be the slope of edge (u, v). Then $s'(v)$ is formed by adding h copies of s_{uv} into the sorted vector $s(u)$. This can be done in $O(n)$ time, and $s'(v)$ can be compared to $s(v)$ in $O(n)$ time. Thus finding the shortest path takes time $O(|E|n)$.

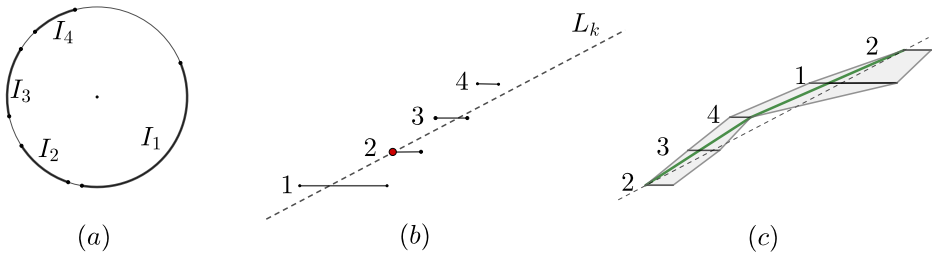


Figure 3.5: Solving the lexicographic dispersion problem for disjoint arcs on a circle. (a) the input; (b) the line L_k of slope n/D through the extreme left endpoint (on interval I_2); (c) the shortest path in the resulting polygon.

The total runtime of the algorithm is $O(N^2 n)$ in the worst case (recall that N is the number of intervals in all the sets S_i). We note that the same idea solves max-min dispersion for ordered sets of intervals in $O(N^2)$ time. These run times can probably be improved by exploiting the special structure of the polygonal regions we construct.

Theorem 3. *There is a polynomial-time algorithm that solves the lexicographic dispersion problem for ordered sets of multiple intervals on a line.*

3.3.3 Intervals on a Closed Curve.

In this section we consider the case of intervals on a closed curve in the plane, rather than on a line. See Figure 3.5(a). The input consists of an ordered list of n intervals $I_1, \ldots, I_n$ on a curve of length D. We assume that interval I_i is the clockwise interval from point l_i to r_i, and that points are given by their clockwise distance along the curve from the reference point l_1. A feasible solution for this dispersion problem consists of points $\boldsymbol{p} = (p_1, \ldots, p_n)$ in clockwise order around the curve, with $p_i \in I_i$. These points determine n gaps between successive pairs, namely p_i, p_{i+1}, $i = 1, \ldots, n-1$ and p_n, p_1. The length of a gap is measured along the curve (so their sum is D). The sorted distance vector $\boldsymbol{\delta}(\boldsymbol{p})$ is the list of n gap lengths sorted from minimum to maximum. The max-min and lexicographic objective functions are defined in the natural way.

There are solutions for max-min dispersion for *disjoint* intervals on a curve. Dumitrescu and Jiang [36] gave a linear programming formulation—they only need to add one additional constraint for the distance between p_n and p_1 along the curve. Li and Wang [68] gave a linear time algorithm by reducing to the case of intervals on a line. It is important

to note that these solutions do not apply to the case of overlapping intervals along a curve, even if the ordering is specified, because cutting the curve may split intervals in two.

We use our geometric approach to solve the lexicographic dispersion problem for disjoint intervals in linear time. Then we solve the lexicographic dispersion problem for overlapping intervals in polynomial time using our algorithm from Section 3.3.2.

3.3.3.1 Linear Time for Disjoint Intervals.

In this section we give a linear time algorithm for lexicographic dispersion for disjoint intervals on a closed curve.

We begin by describing Li and Wang's algorithm for max-min dispersion in the same setting. They traverse the curve twice starting from l_1 to create a list of $2n$ intervals on a line segment of length $2D$. They apply their algorithm for intervals on a line to these $2n$ intervals. They prove that the solution uses the left endpoint of some "special" interval I_j, and that the subsequence of the solution from interval I_j to its second copy solves the original problem.

Our solution is similar, but the advantage of the geometric approach is that it provides a simpler and more intuitive rule for choosing the special interval, and we can optimize the stronger lexicographic objective function.

Observe that the best solution we can hope for is a uniform spacing of points, so the length of each gap is D/n. In the geometric interpretation, where interval I_i is raised to y-coordinate i, such a solution corresponds to a line L_{opt} of slope n/D that intersects all the segments. We would need all left endpoints (l_i, i) to the left of L_{opt} and all right endpoints (r_i, i) to the right of L_{opt}.

Algorithm 3 Lexicographic Dispersion for Disjoint Intervals on a Curve

1. $k := \text{argmax}\{nl_i - Di : i = 1, \ldots, n\}$ (breaking ties arbitrarily)
2. Recompute left and right points of each interval as clockwise distances from l_k and then renumber the intervals starting from I_k
3. Raise each interval I_i by i units vertically to get segments I'_i
4. Add a final interval I'_{n+1} that is a copy of I_1 translated D units to the right and $n+1$ units up
5. Create the polygon P around the intervals I'_i (as in Algorithm 1)
6. Find the shortest path in P from the leftmost point of I'_1 to the leftmost point of I'_{n+1}
7. Return the intersection points of the path with I'_i

This motivates finding the line L of slope n/D that goes through the most *extreme* left endpoint, i.e., such that L goes through a left endpoint (l_k, k) and such that no other left endpoint lies to the right of L. See Figure 3.5(b). The vector orthogonal to (D, n) is $(n, -D)$ so L is found by maximizing $nx - Dy$ over left endpoints $(x, y) = (l_i, i)$. Suppose the maximum occurs at (l_k, k) and let L_k be the line L through this point. See Step 1 of Algorithm 3.

Claim 4. *There is an optimum solution through the point* (l_k, k).

Proof. Consider an optimum solution that does not contain (l_k, k), and let L be the rightmost line of slope n/D that intersects some point in this optimum solution. Line L must lie on or to the right of the solution point on I'_k, hence strictly to the right of L_k. If all points of the solution lie on L then L intersects all intervals. In this case, L_k also intersects all intervals, and the claim holds. Otherwise, not all solution points lie on L. We will show that the solution was not optimum. Choose a solution point (p_j, j) on L such that one of its neighbours p_{j-1} or p_{j+1} does not lie on L (note that such a point exists). Then the path turns left at (p_j, j). Since no left endpoint of an interval lies to the right of L_k, the point (p_j, j) is not at the left endpoint of I'_j. This violates Claim 2 for a locally optimum path. □

The algorithm then breaks the cyclic order of intervals at I_k (see Step 2). After this re-numbering, the left endpoint of I_1 is guaranteed to be part of an optimum solution, and we can apply the same idea that was used in Algorithm 1. The one difference is that we add an extra copy of I_1 (Step 4) so we can force the path to go through the left endpoint of I_1. See Figure 3.5(c).

Algorithm 3 clearly runs in linear time.

Theorem 5. *There is a linear-time algorithm that solves the lexicographic dispersion problem for disjoint intervals on a curve.*

3.3.3.2 Overlapping Intervals.

In this section we give a polynomial time algorithm for lexicographic dispersion for (possibly) overlapping intervals on a closed curve. Any optimal solution can be shifted counterclockwise until one point hits the left end of its interval. This means that it suffices to find the best solution using the left endpoint of each of the intervals; the final answer is the lexicographic optimum among these n solutions.

To find the optimum solution using the left endpoint l_i of interval I_i, we break the cyclic order of intervals at I_i and compute left and right endpoints as clockwise distances from reference point l_i. Any interval I_j that includes the point l_i is now broken into two intervals, $(0, l_j)$ and (r_j, D). We apply the idea of Algorithm 2 from the previous section. Create two copies of I_i, one at the start, and one (shifted D units right) at the end, in order to find a path using l_i. Then apply Algorithm 2 to find the optimum solution. The worst case runtime is $O(n^4)$. This can no doubt be improved.

Theorem 6. *There is a polynomial-time algorithm that solves the lexicographic dispersion problem for intervals on a curve.*

3.3.4 Weighted Distances.

Our algorithm extends to the case when the distances between successive pairs are weighted. For example, in our original motivating scenario of people maintaining social distance in a line, the neighbours of a person who is coughing may wish to keep twice as far away, so we would apply a larger weight to the two incident distances. Let $w_{i,i+1} \geq 1$ be the weight that applies to the distance $p_{i+1} - p_i$. Then the objective function is to lexicographically maximize the vector with values $\frac{1}{w_{i,i+1}}(p_{i+1} - p_i)$ sorted from minimum to maximum. To solve this weighted lexicographic dispersion problem, we set the vertical gaps between raised intervals I'_i and I'_{i+1} to $w_{i,i+1}$. The slope of the line segment from the raised point p'_i to p'_{i+1} is then $w_{i,i+1}/(p_{i+1} - p_i)$ so maximizing the distance vector is the same as minimizing the slope vector, and the shortest path solution works.

3.4 Conclusions

In Section 3.2, we described a simple linear time algorithm to solve the dispersion problem for ordered intervals on a line. Our algorithm finds the lexicographically maximum distance vector, i.e., it finds a Pareto optimal solution where no point can move to improve its distances from its two neighbours.

Open Problems. Araki et al. [7] consider the problem of finding k facilities on n (disjoint and unsorted) intervals on a line such that the distance between the closest pair is maximized (also see Araki and Nakano [8]). They give an $O((4k^2)^k n)$ time algorithm based on a greedy approach. The authors consider it a linear time algorithm since k is assumed

to be a fixed constant. By considering k copies of the n intervals as in Section 3.3.2, our method gives an $O(n^2k^2)$ time algorithm for the same problem. The advantage here is that the execution time is polynomial in k and n. We leave open the possibility of faster algorithms (say, an $O(n + k)$ (possibly with polylogarithmic factors) time algorithm) for this problem.

Chapter 4

Center Problems and Farthest-site Voronoi diagrams

> Pick a flower on Earth and you move the farthest star
>
> Paul Dirac

In Chapter 1, we informally introduced a class of facility location problems called center problems: problems that take as input a set S of sites and output the point that minimizes, under some metric, the distance to the farthest site from S. The simplest problem in this class is when the set S consists of points in the plane, and the distances are Euclidean distances (Sylvester [98]).

This chapter first discusses some background on center problems (Section 4.1). Since they are closely related to farthest Voronoi diagrams (defined in Chapter 2), we will discuss both. We then describe two techniques that are indispensable in the solution of center problems—prune-and-search (in Section 4.2), and geometric divide-and-conquer using ϵ-nets (in Section 4.3). We will use both of these techniques for our solutions to the visibility center problem (Chapter 5) and the geodesic edge center problem (Chapter 6).

4.1 Background

The problem of Sylvester can be solved using the farthest point Voronoi diagram—the center is a vertex of the Voronoi diagram unless the center has only two farthest points,

in which case it lies on an edge of the Voronoi diagram. Shamos and Hoey [92] gave an $O(n \log n)$ time algorithm to find the farthest point Voronoi diagram and the center. There is an $\Omega(n \log n)$ lower bound for finding Voronoi diagrams of points in the plane, but the center problem is strictly easier—Megiddo [78] used his "prune-and-search" technique to give a linear time algorithm to find the center of a set of points in the plane. Almost all the known algorithms for geodesic center problems end up using Megiddo's prune-and-search techniques in the last stages. Since Megiddo's ideas are crucial to our algorithms, we outline his technique for solving Sylvester's problem in Section 4.2.

The problems we address in Chapter 5 and Chapter 6 use distances that are geodesic rather that Euclidean, and sites that are segments (edges) rather than points. Although that variant had not been addressed prior to this work, the two other possibilities—segments in the plane and points in a polygon—are well-studied, as we briefly describe.

For Euclidean distances, Megiddo's method extends to linear time algorithms to find the center of line segments or lines in the plane [17], a problem called the "intersection radius problem" since the center point is the center of a minimum radius disc that intersects the given sites. The farthest Voronoi diagram of segments in the plane was considered by Aurenhammer et al. [13] who called it a "stepchild in the vast Voronoi diagram literature". They gave an $O(n \log n)$ time algorithm which was improved to output-sensitive time $O(n \log h)$, where h is the number of faces of the diagram [90].

There are algorithms to find the Euclidean center of more general sites such as convex polygons [60], and to find the smallest disc that *contains* a set of discs [80] which can be viewed as a center problem for points with additive weights, and is one of the ingredients in the algorithms for finding the vertex center of a polygon. With regard to Voronoi diagrams, there is an algorithm to find the farthest Voronoi diagram of disjoint polygons in the plane [28], and more generalizations can be found in the book by Aurenhammer et al. [14].

We now turn to geodesic distances, defined in Chapter 2. Ahn et al. [4] gave a linear time algorithm to find the geodesic center of the vertices of a polygon [4]. The farthest Voronoi diagram of the vertices of a polygon can be found in time $O(n \log \log n)$ [87], and in expected linear time [15]. This beats the $\Theta(n \log n)$ bound for points in the plane with Euclidean distances, but does not (yet) match the linear time bound for finding the geodesic vertex center. More generally, for m points inside an n-vertex polygon, an algorithm to find their farthest Voronoi diagram was first given by Aronov et al. [10] with run-time $O((n+m)\log(n+m))$, and improved in a sequence of papers [87, 15, 86], culminating in an optimal run time of $O(n + m \log m)$ [102]. For the problem of finding the center of m points in a simple polygon, we are not aware of any algorithm that beats this bound.

There is little prior work on geodesic centers of sites other than point sites. The remaining part of this thesis makes a few contributions in this direction:

1. We give an $O((n+m)\log(n+m))$ time algorithm to find the geodesic center of m *half-polygons* or chords inside a polygon [72] (described in detail in Chapter 5).

   Note that the geodesic center of the edges of a simple polygon is a special case of the half-polygon problem, and our algorithm directly yields an $O(n \log n)$ time algorithm to find the geodesic center of the edges of a polygon.

2. We give an optimal linear-time algorithm for the geodesic center of the edges of a simple polygon. This generalizes the algorithm to find the geodesic center of the vertices (solved by Ahn et al. in linear time).

Finally, we mention a curious difference between nearest/farthest site Voronoi diagrams of edges in a polygon. One of the most famous and useful Voronoi diagrams is the medial axis, which is the nearest Voronoi diagram of the edges of a polygon. The medial axis has a host of applications, to shape and character recognition, motion planning, etc. [58], as well as for computing (nearest) Voronoi diagrams of more complicated sites [5]. However, the *farthest* Voronoi diagram of edges in a polygon has received virtually no attention. The one exception we are aware of is for the case of convex polygons. Aurenhammer et al. [13] conjectured a linear time algorithm for this case, since the other three variants—the medial axis (the nearest edge Voronoi diagram), the nearest vertex Voronoi diagram, and the farthest vertex Voronoi diagram can all be computed in linear time for a convex polygon (Aggarwal et al. [2]). Drysdale and Mukhopadhyay [35] gave an $O(n \log n)$ time algorithm to find the farthest edge Voronoi diagram in a convex polygon. Khramtcova and Papadopoulou [62] gave an expected linear time algorithm to find the Euclidean farthest segment Voronoi diagram of segments in convex position.

4.2 Megiddo's Prune-and-Search Paradigm

We give an outline of the prune-and-search paradigm introduced by Megiddo [79]. Although the method was introduced for solving linear programming problems in linear-time for fixed dimensions, it is also an important tool for many types of center problems. We describe the approach since it is used repeatedly in Chapters 5 and 6. The basic structure of such problems is the following: Given a set S of input sites in $\mathbb{R}^d$ find a disk of smallest radius that intersects/contains all the sites. This is equivalent to searching for a point in $\mathbb{R}^d$ from which the maximum distance to a site is minimized.

We explain the prune-and-search method for the minimum enclosing ball problem for point sites. For any point $x \in \mathbb{R}^d$, define the radius function as: $r(x) := \max_{s \in S} d(x, s)$. We wish to locate a point $x^* \in \mathbb{R}^d$ where the radius function is minimized. As the distance to a point is a convex function, the radius function (being the maximum of convex functions) is also convex. Furthermore, we can find a set S^* with at most $d + 1$ points from S that defines the minimum enclosing ball, such that the minimum enclosing ball of $S^* \setminus \{s\}$ for $s \in S^*$ has a strictly smaller radius [78]. The points of S not in S^* may be removed without changing the center and radius of the minimum enclosing ball. Therefore, the points in $S \setminus S^*$ are irrelevant constraints with respect to the minimum enclosing ball problem. The prune-and-search method systematically removes points from S that lie in $S \setminus S^*$.

Hyperplane oracles. The most important component of Megiddo's algorithm is the ***hyperplane oracle***. A hyperplane oracle for the minimum enclosing ball problem takes as input a hyperplane H in $\mathbb{R}^d$, and does the following:

1. Finds a point x^*_H on H that minimizes the radius function.
2. Determines if $x^* = x^*_H$.
3. If $x^* \neq x^*_H$, finds the halfspace defined by H that contains x^*.

Notice that the oracle must first solve the problem of finding the point of minimum radius restricted to a space of one lower dimension. Due to the convexity of the function whose value we are trying to minimize, it can be shown that Steps 2 and 3 of the oracle are equivalent to solving two additional versions of the problem restricted to the lower dimension [38, 78]. Basically, we solve the problem on hyperplanes H^+ and H^- parallel to H and at a small distance on either side of it. Each of these solutions gives a value for the minimum radius function and comparing these values tells us whether $x^* = x^*_H$, and if not, the halfspace containing the global minimum.

Bisecting hyperplanes. We can determine a bisecting hyperplane $H(a, b)$ for any two points a and b in S, a plane consisting of points equidistant from a and b. If the hyperplane oracle for $H = H(a, b)$ determines that $x^* = x^*_H$, we determine the center of the minimum enclosing ball. If $x^* \neq x^*_H$, however, it allows us to prune away one of the two points a or b. This is because in the halfspace defined by the bisecting hyperplane that contains the point x^*, the distance to one of a or b is strictly lesser than the other. Since the center lies on this side of the bisecting hyperplane, one of a or b must lie in $S \setminus S^*$ and is irrelevant. This point may be pruned away.

Megiddo's Algorithm. Running the hyperplane oracle on a bisecting plane allows us to prune away one point site from S. It is quite inefficient to run the oracle on each bisecting plane. Megiddo shows how to prune away a 'large' number of irrelevant points using only a 'few' (constant for any fixed dimension) applications of the hyperplane oracle. Specifically, for a fixed dimension d, Megiddo shows how to run the oracle on the hyperplanes in a constant-sized set Q to prune away a constant fraction of points in S.

- While the size of S is greater than some fixed constant c:
  1. Arbitrarily pair up the points in S
  2. Determine the set H_B containing bisecting hyperplanes for each pair
  3. Run the hyperplane oracle on a set Q (where $|Q| = s(d)$ independent of n) to narrow down the location of x^* to a region C in space that intersects at most a constant fraction $f(d)$ (independent of n) of hyperplanes from the set H_B
  4. For each bisecting plane of H_B that does not intersect C, remove a corresponding irrelevant point from S
- Solve the center problem directly for a problem of size c

Since the region C avoids at least a constant fraction of the bisecting hyperplanes, one point from the pair corresponding to each such bisecting hyperplane is irrelevant at x^* and may be pruned. To obtain such a region, we may use the original methods of Megiddo [78], Dyer [40], or more modern methods using cuttings defined in Section 4.3 (See Lemma 9). Thus the size of the problem is reduced by a constant fraction.

The recurrence for the runtime is:

$$T(n,d) \leq 3s(d)T(n-1,d) + T(\tfrac{1+f(d)}{2}n, d) + O(nd)$$

The above recurrence solves to $T(n,d) < K2^{2^d}n$ where K is an ablsolute constant independent of both d and n. Thus, the time taken by the algorithm to determine the center is linear in n. Although the dependence on d is doubly exponential, it is constant for any fixed d.

Generalizing to other center problems. The prune and search paradigm may be generalized to many other types of center problems. Some examples include intersection radius problems [17] and finding the smallest ball that contains a set of balls [80].

For prune-and-search to be applicable, the distance function to the sites must be a convex function, and a constant number of sites define the final solution (in the plane, three points defined the minimum enclosing ball). We must find a point to minimize the distance to the farthest site. The hyperplane oracle is defined analogously for the problem—given a hyperplane H, we solve a restricted version of the problem on H before determining the halfspace defined by H that contains the optimum. We also need the bisector between any pair of sites to be a hyperplane (this sometimes requires a non-trivial transformation of the problem [80, 40]). Once we have a set of bisecting hyperplanes, the same ideas as before (running the oracle on a constant number of hyperplanes to prune a constant fraction of irrelevant input sites) provide a solution to the center problem being solved.

4.3 Overview of ϵ-Nets

Next, we give a brief overview of ϵ-nets and their use in geometric divide-and-conquer algorithms. For more details, we refer to the paper by Haussler and Welzl [54], the survey by Mustafa and Varadarajan [99, Chapter 47], and the book by Mustafa [84].

A ***range space*** is a pair $(X, \mathcal{R})$ where X is a ***ground set*** of elements and $\mathcal{R}$ is a set of subsets of X. We refer to the elements of $\mathcal{R}$ as the ***ranges*** of the range space. For any ϵ between 0 and 1, an ***ϵ-net*** for the range space $(X, \mathcal{R})$ is a subset $N \subseteq X$ with the following property: for every range R in $\mathcal{R}$ with $|R| \geq \epsilon|X|$, we have $N \cap R \neq \phi$. We use this as:

$$\text{if } N \cap R = \phi, \text{ then } |R| < \epsilon|X| \tag{4.1}$$

The size of the ϵ-net directly controls the number of subproblems in the divide and conquer algorithm. Efficient algorithms using this approach require an ϵ-net of small size. One way to guarantee constant sized ϵ-nets is using combinatorial properties like the VC-dimension or shattering dimension.

Consider the range space $(X, \mathcal{R})$. For a set $A \subseteq X$, the ***restriction of the ranges*** to A, denoted $\boldsymbol{\mathcal{R}_{|A}}$, is defined to be $\{A \cap R : R \in \mathcal{R}\}$. A set A is ***shattered*** by the range space $(X, \mathcal{R})$ if $\mathcal{R}_{|A}$ is the power set of A. The ***VC-dimension of a range space*** $(X, \mathcal{R})$ is the maximum size of a set that can be shattered by the range space. If a range space can shatter sets of arbitrarily large size, it has infinite VC-dimension.

The ***shattering dimension*** of the range space $(X, \mathcal{R})$ is the minimum number d such that for all m and for all sets $A \subseteq X$ with $|A| = m$, we have $|\mathcal{R}_{|A}| \in O(m^d)$. Equivalently,

this says that the number of ranges when restricted to any subset of size m is upper bounded by a polynomial in m of degree equal to the shattering dimension. Usually, upper bounds for the shattering dimension can be found more readily than those for the VC-dimension, and upper bounds on the shattering dimension imply upper bounds on the VC-dimension, as expressed by the following restatement of Lemma 5.14 from Har-Peled [52]:

Lemma 7. *If a range space has shattering dimension d, its VC-dimension is bounded by $O(d \log d)$, specifically by $12d \ln (6d)$.*

In the next lemma, we state the result of Haussler and Welzl [54] that a range space $(X, \mathcal{R})$ of finite VC dimension has constant-sized ϵ-nets. For a divide and conquer algorithm we also need an algorithm to *find* an ϵ-net of constant size. Such a deterministic algorithm[1] was given by Matousek [76] for any range space of shattering dimension d that has a ***subspace oracle*** which is defined to be a deterministic algorithm that, given a subset $X' \subseteq X$, computes the set $\mathcal{R}_{|X'}$ in time $O(|X'|^{(d+1)})$.

We summarize the results of Haussler and Welzl [54] and Matousek [74] in the following lemma. Other sources for these results include the survey by Mustafa and Varadarajan [99, Chapter 47, Theorem 47.4.3], and the textbook by Mulmuley [83].

Lemma 8. *A range space $(X, \mathcal{R})$ of finite VC-dimension has ϵ-nets of size $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$. Furthermore, if the range space has a subspace oracle then such an ϵ-net can be found in deterministic time $O(|X|)$.*

In many geometric settings, the ground set consists of hyperplanes. In such cases, an ϵ-net N determines a hyperplane arrangement that partitions the space and suggests a divide and conquer approach based on the cells of this partition, called a ***cutting*** [25, 75]. For cuttings, we have the following lemma(See Mustafa and Varadarajan [99, Chapter 47, Theorem 47.4.3]):

Lemma 9. *For any given set of n hyperplanes in $\mathbb{R}^d$, there exists a partition of $\mathbb{R}^d$ into $O(r^d)$ interior disjoint simplices such that the interior of each simplex intersects at most $\frac{n}{r}$ hyperplanes. These simplices (together with the list of hyperplanes intersecting the interior of each simplex) can be found in $O(nr^{d-1})$ time.*

[1] A randomized algorithm is easier to obtain, but we need the stronger deterministic result.

Chapter 5

The Visibility Center of a Simple Polygon[1]

> "What makes the desert beautiful," said the little prince, "is that somewhere it hides a well"
>
> *The Little Prince*,
> Antoine de Saint-Exupery

Suppose you want to guard a polygon and you have many sensors but only one guard to check on the sensors. The guard must be positioned at a point c_V in the polygon such that when any sensor sends an alarm, the guard travels from c_V on a shortest path inside the polygon to *see* the point u where the sensor sending the alarm is located; the goal is to minimize the maximum distance the guard must travel. More precisely, we must choose c_V to minimize the maximum, over points u, of the geodesic distance from c_V to a point that sees u. The optimum guard position c_V is called the ***visibility center*** of the set U of possible query points. See Figure 5.1. We give an $O((n+m)\log(n+m))$ time algorithm to find the visibility center of a set U of size m in an n-vertex simple polygon. To find the visibility center of *all* points inside a simple polygon, we can restrict our attention to the vertices of the polygon, which yields an $O(n \log n)$ time algorithm.

[1]Based on joint work with Anna Lubiw. Published in the paper 'The Visibility Center of a Simple Polygon' at the European Symposium on Algorithms (ESA), 2021 ([71]).

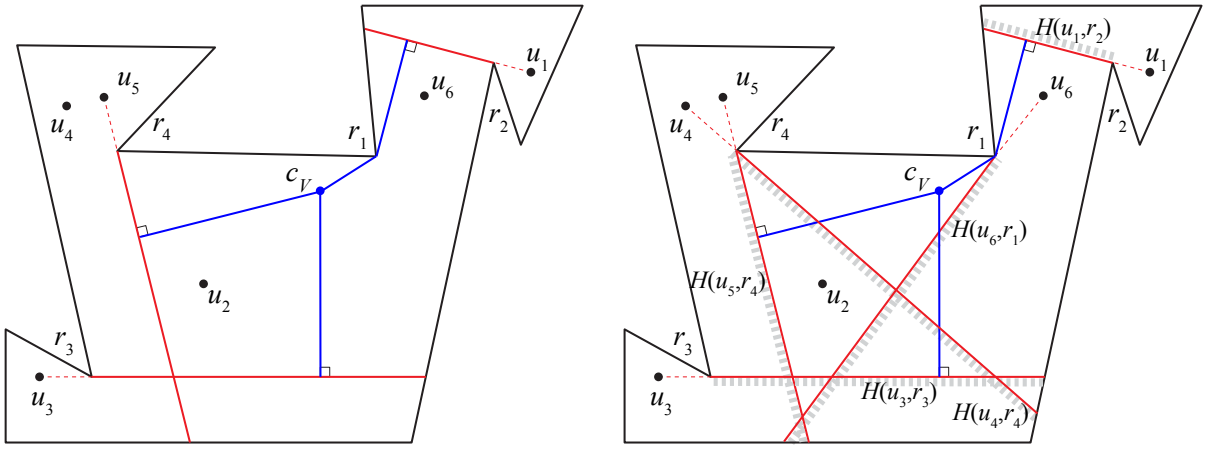


Figure 5.1: (left) Point c_V is the ***visibility center*** of points $U = \{u_1, \ldots, u_6\}$. Starting from c_V, the three points we need to travel (equally) farthest to see are u_1, u_3 and u_5. The shortest paths (in blue) to see these points must reach the half-polygons bounded by the chords (in red) emanating from the points. (right) Equivalently, c_V is the geodesic center of five half-polygons (each shown as a red boundary chord shaded on one side).

To the best of our knowledge, the idea of visibility centers is new, though it is a very natural concept that combines two significant branches of computational geometry: visibility problems [47]; and center problems and farthest Voronoi diagrams [14].

Inside a polygon the relevant distance measure is not the Euclidean distance but rather the shortest path, or *geodesic*, distance. The geodesic center of a simple polygon is a point p that minimizes the maximum geodesic distance from p to any point q of the polygon, or equivalently, the maximum geodesic distance from p to any vertex of the polygon. Pollack, Sharir, and Rote [91] gave an $O(n \log n)$ time divide-and-conquer algorithm to find the geodesic center of a polygon. Our algorithm builds on theirs. A more recent algorithm by Ahn et al. [4] finds the geodesic center of a polygon in linear time. It seems hard to extend their results to obtain more efficient results for the visibility center problem. Another notion of the center of a polygon is the so-called link center, which can be found in $O(n \log n)$ time [32].

Our Results. The ***distance to visibility*** from a point x to point u in P, denoted $d_V(x, u)$ is the minimum distance in P from x to a point y such that y sees u. For a set of points U in P, the ***visibility radius*** of x with respect to U is $r_V(x, U) := \max\{d_V(x, u) : u \in U\}$. The ***visibility center*** c_V of U is a point x that minimizes $r_V(x, U)$. Our main

result is:

Theorem 10. *There is an algorithm to find the visibility center of a point set U of size m in a simple n-vertex polygon P with run-time $O((n+m)\log(n+m))$.*

It is an open problem to find visibility centers in other domains (say, polyhedral terrains or polygons with holes).

The key to our algorithm is to reformulate the visibility center problem in terms of distances to certain *half-polygons* inside the polygon. We illustrate the idea by means of the example in Figure 5.1 where the visibility center of the 6-element point set U is the *geodesic center* of a set of five half-polygons.

More generally, we will reduce the problem of finding the visibility center to the problem of finding a geodesic center of a linear number of half-polygons. The input to this problem is a set $\mathcal{H}$ of k half-polygons (see Section 5.1 for precise definitions) and the goal is to find a *geodesic center* c that minimizes the maximum distance from c to a half-polygon. More precisely, the ***geodesic radius*** from a point x to $\mathcal{H}$ is $r(x,\mathcal{H}) := \max\{d(x,H) : H \in \mathcal{H}\}$, and the ***geodesic center*** c of $\mathcal{H}$ is a point x that minimizes $r(x,\mathcal{H})$. Our second main result is:

Theorem 11. *There is an algorithm to find the geodesic center of a set $\mathcal{H}$ of k half-polygons in a simple n-vertex polygon P with run-time $O((n+k)\log(n+k))$.*

Our algorithm extends the divide-and-conquer approach that Pollack et al. [91] used to compute the geodesic center of the vertices of a simple polygon.

Our main motivation for finding the geodesic center of half-polygons is to find the visibility center, but the geodesic center of half-polygons is of independent interest. Euclidean problems of a similar flavour are to find the center (or the farthest Voronoi diagram) of line segments or convex polygons in the plane [17, 60]. These problems are less well-studied than the case of point sites (e.g., see [13] for remarks on this). The literature for geodesic centers is even more sparse, focusing almost exclusively on geodesic centers of points in a polygon. It is thus interesting that the center of half-polygons inside a polygon can be found efficiently. As a special case, we can find the geodesic center of the edges of a simple polygon in $O(n \log n)$ time.

For more background on center problems, please see Chapter 4.

There are algorithms for the "quickest visibility problem"—to find the shortest path from point s to see point q, and to solve the query version where s is fixed and q is a query

point [9, 101]. For a simple polygon [9], the preprocessing time and space are $O(n)$ and the query time is $O(\log n)$. We do not use these results in our algorithm to find the visibility center c_V, but they are useful afterwards to find the actual shortest path from c_V to see a query point.

A more basic version of our problem is to find, if there is one, a point that sees all points in U. The set of such points is the ***kernel*** of U. When U is the set of vertices, the kernel can be found in linear time [65]. For a general set U of size m, Ke and O'Rourke [61] gave an $O(n + m \log(n + m))$ time algorithm to compute the kernel, and we use some of their results in our algorithm.

Another problem somewhat similar to the visibility center problem is the watchman problem [30, 34]—to find a minimum length tour from which a single guard can see the whole polygon. Our first step is similar in flavour to the first step for the watchman problem, namely, to replace the condition of "seeing" everything by a condition of visiting certain "essential chords".

Background on visibility can be found in Chapter 2. The reduction from the visibility center problem to the geodesic center of half-polygons is in Section 5.2. The run time is $O((n+m)\log(n+m))$. The algorithm that proves Theorem 11 (finding the geodesic center of half-polygons) is in Section 5.3. Together these prove Theorem 10 (finding the visibility center).

5.1 Preliminaries

We refer to Chapter 2 for an introduction to standard notions from computational geometry.

Any chord in P divides it into two weakly simple half-polygons. A half-polygon will be specified by its defining chord (p, q) with the convention that the half-polygon contains the path clockwise from p to q. For half-polygon H, the ***geodesic distance*** $\boldsymbol{d(x, H)}$ is the minimum distance from x to a point in H.

The ***distance to visibility*** from x to u, denoted $d_V(x, u)$ is the minimum distance from x to a point y such that y sees u. If x sees u, then this distance is 0.

If x does not see u, the distance to visibility is the distance from x to the half-polygon defined as follows. Let r be the last reflex vertex on the shortest path from x to u. Extend the ray $\overrightarrow{ur}$ from r until it hits the polygon boundary ∂P at a point p to obtain a chord rp (an edge of the visibility polygon of u). Of the two half-polygons defined by rp, let $H(u, r)$ be the one that contains u. See Figure 5.1. For such points x, we have:

Observation 12. $d_V(x, u) = d(x, H(u, r))$.

In the remainder of this section we establish the basic result that the visibility center of a set of points U and the geodesic center of a set of half-polygons $\mathcal{H}$ are unique except in very special cases, and that two or three tight constraints suffice to determine the centers. We explain this for the geodesic center of half-polygons, but the same argument works for the visibility center (or, alternatively, one can use the reduction from the visibility center to the geodesic center in Section 5.2). Note that if the geodesic radius is 0, then any point in the intersection of the half-polygons is a geodesic center.

Claim 13. *Suppose that the geodesic radius r satisfies $r > 0$. There is a set $\mathcal{H}' \subseteq \mathcal{H}$ of two or three half-polygons such that the set of geodesic centers of $\mathcal{H}$ is equal to the set of geodesic centers of $\mathcal{H}'$ and furthermore*

1. *if $\mathcal{H}'$ has size 3 then the geodesic center is unique (e.g., see Figure 5.1)*
2. *if $\mathcal{H}'$ has size 2 then either the geodesic center is unique or the two half-polygons of $\mathcal{H}'$ have chords that are parallel and the geodesic center consists of a line segment parallel to them and midway between them.*

The proof of this claim depends on a basic convexity property of the geodesic distance function that was proved for the case of distance to a vertex by Pollack et al. [91, Lemma 1] and that we extend to a half-polygon.

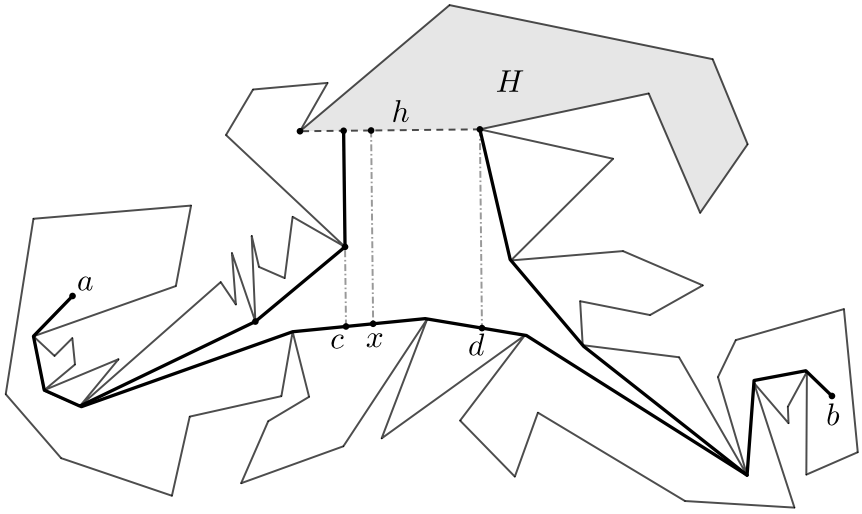


Figure 5.2: Proving that $d(x, H)$ is a convex function of $d(a, x)$.

Lemma 14. *For any half-polygon H, the distance function $d(x, H)$ is geodesically convex. Furthermore, on any geodesic path $\pi(a, b)$ with a and b outside H, the minimum of $d(x, H)$ occurs at a point or along a line segment parallel to h, the defining chord of H.*

Proof. Pollack et al. [91] proved the version of this where H is replaced by a point—in particular, they proved that the distance function is strictly convex which implies that the minimum occurs at a point.

If a and b are inside H, then so is $\pi(a, b)$ and the distance function is zero at every point on $\pi(a, b)$.

So suppose $a \notin H$. If $b \in H$, the path $\pi(a, b)$ first intersects H at some point b^* on the defining chord h of H. For every point $y \in \pi(b^*, b)$, $d(y, H) = 0$. So we initially restrict our attention to points b that do not lie in the interior of H. It is possible that $\pi(a, b)$ coincides with h on a subinterval of h. For points $x \in \pi(a, b)$ that lie on h we think of the path $\pi(x, H)$ as 0-length segments perpendicular to h at x.

The shortest paths $\pi(x, H)$ for $x \in \pi(a, b)$ reach a subinterval $[h_1, h_2]$ of h. See Figure 5.2. In case this subinterval is a single point, i.e., $h_1 = h_2$, the convexity result of Pollack et al. proves the lemma. Otherwise, since shortest paths do not cross, there are points c and d on $\pi(a, b)$ such that for $x \in \pi(a, c)$ the path $\pi(x, H)$ arrives at h_1, for $x \in \pi(c, d)$ the path $\pi(x, H)$ is a straight line segment reaching h at a right angle, and for $x \in \pi(d, b)$ the path $\pi(x, H)$ arrives at h_2. The convexity result of Pollack et al. applies to the paths arriving at the points h_1 and h_2.

It remains to consider $x \in \pi(c, d)$. As x moves along $\pi(c, d)$, the endpoint of $\pi(x, H)$ moves continuously with a one-to-one mapping along the segment $[h_1, h_2]$. Since the curve $\pi(c, d)$ is convex, this implies that $d(x, H)$ is a convex function of $d(a, x)$ for $x \in \pi(c, d)$. Furthermore, the minimum is unique unless a segment of the geodesic is parallel to h.

Finally, one can verify that convexity holds at the points c and d, i.e., that the three convex functions join to form a single convex function. □

Because the intersection of geodesically convex sets is geodesically convex, and the maximum of geodesically convex functions is geodesically convex, we get the following consequences.

Corollary 14.1. *The geodesic radius function $r(x, \mathcal{H})$ is geodesically convex. The geodesic ball $B(t, H) := \{x \in P : d(x, H) \leq t\}$ for any half-polygon H, and the geodesic ball $B(t) := \{x \in P : r(x, \mathcal{H}) \leq t\}$ are geodesically convex.*

Proof of Claim 13. The set of geodesic centers is $C := \{x \in P : d(x, H) \leq r \text{ for all } H \in \mathcal{H}\}$. By Corollary 14.1, C is geodesically convex. If C contains two distinct points x_1 and x_2 then it contains the geodesic path $\gamma = \pi(x_1, x_2)$. By Lemma 14, along γ, for each $H \in \mathcal{H}$, the minimum of the distance function $d(x, H)$ occurs at a point or along a line segment parallel to h. This implies that γ can only be a single line segment parallel to two of the half-polygons of $\mathcal{H}$, which is Case 2 of the Claim.

For Case 1, let us now suppose that C consists of a single point. Because the boundary of each geodesic ball $\{x \in P : d(x, H) \leq r\}$ consists of circular arcs and line segments, the single point C is uniquely determined as the intersection of some circular arcs and line segments, and three of those suffice to determine the point. □

5.2 Reducing the Visibility Center to the Center of Half-Polygons

In this section we reduce the problem of finding the visibility center of a set of points U in a polygon P to the problem of finding the geodesic center of a linear number of "essential" half-polygons $\mathcal{H}$, which is solved in Section 5.3.

By Observation 12 (and see Figure 5.1) the visibility center of U is the geodesic center of the set of $O(mn)$ half-polygons $H(u, r)$ where $u \in U$, r is a reflex vertex of P that sees u, and $H(u, r)$ is the half-polygon containing u and bounded by the chord that extends $\overrightarrow{ur}$ from r until it hits ∂P at a point t. Note that finding t is a ray shooting problem and costs $O(\log n)$ time after an $O(n)$ time preprocessing step [55].

However, this set of half-polygons is too large. We will find a set $\mathcal{H}$ of $O(n)$ "essential" half-polygons that suffice, i.e., such that the visibility center of U is the geodesic center of the half polygons of $\mathcal{H}$. In fact, we give two possible sets of essential half-polygons, $\mathcal{H}_{\text{reflex}}$ and $\mathcal{H}_{\text{hull}}$, where the latter set can be found more efficiently. Although the bottleneck is still the algorithm for geodesic center of half-polygons, it seems worthwhile to optimize the reduction.

We first observe that any half-polygon that contains another one is redundant. For example, in Figure 5.1, $H(u_4, r_4)$ is redundant because it contains $H(u_5, r_4)$. At each reflex vertex r of P, there are at most two minimal half-polygons $H(u, r)$. Define $\mathcal{H}_{\text{reflex}}$ to be this set of minimal half-polygons. Note that $\mathcal{H}_{\text{reflex}}$ has size $O(n_r)$ where n_r is the number of reflex vertices of P.

Observe that for the case of finding the visibility center of *all* points of P, $\mathcal{H}_{\text{reflex}}$ consists of the half-polygons $H(v,r)$ where (v,r) is an edge of P, so $\mathcal{H}_{\text{reflex}}$ can be found in time $O(n + n_r \log n)$.

For a point set U, the set $\mathcal{H}_{\text{reflex}}$ was also used by Ke and O'Rourke [61] in their algorithm to compute the kernel of point set U in polygon P. (Recall from the Introduction that the kernel of U is the set of points in P that see all points of U.) They gave a sweep line algorithm ("Algorithm 2") to find $\mathcal{H}_{\text{reflex}}$ in time $O((n+m)\log(n+m))$. To summarize:

Proposition 15. *The geodesic center of $\mathcal{H}_{\text{reflex}}$ is the visibility center of U. Furthermore, $\mathcal{H}_{\text{reflex}}$ can be found in time $O((n+m)\log(n+m))$.*

In the remainder of this section we present a second approach using $\mathcal{H}_{\text{hull}}$ that provides the solution in $O(n+m\log m)$ time (thus effectively eliminating an $O(n\log n)$ term). This does not change the runtime to find the visibility center, but it means that improving the algorithm to find the geodesic center of half-polygons will automatically improve the visibility center algorithm. The idea is that $\mathcal{H}_{\text{reflex}}$ is wasteful in that a single point $u \in U$ can give rise to n_r half-polygons. Note that we really only need three half-polygons in an essential set, though the trouble is to find them!

We first eliminate the case where the kernel of U is non-empty (i.e., the visibility radius $r_V = 0$) by running the $O(n + m\log(n+m))$ time kernel-finding algorithm of Ke and O'Rourke [61]. Next we find $\mathcal{H}_{\text{hull}}$ in two steps. First make a subset $\mathcal{H}_0$ as follows. Construct R, the geodesic convex hull of U in P in time $O(n + m\log(m+n))$ [50, 100]. For each edge (u,r) of R where $u \in U$ and r is a reflex vertex of P, put $H(u,r)$ into $\mathcal{H}_0$. Note that $\mathcal{H}_0$ has size $O(\min\{n_r, m\})$ so ray shooting to find the endpoints of the chords $H(u,r)$ takes time $O(n+\min\{n_r,m\}\log n)$. Unfortunately, as shown in Figure 5.3, $\mathcal{H}_0$ can miss an essential half-polygon.

Next, construct a geodesic center c_0 of $\mathcal{H}_0$ using the algorithm of Section 5.3. (Note that the geodesic center can be non-unique and in such cases c_0 denotes any one point from the set of geodesic centers.) Then repeat the above step for $U \cup \{c_0\}$. More precisely, construct R', the geodesic convex hull of $U \cup \{c_0\}$ in P and for each edge (u,r) of R' where $u \in U$ and r is a reflex vertex of P, add $H(u,r)$ to $\mathcal{H}_0$. This defines $\mathcal{H}_{\text{hull}}$. Again, $\mathcal{H}_{\text{hull}}$ has size $O(\min\{n_r, m\})$ and ray shooting costs $O(n + \min\{n_r, m\}\log n)$.

Theorem 16. *Suppose the kernel of U is empty. Then the geodesic center of $\mathcal{H}_{\text{hull}}$ is the visibility center of U. Furthermore $\mathcal{H}_{\text{hull}}$ can be found in time $O(n + m\log(n+m))$ plus the time to find the geodesic center of $O(\min\{n_r, m\})$ half-polygons.*

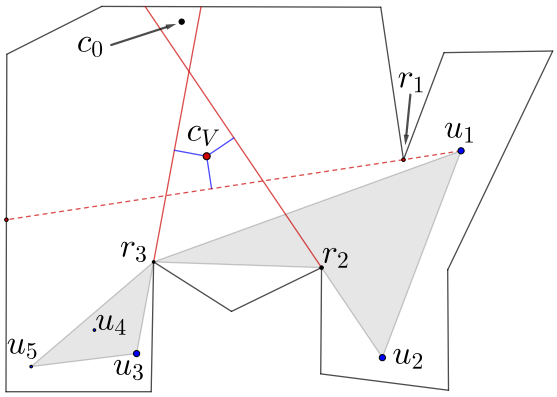


Figure 5.3: The geodesic convex hull of $U = \{u_1, \ldots, u_5\}$ is shaded grey. $\mathcal{H}_0$ consists of the two half-polygons $H(u_2, r_2)$ and $H(u_3, r_3)$ (with solid red chords), but misses $H(u_1, r_1)$, which is essential for the visibility center c_V. The point c_0 denotes a geodesic center of $\mathcal{H}_0$.

Proof. The run-time was analyzed above. Consider the visibility center c_V. By assumption, $r_V > 0$. We consider the half-polygons $H(u, r) \in \mathcal{H}_{\text{reflex}}$ such that $r_V = d(c_V, H(u, r))$. By Claim 13 either there are three of these half-polygons, H_1, H_2 and H_3, that uniquely determine c_V, or there are two, H_1 and H_2, that determine c_V. Then c_V is the geodesic center of H_i $i = 1, 2, 3$ or $i = 1, 2$ depending on which case we are in. Let $H_i = H(u_i, r_i)$.

If all the H_i's are in $\mathcal{H}_0$, we are done. We will show that at least two are in $\mathcal{H}_0$ and the third one (if it exists) is "caught" by c_0. See Figure 5.3. Let h_i be the chord defining H_i and let $\overline{H}_i$ be the other half-polygon determined by h_i.

Claim 17. *If U contains a point in $\overline{H}_i$, then (u_i, r_i) is an edge of R so $H_i \in \mathcal{H}_0$.*

Proof. Let u be a point in $\overline{H}_i$. Observe that $\pi(u_i, u)$ contains the segment $u_i r_i$. Thus r_i is a vertex of R. Furthermore $u_i r_i$ is an edge of R. (Note that H_i is extreme at r since we picked it from $\mathcal{H}_{\text{reflex}}$.) Thus H_i is in $\mathcal{H}_0$. □

Claim 18. *At least two of the H_i's lie in $\mathcal{H}_0$.*

Proof. First observe that if two of the half-polygons are disjoint, say H_i and H_j, then they lie in $\mathcal{H}_0$, because $u_i \in H_i$ implies $u_i \in \overline{H}_j$ so by Claim 17, $H_i \in \mathcal{H}_0$, and symmetrically, $H_j \in \mathcal{H}_0$.

We separate the proof into cases depending on the number of H_i's. If there are two then they must be disjoint otherwise a point in their intersection would be a visibility center

with visibility radius $r_V = 0$, which contradicts the fact that the kernel is empty. Then by the above observation, they are both in $\mathcal{H}_0$.

It remains to consider the case of three half-polygons. If two are disjoint, we are done, so suppose each pair H_i, H_j intersects. Then the three chords h_i form a triangle. Furthermore, since $\bigcap \overline{H}_i$ is non-empty (it contains c_V), the inside of the triangle is $\bigcap \overline{H}_i$. Now suppose $H_1 \notin \mathcal{H}_0$. Then by Claim 17, $u_2, u_3 \in H_1$. This implies (see Figure 5.3) that $u_2 \in \overline{H}_3$ and $u_3 \in \overline{H}_2$, so by Claim 17, H_2 and H_3 are in $\mathcal{H}_0$. □

We now complete the proof of the theorem. We only need to consider the case of three H_i's, where one of them, say H_1, is not in $\mathcal{H}_0$. Our goal is to show that c_0, the geodesic center of $\mathcal{H}_0$, lies in $\overline{H}_1$ and thus H_1 is in $\mathcal{H}_{\text{hull}}$. Let $X = \{x \in P : d(x, H_2) \leq r_V \text{ and } d(x, H_3) \leq r_V\}$. Observe that $c_0 \in X$ (because the radius is non-increasing as we eliminate half-polygons). Now, c_V is the unique point within distance r_V of the half-polygons H_1, H_2 and H_3. If $c_0 \in H_1$, then c_0's distance to H_1 would be 0 which contradicts the uniqueness property of c_V. Thus $c_0 \in \overline{H}_1$. By the same reasoning as in Claim 17, this implies that $u_1 r_1$ is an edge of R', the geodesic convex hull of $U \cup \{c_0\}$. Thus H_1 is in $\mathcal{H}_{\text{hull}}$ by definition of $\mathcal{H}_{\text{hull}}$. □

5.3 The Geodesic Center of Half-Polygons

In this section, we give an algorithm to find the geodesic center of a set $\mathcal{H}$ of k half-polygons inside an n-vertex polygon P with run time $O((n + k) \log(n + k))$. (Note that although we say "the" geodesic center, it need not be unique, see Claim 13.) A half polygon is defined by a chord ab formed by two mutually visible points on the polygon boundary. If we assign a clockwise orientation to the polygon boundary, the polygon chain for the half-polygon either goes clockwise from a to b, or from b to a. The 'first endpoint' for the former half-polygon is a and for the latter half-polygon is b. We preprocess by sorting the half-polygons in cyclic order of their first endpoints around ∂P in time $O(k \log k)$. We assume that no half-polygon contains another—such irrelevant non-minimal half-polygons can be detected from the sorted order and discarded. We also make the general position assumption that no point in P has equal non-zero distances to more than a constant number of half-polygons of $\mathcal{H}$.

We follow the approach that Pollack et al. [91] used to find the geodesic center of the vertices of a polygon. Many steps of their algorithm rely, in turn, on search algorithms of Megiddo's [78].

The main ingredient of the algorithm is a linear time ***chord oracle*** that, given a chord $K = ab$ of the polygon, finds the ***relative geodesic center***, c_K (the optimum center point restricted to points on the chord), and tells us which side of the chord contains the center. We must completely redo the chord oracle in order to handle paths to half-polygons instead of vertices, but the main steps are the same. Our chord oracle runs in time $O(n + k)$. The chord oracle of Pollack et al. was used as a black box in subsequent faster algorithms [4], so we imagine that our version will be an ingredient in any faster algorithm for the geodesic center of half-polygons.

Using the chord oracle, we again follow the approach of Pollack et al. to find the geodesic center. The total run time is $O((n + k) \log(n + k))$.

We give a road-map for the remainder of this section, listing the main steps, which are the same as those of Pollack et al., and highlighting the parts that we must rework.

§ 5.3.1 A Linear Time Chord Oracle

1. Test a candidate center point. Given the relative geodesic center c_K on chord $K = ab$, is the geodesic center to the left of right of chord K? This test reduces the chord oracle to finding the relative geodesic center, which is done via the following steps.

2. Find shortest paths from a and from b to all half-polygons. The details of this step are novel, because we need shortest paths to half-polygons rather than vertices.

3. Find a linear number of simple functions defined on K whose upper envelope is the geodesic radius function. We must redo this from the ground up.

4. Find the relative center on K (the point that minimizes the geodesic radius function) using Megiddo's technique.

§ 5.3.2 Finding the Geodesic Center of Half-Polygons

1. Use the chord oracle to find a region of P that contains the center and such that for any half-polygon $H \in \mathcal{H}$, all geodesic paths from the region to H are combinatorially the same. We give a more modern version of this step using epsilon nets.

2. Solve the resulting Euclidean problem of finding a smallest disk that contains given disks and intersects given half-planes. This is new because of the condition about intersecting half-planes.

5.3.1 A Linear Time Chord Oracle

In this section we give a linear time chord oracle. Given a chord $K = ab$ the chord oracle tells us whether the geodesic center of $\mathcal{H}$ lies to the left, to the right, or on the chord K. It does this by first finding the relative geodesic center $c_K = \text{argmin}\{r(x, \mathcal{H}) : x \in K\}$, together with the half-polygons that are farthest from c_K and the first segments of the shortest paths from c_K to those farthest half-polygons. From this information, we can identify which side of K contains the geodesic center c in the same way as Pollack et al. by testing the vectors of the first segments of the shortest paths from c_K to its furthest half-polygons. This test is described in subsection 5.3.1.1.

The chord oracle thus reduces to the problem of finding the relative geodesic center and its farthest half-polygons. The main idea here is to capture the geodesic radius function along the chord (i.e., the function $r(x, \mathcal{H})$ for $x \in K$) as the upper envelope of a linear number of easy-to-compute convex functions defined on overlapping subintervals of K. In order to find the functions (Section 5.3.1.3) we first compute shortest paths from a and from b to all the half-polygons (Section 5.3.1.2). Finally we apply Megiddo's techniques (Section 5.3.1.4) to find the point c_K on K that minimizes the geodesic radius function.

We will thus prove the following lemma:

Lemma 19. *There is a linear-time (i.e. $O(n+k)$ time) chord oracle for the geodesic center of k half-polygons in an n vertex simple polygon.*

5.3.1.1 Testing a Candidate Center Point

In this section we show how to test in constant time whether a candidate point is a geodesic center, or relative center, and if not, in which direction the center lies. The basic idea is that a local optimum is a global optimum, so a local test suffices. In more detail, the input is a point x on a chord $K = ab$ (K is part of the input) together with its geodesic radius $r(x, \mathcal{H})$ and the first segments of the shortest paths from x to its farthest half-polygons. The goal is to test in constant time: (1) whether x is a relative geodesic center of K, and if not, which direction to go on K to reach a relative center; and (2) if x is a relative geodesic center, whether x is a geodesic center of P, and if not, which side of K contains a geodesic center of P. These tests are illustrated in Figure 5.4. Note that if $r(x, \mathcal{H})$ is zero, then x is a geodesic center and no further work is required.

The tests are accomplished via the following lemma, which is analogous to Lemmas 2 and 3 of Pollack et al. [91].

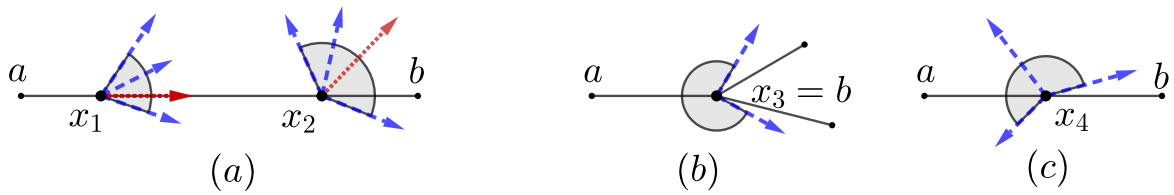


Figure 5.4: Points x_i on chord $K = ab$ with directions of paths to farthest half-polygons in dashed blue, wedge W_α shaded, the direction for improvement in dotted red. (a) x_1 is not a relative center of K, whereas x_2 is a relative center but not the geodesic center of P. (b,c) x_3 (a reflex vertex of the polygon) and x_4 are geodesic centers.

Lemma 20. *Let x be a point on chord K, and let V be the vectors of the first segments of the shortest paths from x to its farthest half-polygons $\mathcal{F}$. Let α be the smallest angle of a wedge W_α with apex x that contains all the vectors of V and such that W_α, restricted to a small neighbourhood of x, is contained in P.*

1. **Location of the relative center.** *Let L be the line through x perpendicular to K. If one of the open half-planes determined by L contains W_α, then x is not a relative center, and all relative centers lie on that side of L. Otherwise, x is a relative center.*

2. **Location of the center.** *Now suppose that x is a relative geodesic center. If $\alpha < \pi$ then x is not a geodesic center, and all geodesic centers lie on the side of K that contains the ray bisecting the angle of W_α. If $\alpha > \pi$, then x is the unique geodesic center, and furthermore, x is determined by two or three vectors of V—the two that bound W_α, plus one inside W_α unless x is on the boundary of P. Finally, if $\alpha = \pi$ then x is a geodesic center (though not necessarily unique), and furthermore, x is determined by the two vectors of V that bound W_α.*

Proof. We prove the two parts separately.

1. Suppose W_α lies in an open half-plane determined by L (say, the right side of L). Then moving x an epsilon distance left along K gives a point with smaller geodesic radius since the distance to any half-polygon in $\mathcal{F}$ decreases, and no other half-polygon becomes a farthest half-polygon. Therefore x is not a relative center. Furthermore, because the geodesic radius function $r(x, \mathcal{H})$ is convex on K (by Corollary 14.1), the relative center lies to the left on K.

   Next suppose W_α does not lie in an open half-plane of L. Then any epsilon movement of x along K increases the distance to some half-polygon in $\mathcal{F}$, so x is a local minimum

on K and therefore x is the relative center (again using the face that the geodesic radius function is convex on K).

2. Suppose $\alpha < \pi$. Let b be the ray that bisects W_α. Moving x an epsilon distance along b gives a point x' with smaller geodesic radius. Therefore x is not the center. Next we prove that the center c lies on the side of K that contains b. Suppose not. Consider the geodesic $\pi(c, x')$. By Corollary 14.1, the geodesic radius function is convex on $\pi(c, x')$. But then the point where the geodesic crosses K has a smaller geodesic radius than x, a contradiction to x being the relative center.

   Next suppose $\alpha > \pi$. Let v_1 and v_2 be the two vectors that bound W_α. If x is on the boundary of P it must be at a reflex vertex of P. Otherwise, since no smaller wedge contains V, there must be a third vector v_3 in V, making an angle $< \pi$ with each of v_1 and v_2. In either case (x on the boundary of P or not) any epsilon movement of x in P increases the distance to the half-polygon corresponding to one of the v_i's. Thus x is a local minimum in P and (by geodesic convexity of the radius function) x is the center. Furthermore, x is determined by v_1 and v_2—and v_3 if x is interior to P.

   Finally, suppose $\alpha = \pi$. As in the previous case, x is a geodesic center and is determined by the two vectors v_1 and v_2 of V that bound W_α. Furthermore, x is unique unless the two corresponding half-polygons have parallel defining chords, and v_1 and v_2 reach those chords at right angles. In this case the set of geodesic centers consists of a line segment through x parallel to the chords.

□

5.3.1.2 Shortest Paths to Half-Polygons

In this section we give a linear time algorithm to find the shortest path tree from point a on the polygon boundary to all the half-polygons $\mathcal{H}$. Recall that each half-polygon is specified by an ordered pair of endpoints on ∂P, and the half-polygons are sorted in clockwise cyclic order by their first endpoints. From this order, we identify the half-polygons that contain a, and we discard them—their distance from a is 0. Let $H_1, \ldots, H_{k'}$ be the remaining half-polygons where H_i is bounded by endpoints $p_i q_i$, and the H_i's are sorted by p_i, starting at a.

The idea is to first find the shortest path map T_a from a to the set consisting of the polygon vertices and the points p_i and q_i, for all i. Recall that the shortest path map

is an augmentation of the shortest path tree that partitions the polygon into triangular regions in which the shortest path from a is combinatorially the same (see Figure 5.5). The shortest path map can be found in linear time [49]. Note that T_a is embedded in the plane (none of its edges cross) and the ordering of its leaves matches their ordering on ∂P. Our algorithm will traverse T_a in depth-first order, and visit the triangular regions along the way.

Our plan is to augment T_a to a shortest path tree $\bar{T}_a$ that includes the shortest paths from a to each half-polygon H_i. Note that $\bar{T}_a$ is again an embedded ordered tree. We can find $\pi(a, H_i)$ by examining the regions of the shortest path map intersected by p_iq_i. These lie in the *funnel* between the shortest paths $\pi(a, p_i)$ and $\pi(a, q_i)$. Note that edges of the shortest path map T_a may cross the chord p_iq_i. Also, the funnels for different half-polygons may overlap. The key to making the search efficient is the following lemma:

Lemma 21. *The ordering $H_1, H_2, \ldots, H_{k'}$ matches the ordering of the paths $\pi(a, H_i)$ in the tree $\bar{T}_a$.*

Proof. Consider two half-polygons $H_i = p_iq_i$ and $H_j = p_jq_j$, with $i < j$. We prove that $\pi(a, H_i)$ comes before $\pi(a, H_j)$ in $\bar{T}_a$. If H_i and H_j are disjoint, the result is immediate since the corresponding funnels do not overlap. Otherwise (because neither half-polygon is contained in the other) p_iq_i and p_jq_j must intersect, say at point x. See Figure 5.5. Let t_i and t_j be the terminal points of the paths $\pi(a, H_i)$ and $\pi(a, H_j)$, respectively. If t_i lies in p_ix and t_j lies in xq_j then the result follows since t_i and t_j lie in order on the boundary of the truncated polygon formed by removing H_i and H_j. So suppose that t_j lies in p_jx (the other case is symmetric). Then $\pi(a, t_j)$ crosses p_iq_i at a point z in p_ix. From z to t_j the path $\pi(a, t_j)$ lies inside the cone with apex x bounded by the rays from x through z and from x through t_j. Within that cone, the path only turns left. The angle α_j at t_j is $\geq 90°$ (it may be $> 90°$ if $t_j = p_j$), which implies that the angle α_i at z is $> 90°$. Therefore t_i lies to the left of z, as required. □

Based on the lemma, the algorithm traverses the regions of the shortest path map T_a in depth first search order, and traverses the half-polygons H_i in order $i = 1, 2, \ldots, k'$. It is easy to test if one region contains the shortest path to H_i (either to p_i, or to q_i, or reaching an internal point of p_iq_i at a right angle); if it does, we increment i, and otherwise we proceed to the next region. The total time is $O(n + k)$.

5.3.1.3 Functions to Capture the Distance to Farthest Half-Polygons

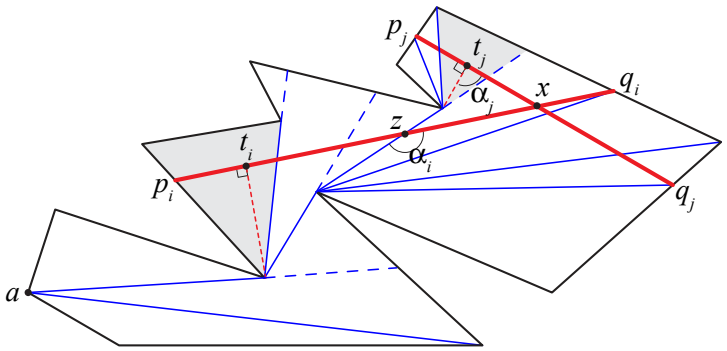


Figure 5.5: The shortest path map T_a (thin blue) and the augmentation (dashed red) to include shortest paths to the two half-polygons bounded by chords p_iq_i and p_jq_j (thick red).

In this section we capture the geodesic radius function for points on a chord $K = ab$ as the upper envelope of functions defined on overlapping subintervals of K. Besides extending the method of Pollack et al. [91] to deal with half-polygons (rather than vertices), our aim is to give a clearer and easier-to-verify presentation.

In more detail, we give a linear time algorithm to find a linear number of easy-to-compute convex functions defined on the chord $K = ab$ whose upper envelope is the geodesic radius function $r(x, \mathcal{H})$ for $x \in K$. Specifically, a ***coarse cover for a chord*** K is a set of triples (I, f, H) where:

1. I is a subinterval of K, f is a function defined on domain I, and $H \in \mathcal{H}$.
2. $f(x) = d(x, H)$ for all $x \in I$, and f has one of the following forms:
   - $f(x) = 0$.
   - $f(x) = d_2(x, v) + \kappa$ where d_2 is the Euclidean distance, κ is a constant, v is a vertex of P, and the segment xv is the first segment of the path $\pi(x, H)$.
   - $f(x) = d_2(x, \bar{h})$, where d_2 is the Euclidean distance, $\bar{h}$ is the line through the defining chord of H, and the path $\pi(x, H)$ is the straight line segment from x to $\bar{h}$ (meeting $\bar{h}$ at right angles).
3. For any point $x \in K$ and any half-polygon H that is farthest from x, there is a triple (I, f, H) in the coarse cover with $x \in I$—with the exception that if two triples have identical I and identical $f = d_2(x, v) + \kappa$ then we may eliminate one of them.

In particular, this implies that the upper envelope of the functions is the geodesic radius, i.e., for any $x \in K$, the maximum of $f(x)$ over intervals I containing x is equal to $r(x, \mathcal{H})$.

For intuition, see Figure 5.6, which shows several intervals and their associated functions. We will find the elements of the coarse cover separately for the two pieces of the polygon on each side of K, and then take the union of the two sets. In this section we visualize K as horizontal and deal with the upper piece of the polygon.

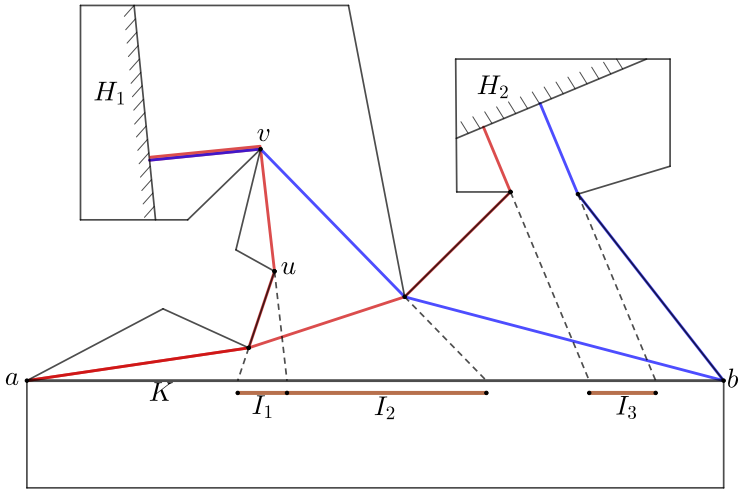


Figure 5.6: An illustration of functions and intervals. For x in interval I_1, $d(x, H_1) = d_2(x, u) + \kappa_1$. For x in I_2, $d(x, H_1) = d_2(x, v) + \kappa_2$. For x in I_3, $d(x, H_2) = d_2(x, H_2)$.

A large coarse cover. We first describe a coarse cover of $O(nk)$ triples and then show how to reduce it to linear size. Consider a half-polygon H with defining chord h. Suppose first that K does not intersect H, i.e., a and b lie outside H. All shortest paths from points on K to H lie in the ***funnel*** $Y(H)$ which is a subpolygon bounded by the chord K, the path $\pi(a, H)$ (which is a path in $\bar{T}_a$), the path $\pi(b, H)$ (in $\bar{T}_b$), and the segment along h between the terminals of those two paths. See Figure 5.7. If the paths $\pi(a, H)$ and $\pi(b, H)$ are disjoint then they are both reflex paths and all vertices on the paths are visible from K (see Figure 5.7(a)). Otherwise, the paths are reflex and visible from K until they reach the first common vertex u, and then they have a common subpath from u to H that is not visible from K (see Figure 5.7(b)).

Before describing how to obtain triples of the coarse cover from $Y(H)$, we first consider the case when K intersects H, i.e., a or b lies inside H. If both a and b are inside H, then

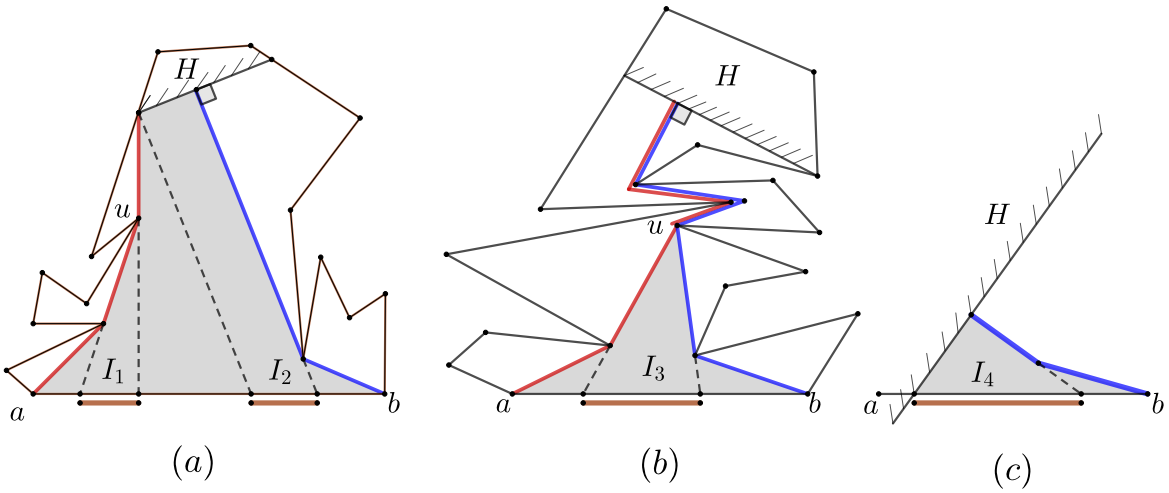


Figure 5.7: The funnel $Y(H)$ (shaded) and its shortest path map which is bounded by edge extensions (dashed segments). (a) The case of disjoint paths $\pi(a, H)$ and $\pi(b, H)$. For $x \in I_1$, $d(x, H) = d_2(x, u) + \kappa$. For $x \in I_2$, $d(x, H) = d_2(x, \bar{h})$. (b) The case of paths that meet at vertex u. (c) The case where $a \in H$.

we add the triple $(I = ab, f = 0, H)$ to the coarse cover. If b is outside but a is inside (the other case is symmetric), then h and K intersect at a point p. If $\pi(b, H)$ reaches H below K, then H will be handled when we deal with the piece of the polygon below K. Otherwise (see Figure 5.7(c)) we add the triple $(I = ap, f = 0, H)$ to the coarse cover, and we deal with the pb portion of the chord as in the general case above but modifying the funnel $Y(H)$ so that the path $\pi(a, H)$ is replaced by p.

Each funnel $Y(H)$ can be partitioned into its ***shortest path map* $M(H)$** where two points are in the same region of $M(H)$ if their paths to H are combinatorially the same. (We consider a path that arrives at an endpoint of h and a path that arrives at an interior point of h to be combinatorially different.) Observe that boundaries of the regions of $M(H)$ are extensions of tree edges plus lines perpendicular to h. See Figure 5.7. The regions of $M(H)$ are triangles, plus possibly one trapezoid. A triangle region has a base segment $I \subseteq K$, and an apex vertex u on $\pi(a, H)$ [or $\pi(b, H)$]; the shortest path from any point $x \in I$ to H consists of the line segment xu plus the path in $\bar{T}_a$ [or $\bar{T}_b$] from u to H, so $d(x, H) = d_2(x, u) + \kappa$ where κ is the tree distance from u to the leaf corresponding to H. A trapezoid region has a base segment $I \subseteq K$, and two sides orthogonal to h; the shortest path from any point $x \in I$ to H consists of the line segment orthogonal to h from x to h, so $d(x, H) = d_2(x, \bar{h})$ where $\bar{h}$ is the line through h. Thus each region of $M(H)$ gives rise to a triple (I, f, H) satisfying properties (1) and (2) of a coarse cover.

We claim that the set of triples defined above, i.e., all the triples defined from $Y(H)$ together with the special triples when H intersects K, form a coarse cover. Properties (1) and (2) are satisfied, and property (3) is satisfied because we have captured all shortest paths from x to H for all $x \in K$ and all half-polygons H. Since each $Y(H)$ has size $O(n)$, this coarse cover has size $O(nk)$.

Intuition for a linear-size coarse cover. The secret to reducing the size of the coarse cover is to observe that if the funnels for some half-polygons $\mathcal{H}' \subseteq \mathcal{H}$ share an edge uv of $\bar{T}_a$ with u closer to the root, and both u and v visible from K, then their shortest path maps share the same triangle with apex u, base I, and sides bounded by the extension of the edge from v to u and the extension of the edge from u to its parent in $\bar{T}_a$ (see Figure 5.8(a)). In this case, we claim that for this triangle, we only need a coarse cover element for one of the half-polygons in $\mathcal{H}'$, specifically, for one half-polygon that has the maximum distance from v in the tree $\bar{T}_a$. This is because only half-polygons farthest from v matter, and furthermore, we need not keep more than one half-polygon that has the maximum distance because the interval I and the function $f(x) = d_2(x, u) + \kappa$ are the same. We first specify the coarse cover precisely and then prove correctness, which makes the above observation formal.

Definitions. Let $\bar{T}_a$ and $\bar{T}_b$ be directed from root to leaves. For any node v in $\bar{T}_a$ define $\boldsymbol{\ell_a(v)}$ to be the maximum length of a directed path in $\bar{T}_a$ from v to a leaf node representing a terminal point on some half-polygon, and define $\boldsymbol{F_a(v)}$ to be that farthest half-polygon (breaking ties arbitrarily). Define functions $\boldsymbol{\ell_b}$ and $\boldsymbol{F_b}$ similarly. We can compute these functions in linear time in leaf-to-root order. In particular, we compute $\ell_a(u)$ for the nodes u of $\bar{T}_a$ as follows. Initialize $\ell_a(u)$ to 0 if u represents a terminal point of a half-polygon chord, and to $-\infty$ otherwise. Then from leaf-to-root order, update $\ell_a(u)$ to $\max\{\ell_a(u), \max\{|uv| + \ell_a(v) : v \text{ a child of } u\}\}$. We can compute $\ell_b(u)$ similarly. The runtime is $O(n + k)$.

Define $\boldsymbol{p_a(u)}$ and $\boldsymbol{p_b(u)}$ to be the parents of node u in $\bar{T}_a$ and $\bar{T}_b$, respectively. As noted by Pollack et al. [91], a vertex u is visible from some point on K if and only if $p_a(u) \neq p_b(u)$. Furthermore, we note that if u is visible from some point on K, then extending the edge from u through $p_a(u)$ reaches a point $\boldsymbol{x_a(u)}$ on K from which u is visible. Similarly, extending the edge from u through $p_b(u)$ reaches a point $\boldsymbol{x_b(u)}$ on K from which u is visible.

In defining the shortest path map $M(H)$, we added boundary lines orthogonal to the defining chord h at the terminals of the paths $\pi(a, H)$ and $\pi(b, H)$. If a path terminates at an internal point of h then the last edge of the path is orthogonal to h, and the boundary line extends the last edge. In order to avoid special cases, it will be convenient if all boundary

lines are extensions of tree edges, i.e., to assume that even the paths that terminate at endpoints of h end with a segment orthogonal to H. We add 0-length segments to the trees $\bar{T}_a$ and $\bar{T}_b$ to make this true. The extension of such a 0-length edge is orthogonal to H. Note that it is possible that both $\pi(a, H)$ and $\pi(b, H)$ terminate at the same endpoint of h, in which case the added 0-length segment is common to both trees, so we regard the terminal point of the paths as *not* visible from K. See Figure 5.8(c). (The other endpoint of the 0-length segment may or may not be visible.)

The coarse cover $\mathcal{T}$. Define $\boldsymbol{\mathcal{T}}$ to have elements of the following four types. See Figure 5.8.

0. For each half-polygon H that intersects K there is a coarse cover element (I, f, H) where $I = K \cap H$ and $f(x) = 0$.
1. For each edge (u, v) in $\bar{T}_a$ where $u = p_a(v)$, $u \neq a$, and u and v are both visible from K, there is an associated **a-side triangle** that has apex u and base $I = [x_a(u), x_a(v)] \subseteq K$. The associated coarse cover element is (I, f, H) where $H = F_a(v)$ and $f(x) = d_2(x, u) + |uv| + \ell_a(v)$. Define **$b$-side triangles** and their associated coarse cover elements symmetrically.
2. For each edge (u, v) that is common to $\bar{T}_a$ and $\bar{T}_b$ where u is visible from K and v is not (i.e., $u = p_a(v) = p_b(v)$) there is an associated **central triangle** that has apex u and base $I = [x_a(u), x_b(u)] \subseteq K$. The associated coarse cover element is (I, f, H) where $H = F_a(v) = F_b(v)$ and $f(x) = d_2(x, u) + |uv| + \ell_a(v)$.
3. For each half-polygon H such that the terminal points $\boldsymbol{t(a, H)}$ of $\pi(a, H)$ and $\boldsymbol{t(b, H)}$ of $\pi(b, H)$ are distinct, there is a **central trapezoid** with base $I \subseteq K$ bounded by the two lines perpendicular to h emanating from $t(a, H)$ and $t(b, H)$—these lines are the extensions of the (possibly 0-length) last edges of the paths. The associated coarse cover element is (I, f, H) where H is the given half-polygon and $f(x) = d_2(x, \bar{h})$ where $\bar{h}$ is the line through the defining chord of H.

Note that we include 0-length edges in cases 1 and 2 above. Altogether, $\mathcal{T}$ contains $O(n + k)$ triples—at most one associated with each edge of the trees, and at most two associated with each half-polygon $H \in \mathcal{H}$.

Lemma 22. *$\mathcal{T}$ is a linear-sized coarse cover of chord K with respect to farthest half-polygons.*

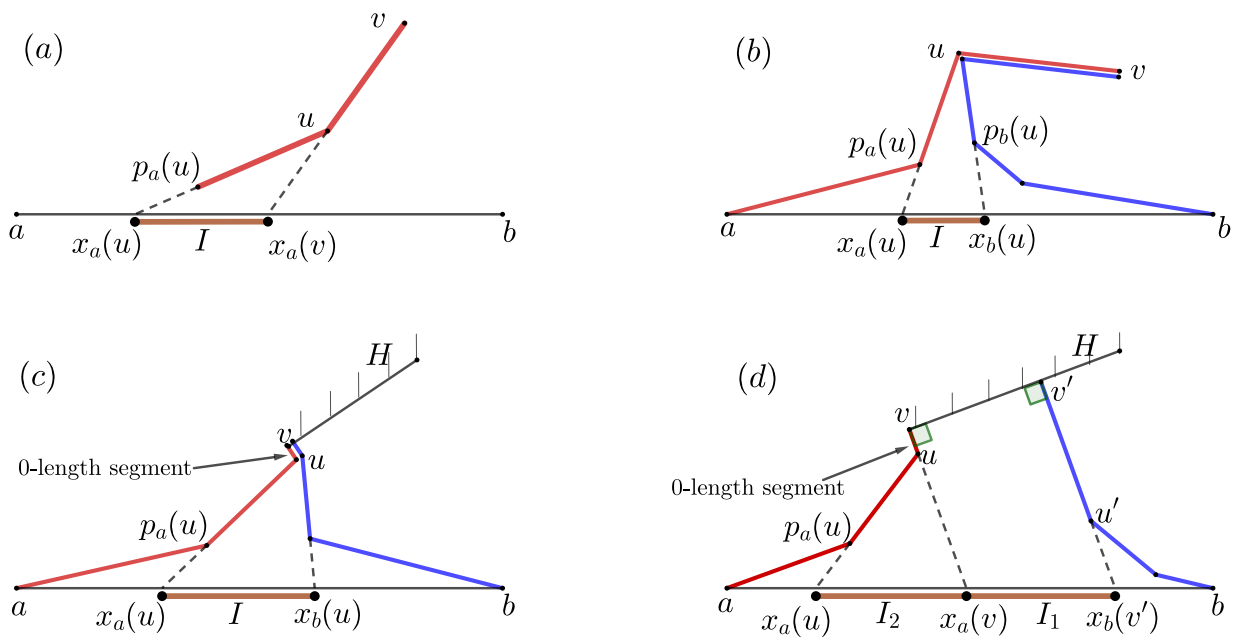


Figure 5.8: Elements of the coarse cover: (a) I is the base of an a-side triangle associated with edge uv; (b) I is the base of a central triangle associated with edge uv; (c) I is the base of a central triangle associated with 0-length edge uv; (d) I_1 is the base of a central trapezoid associated with terminal points v and v' on H; I_2 is the base of an a-side triangle associated with 0-length edge uv.

Proof. There are three properties for a coarse cover. For each triple (I, f, H) it is clear that I is a subinterval of K and f is defined on I—this is property (1). For property (2), f is defined to have one of the three forms. Furthermore, we claim that for each triple (I, f, H), and each $x \in I$, $f(x) = d(x, H)$. This is clear for elements of type 0. For the other three cases, the triangle or trapezoid is part of the shortest path map $M(H)$, so the formula for $f(x)$ matches the distance $d(x, H)$.

Finally, we must prove property (3). Let $\mathcal{T}^0$ be the initial large coarse cover defined above, consisting of the set of triples (I, f, H) from elements of type 0 and the union of all the triples arising from the shortest path maps $\{M(H) : H \in \mathcal{H}\}$. Then $\mathcal{T} \subseteq \mathcal{T}^0$, and we must show that no triple of $\mathcal{T}^0$ that is omitted from $\mathcal{T}$ causes a violation of property (3). Any triple from the shortest path maps corresponds to a triangle that arises in elements of type 1 or 2 (i.e., with the same u, v, I), or to a trapezoid considered in elements of type 3. No trapezoids are omitted in $\mathcal{T}$, so it suffices to consider elements of type 1 and 2.

We first examine elements of type 1. Consider an edge (u, v) in $\bar{T}_a$ where $u = p_a(v)$ and

u and v are both visible from K, and consider the interval $I = [x_a(u), x_a(v)]$. Suppose that for some f, H, there is a triple (I, f, H) that is included in the triples from the shortest path maps, but omitted from $\mathcal{T}$. Then there is a directed path in $\bar{T}_a$ from v to a leaf corresponding to H, and $f(x)$ is $d_2(x, u) + |uv| + \kappa$ where κ is the length of the tree path from v to the leaf corresponding to H. But then $f(x) \leq d_2(x, u) + |uv| + \ell_a(v)$, since $\ell_a(v)$ is the maximum distance from v to a leaf corresponding to farthest half-polygon $F_a(v)$. If the inequality is strict, then H is not a farthest half-polygon from any $x \in I$ so property (3) is satisfied without the triple (I, f, H). And if equality holds, then property (3) allows us to omit the triple (I, f, H) since the triple $(I, f, F_a(v))$ has the same I and f. The case of triples omitted in elements of type 2 is similar.

The linear size of $\mathcal{T}$ follows from the fact that we define at most one element of $\mathcal{T}$ for each edge and vertex of $\bar{T}_a$ and $\bar{T}_b$. □

5.3.1.4 Finding the Relative Geodesic Center on a Chord

The last step of the chord oracle is exactly the same as in Pollack et al. [91]. Given a chord K and the coarse cover $\mathcal{T}$ from Section 5.3.1.3—which provides a set of $O(n + k)$ functions whose upper envelope is the geodesic radius function on chord K—we want to find the relative center, c_K, that minimizes the geodesic radius function. Pollack et al. use a technique of Megiddo's to do this in $O(n + k)$ time by recursively reducing to a smaller subinterval of K while eliminating elements of the coarse cover whose functions are strictly dominated by others. In brief, the idea is to pair up the functions, define a set of at most 6 "extended intersection points" for each pair, and test medians of those points in order to restrict the search to a subinterval of K and eliminate a constant fraction of the functions. Testing median points is done via the test from Section 5.3.1.1 of whether the relative center is left/right of a query point x on K. This test depends on having the first segments of the shortest paths from x to its farthest half-polygons. Observe that the initial coarse cover from Section 5.3.1.3 captures these segments, and they are preserved throughout the recursion because only strictly dominated functions are eliminated.

We fill in a bit more detail. In each round we have a subinterval K' of K that contains c_K and a subset $\mathcal{T}'$ of the coarse cover $\mathcal{T}$ such that any function omitted from $\mathcal{T}'$ is strictly dominated on interval K' by a function of $\mathcal{T}'$. We want to eliminate a constant fraction of $\mathcal{T}'$ in time $O(|\mathcal{T}'|)$. We pair up the functions of $\mathcal{T}'$. Consider a pair of functions f_1 and f_2. Each function is defined on a subinterval of K and we define it to be $-\infty$ outside its interval. The upper envelope of f_1 and f_2 switches between f_1 and f_2 at ***extended intersection points*** which include the points where the graphs of f_1 and f_2 intersect $(f_1 = f_2)$, and also possibly the endpoints of their intervals. If a subinterval of K does not

contain an extended intersection point for f_1 and f_2, then one of f_1, f_2 is irrelevant because it is dominated by the other (or both are $-\infty$). We know from Section 5.3.1.3 that each function has the form $f(x) = 0$ or $f(x) = d_2(x, s) + \kappa$ where κ is a constant, d_2 is Euclidean distance, and s is a point or line. This implies that there are at most two intersection points of the functions f_1 and f_2, and thus at most six extended intersection points. In fact, a closer examination shows that there are at most four extended intersection points.

Pollack et al. show how to successively test three medians of extended intersections in order to reduce the interval K' and eliminate a constant fraction of the functions of $\mathcal{T}'$. The first median test reduces the domain to a subinterval containing half the extended intersections, so three successive median tests reduce the domain to a subinterval containing one eighth of the extended intersections. This implies that for at least half the pairs f_1, f_2, all four of their extended intersections lie outside the domain, and one of f_1, f_2 is dominated by the other and can be eliminated.

This completes one round of their procedure, with a runtime of $O(|\mathcal{T}'|)$. When $\mathcal{T}'$ is reduced to constant size, the relative center c_K can be found directly. The total run time is then $O(n + k)$.

5.3.2 Finding the Geodesic Center of Half-Polygons

In this section we show how to use the $O(n + k)$ time chord oracle from Section 5.3.1 to find the geodesic center of the k half-polygons in $O((n + k) \log(n + k))$ time. The basic structure of the algorithm is the same as that of Pollack et al. [91].

In the first step we use the chord oracle to restrict the search for the geodesic center to a small region where the problem reduces to a Euclidean problem of finding a minimum radius disk that intersects some half-planes and contains some disks. This step takes $O((n + k) \log(n + k))$ time. In the second step we solve the resulting Euclidean problem in linear time, which involves some new ingredients to handle our case of half-polygons.

5.3.2.1 Finding a Region that Contains the Geodesic Center

Triangulate P in linear time [24]. Choose a chord of the triangulation that splits the polygon into two subpolygons so that the number of triangles on each side is balanced, i.e., at most a constant fraction of the total number of triangles (the dual of a triangulation is a tree of maximum degree 3, which has a balanced cut vertex). Run the chord oracle on this chord, and recurse in the appropriate subpolygon. In $O(\log n)$ iterations, we narrow

our search down to one triangle T^* of the triangulation. This step takes $O((n+k)\log n)$ time.

Next, we refine T^* to a region R that contains the center and such that R is ***homogeneous***, meaning that for any $H \in \mathcal{H}$ the shortest paths from points in R to H have the same combinatorial structure, i.e., the same sequence of polygon vertices along the path.

The idea is to subdivide T^* by $O(n+k)$ lines so that each cell in the resulting line arrangement is homogeneous, and then to find the cell containing the center. Construct the shortest path trees to $\mathcal{H}$ from each of the three corners of triangle $T^* = (a^*, b^*, c^*)$ using the algorithm of Section 5.3.1.2. For each edge (u, v) of each tree, add the line through uv if it intersects T^*. (In fact, we do not need all these lines—as in the construction of the coarse cover in Section 5.3.1.3, it suffices to use tree edges (u, v) such that u is visible from an edge of T^*.) We add three more lines for each half-polygon $H \in \mathcal{H}$, specifically, the chord h that defines H, and the two lines perpendicular to h through the endpoints of h. The result is a set L of $O(n+k)$ lines that we obtain in time $O(n+k)$. It is easy to see that the resulting line arrangement has homogeneous regions.

All that remains is to find the cell of the arrangement that contains the geodesic center. It is simpler to state the algorithm in terms of ϵ-nets instead of the rather involved description of Megiddo's technique used by Pollack et al. [91] (a brief overview was provided in Section 4.3). The high-level idea is to define a range space with ground set L and to find a constant-sized ϵ-net in time $O(|L|)$. Then the lines of the ϵ-net divide our region into a constant number of subregions and we can find which subregion contains the geodesic center by applying the chord oracle $O(1)$ times. By the property of ϵ-nets the subregion is intersected by only a constant fraction of the lines of L, so repeating this step for $O(\log(n+k))$ times, we arrive at a region R with the required properties.

We fill in a bit more detail. The range space has ground set L. To define the ranges, let $\mathcal{T}$ be the (infinite) set of all triangles contained in T^*. For $t \in \mathcal{T}$, let $\Delta_t = \{\ell \in L : \ell \text{ intersects } t\}$. Let $\Delta = \{\Delta_t : t \in \mathcal{T}\}$. Then the range space is $S = (L, \Delta)$. To show that constant-sized ϵ-nets exist, we must show that S has constant VC-dimension, or constant shattering dimension. We argue that the shattering dimension is 6, i.e., that for any subset L' of L of size m the number of ranges is $O(m^6)$. The lines intersecting a triangle t are the same as the lines intersecting the convex hull of the three cells of the arrangement of L' that contain the endpoints of t. There are $O(m^2)$ cells in the arrangement and we choose three of them, giving the bound of $O(m^6)$ possible ranges. Thus, a constant sized ϵ-net of size $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ exists for our range space. In order to construct an ϵ-net in deterministic $O(|L|)$ time, we need a subspace oracle that, given a subset L' of L of size m, computes the set of ranges of L' in time proportional to the output size, $O(m^{6+1})$. Begin by finding, for

each line in L', which cells of the arrangement lie to each side of the line. Then, for every choice of three cells (there are $O(m^6)$ choices), the lines intersecting their convex hull can be listed in time $O(m)$ time.

For the algorithm, we choose $\epsilon = \frac{1}{2}$, and construct an ϵ-net N. Triangulate each cell of the arrangement of N—this is a constant time operation since the arrangement has constant complexity. We can locate the triangle T' of this triangulated arrangement that contains the visibility center in $O(n+k)$ time by running the chord oracle a constant number of times. By the ϵ-net property, no more than $\epsilon \cdot |L|$ lines of L intersect the interior of T'. Thus, in $O(|L|)$ time, we have halved the number of lines going through our domain of interest (the region that contains the geodesic center). Repeating the same sequence of steps $O(\log |L|) = O(\log(n+k))$ times, we will arrive at a triangle containing a constant number of these lines. At this point, a brute force method suffices to locate a region R that satisfies the properties stated in the beginning of this section. The time taken by the chord oracle remains fixed at $O(n+k)$, even though the number of lines is halved at each step. In each iteration of the process, we apply the $O(n+k)$-time chord oracle a constant number of times and thus the total runtime for this step is $O((n+k)\log(n+k))$.

5.3.2.2 Solving an Unconstrained Problem

At this point, we have a homogeneous polygonal region R that contains a geodesic center of the set $\mathcal{H}$ of half-polygons. Our goal is to find the point $x \in R$ that minimizes the maximum over $H \in \mathcal{H}$ of $d(x, H)$. We give a linear time algorithm (in this final step there is no need for an extra logarithmic factor). We show that the problem reduces to one in the Euclidean plane, i.e., the polygon no longer matters. Pick an arbitrary point p in R and find the shortest path tree from p to all half-polygons (this takes linear time). If p has distance 0 to half-polygon H, then the same is true for all points in R, so H is irrelevant and can be discarded. If $\pi(p, H)$ consists of a single line segment that reaches an internal point of the chord defining H (we denote these half-polygons by $\mathcal{H}_1$), then $d(x, H) = d_2(x, \bar{H})$ for all $x \in R$, where $\bar{H}$ is the half-plane defined by H. And if the first segment of $\pi(x, H)$ reaches a vertex u (we denote these half-polygons by $\mathcal{H}_2$), then $d(x, H) = d_2(x, u) + \kappa$ for all $x \in R$, where κ is a constant. Thus we seek a point $x = (x_1, x_2)$ and a value ρ to solve:

$$\begin{array}{lll} \text{minimize} & \rho & \\ \text{subject to} & d_2(x, \bar{H}) \leq \rho & H \in \mathcal{H}_1 \\ & d_2(x, u) + \kappa \leq \rho & \text{for point } u \text{ and constant } \kappa \text{ corresponding to } H \in \mathcal{H}_2 \end{array}$$

Because x is guaranteed to lie in the region R, we can completely disregard the underlying polygon P in solving the problem.

In the Euclidean plane, the problem may be reinterpreted in a geometric manner. We wish to find the disk of smallest radius ρ that *intersects* each of a given set of half-planes and *contains* each of a given set of disks. For $H \in \mathcal{H}_1$, we have $d(x, H) \leq \rho$ if and only if the disk of radius ρ centered at x intersects $\bar{H}$. For $H \in \mathcal{H}_2$, with $d(x, H) = d_2(x, u) + \kappa$, we have $d(x, H) \leq \rho$ if and only if the disk of radius ρ centered at x contains the disk of radius κ centered at u. We will call this Euclidean problem the "minimum feasible disk" problem. The constraints of the problem that correspond to the set of half-polygons $\mathcal{H}_1$ will be referred to as ***half-plane constraints***, while the constraints for $\mathcal{H}_2$ will be called ***disk constraints***.

We observe here that the minimum feasible disk problem belongs to the class of 'LP-type' problems described by Sharir and Welzl [93]. In fact, it satisfies the computational assumptions that allow a derandomization of the Sharir-Welzl algorithm yielding a *deterministic* linear-time algorithm for the problem (see Chazelle and Matousek [27]). However, as this approach is rather complex, we will outline a more direct linear-time algorithm to solve the problem.

The minimum feasible disk problem is a combination of two well-known problems that have linear time algorithms. If all the constraints are half-plane constraints, then, because each such constraint can be written as a linear inequality, we have a 3-dimensional linear program, which can be solved in linear time as shown by Meggido [78] and independently by Dyer [39]. On the other hand, if all the constraints are disk constraints, then this is the "spanning circle problem"—to find the smallest disk that contains some given disks. This problem arose from the geodesic vertex center problem [91] and generalizes the Euclidean 1-center problem where the disks degenerate to points. The problem was solved in linear time by Megiddo [80] using an approach similar to that for the 1-center problem and for linear programming. Because the approaches are similar, it is not difficult to combine them, as we show below.

As described in Section 4.2, the goal is to spend linear time to prune away a constant fraction of the constraints that do not define the final answer, and to repeat this until there are only a constant number of constraints left, after which a brute force method may be employed. We pair up the constraints, and for each pair of constraints c_1, c_2 compute a "bisecting" plane Π such that on one side of Π the constraint c_1 is redundant, and on the other side of Π the constraint c_2 is redundant. If we could identify which side of Π contains the optimum solution, then one of the constraints c_1, c_2 can be removed. We address the existence of such bisecting planes below. There are two other issues. Issue 1 is to identify which side of a plane contains the optimum solution, a subproblem that Megiddo calls an "oracle". This is done by finding the optimum point restricted to the plane (a problem one dimension down), from which the side of the plane can be decided.

(The Chord Oracle from Section 5.3.1 was doing a similar thing.) Issue 2 is to identify the position of the optimum point relative to "many" of the bisecting planes, while testing only a "small" sample of them. We will not discuss these two issues since they are the same as in Megiddo's papers [78, 79] (or see the survey by Dyer et al. [38]). Alternatively, one can use the concept of cuttings described in Lemma 9 to obtain the 'small' set of hyperplanes to run the oracle upon.

For our minimum feasible disk problem, we have two types of constraints—half-plane constraints and disk constraints. Megiddo's prune-and-search approach based on pairing up the constraints can still be applied so long as we pair each constraint with another constraint of the same type. (We note that this idea was previously used by Bhattacharya et al. [17] in their linear time algorithm to find the smallest disk that contains some given points and intersects some lines, a problem they call the "intersection radius problem".) Thus it suffices to describe what are the bisecting planes for the two types of constraints in our minimum feasible disk problem.

A half-plane constraint has the form $d_2(x, \bar{H}) \leq \rho$. If the halfplane $\bar{H}$ is given by $a_i^T x \leq b_i$, normalized so that $||a_i|| = 1$, then the constraint is $a_i^T x \leq b_i + \rho$, a linear inequality. For two such constraints indexed by i and j, the bisecting plane is given by $(a_i^T - a_j^T)x - (b_i - b_j) = 0$.

A disk constraint has the form $d_2(x, u_i) + \kappa_i \leq \rho$, corresponding to a disk with center u_i and radius κ_i. As Megiddo [80] noted, by adding the constraint $\rho \geq \kappa_i$, this can be written as

$$||x - u_i||^2 \leq (\rho - \kappa_i)^2$$

or as

$$f_i(x, \rho) \leq 0$$

where f_i is defined as

$$f_i(x, \rho) = ||x||^2 - 2u_i^T x + ||u_i||^2 - \rho^2 + 2\kappa_i \rho - \kappa_i^2.$$

This is not a linear constraint, but for $i \neq j$, the equation $f_i(x, \rho) = f_j(x, \rho)$ defines a plane since the quadratic terms, $||x||^2$ and ρ^2, cancel out. So the bisecting plane is $f_i(x, \rho) = f_j(x, \rho)$.

This completes the summary of how to solve the minimum feasible disk problem in linear time, and completes our algorithm to find the geodesic center of half-polygons.

5.4 Conclusions

In this chapter, we introduced the notion of the visibility center of a set of points in a polygon and gave an algorithm with run time $O((n+m)\log(n+m))$ to find the visibility center of m points in an n-vertex polygon (Theorem 10). To do this, we gave an algorithm with run time $O((n+k)\log(n+k))$ to find the geodesic center of a given set of k half-polygons inside a polygon (Theorem 11), a problem of independent interest.

Observe that the algorithm for the geodesic center of half-polygons (that was described in Section 5.3) also solves the problem of locating the geodesic center of the edges of a simple polygon. The algorithm for the 'edge center' for an n-vertex polygon takes $O(n \log n)$ time if we use the ideas from this chapter. In the next chapter, we show how to determine the geodesic edge center in linear time.

Open Problems. The following are some interesting open problems to consider:

1. Can the visibility center of a simple polygon be found more efficiently? Note that the geodesic center of the vertices of a simple polygon can be found in linear time [4]. Our current method involves ray shooting and sorting (Section 5.2 and the preprocessing in Section 5.3) , which are serious barriers. A more reasonable goal is to find the visibility center of m points in a polygon in time $O(n + m \log m)$.

2. How hard is it to find the farthest visibility Voronoi diagram of a polygon—a subdivision of the polygon into regions with the same farthest site with respect to distance to visibility? Finally, what about the 2-visibility center of a polygon, where we can deploy two guards instead of one?

3. In presentations of the results of this chapter, John Hershberger asked us the following interesting question: out of the m points to be guarded, suppose that some are more important than the others. This motivates the problem of finding the *weighted visibility center*, where weights (multiplicative or additive) are assigned to the distances to visibility to the m points. The objective is to find a location for the guard so as to minimize the maximum weighted distance to visibility.

Chapter 6

The Geodesic Edge Center of a Simple Polygon[1]

> All parts should go together without forcing. Therefore, if you can't get them back together again, there must be a reason.
> By all means, do not use a hammer.
>
> IBM maintenance manual 1975

The most basic "center" problem is Sylvester's problem was introduced in Chapter 1: given n points in the plane, find the smallest disc that encloses the points. The center of this disc is a point that minimizes the maximum distance to any of the given points. We consider a center problem that differs in two ways from Sylvester's problem. First, the domain is a simple polygon and the distance measure is not Euclidean distance, but rather the shortest path, or "geodesic" distance inside the polygon. Second, the sites are not points but rather the edges of the polygon. More precisely, the problem is to find, given a simple polygon in the plane, the ***geodesic edge center*** which is a point in the polygon that minimizes the maximum geodesic distance to a polygon edge. See Figure 6.1. More formally, let E be the set of edges of the polygon P, and for point $p \in P$ and edge $e \in E$,

[1]Based on joint work with Anna Lubiw. Recently published in the paper 'The Geodesic Edge Center of a Simple Polygon' at the Symposium on Computational Geometry (SoCG), 2023

define $\boldsymbol{d(p, e)}$ to be the geodesic distance from p to e. Define the ***geodesic radius*** of a point $p \in P$ to be $r(p) := \max\{d(p, e) : e \in E\}$. Then the ***geodesic edge center*** is a point $p \in P$ that minimizes $r(p)$.

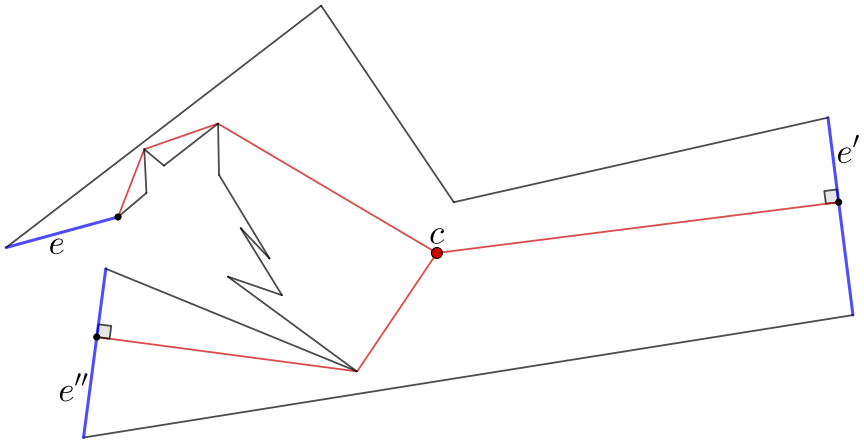


Figure 6.1: The edge center for the above polygon is the point c. Edges e, e', e'' (in blue) are geodesically farthest from c—the geodesic paths (in red) from c to these edges all have the same length.

For a simple polygon that satisfies some general position assumptions (described in detail in Section 6.2.2), we prove the following:

Theorem 23. *There is a linear time algorithm to determine the geodesic edge center of a simple polygon.*

This improves the $O(n \log n)$ time algorithm for the edge center given in Chapter 5 (where we described the solution to the more general problem of locating the geodesic center of half-polygons). The algorithm follows the strategies used to find the geodesic *vertex* center, which is a point in the polygon that minimizes the maximum geodesic distance to a polygon vertex. Our algorithm follows the approach of Ahn et al., modified to deal with edges rather than vertices. We simplify some aspects and we repair some errors in their approach. Further explanations of previous results and of our algorithm can be found below. The edge center problem is more general than the vertex center problem via the reduction of splitting each vertex into two vertices joined by a very short edge. From any point in the polygon, one of the two endpoints of an edge is farther away than the edge itself. The transformed problem ignores the original edges and only takes into account the newly created 'short' edges. Therefore, the geodesic edge center of the transformed problem coincides with the geodesic center of the original problem.

In general, the center of a set of sites can be determined from the farthest Voronoi diagram of those sites, but computing the Voronoi diagram can be more costly. As the first step of our center algorithm we give a linear time algorithm to compute the geodesic farthest edge Voronoi diagram restricted to the boundary of the polygon. It is an open question to compute the entire geodesic farthest edge Voronoi diagram in deterministic linear time, so our algorithm for the geodesic edge center proceeds differently.

Techniques and Our Contributions. Our algorithm to find the geodesic edge center of a polygon has two phases. In the first phase we find the geodesic farthest edge Voronoi diagram restricted to the polygon boundary. In the second phase we reduce the problem to finding a point in the polygon that minimizes the upper envelope of a linear number of easy-to-compute convex functions each defined on a triangle inside the polygon. That problem is then solved using a divide-and-conquer algorithm based on ϵ-nets and Megiddo's prune and search techniques. A more detailed overview of our algorithm can be found in Section 6.1, but since our algorithm follows and builds upon a number of previous results, we first summarize the previous results (in historical order) and explain what is novel about our contributions.

1. Meggido, 1983 [78] and Dyer, 1984 [39]. A linear time algorithm for linear programming in two and three dimensions. There are two ideas from these famous prune-and-search algorithms that are used in algorithms for geodesic centers, including ours. The first idea is to make repeated calls to an "oracle" that solves the problem one dimension down. The second idea is to eliminate constraints by pairing them up and considering the "bisecting" plane or line where the two constraints are equal. The oracle can be used to determine which side of the bisector contains the optimum solution, thus eliminating one of the two constraints. Using techniques now known as cuttings or epsilon-nets, a small number of bisectors can be tested to eliminate a large number of constraints, resulting in a linear time algorithm.

2. Pollack, Sharir, and Rote, 1989 [91]. An $O(n \log n)$ time algorithm to find the geodesic vertex center of a simple polygon. As for the linear programming algorithms described above, a main ingredient is to solve the problem one dimension down. In particular, they develop an $O(n)$ time "chord oracle" that, given a chord of the polygon, finds the *relative center* restricted to the chord and from that, determines whether the center of the polygon lies to left or right of the chord. By applying the chord oracle $O(\log n)$ times, they limit the search to a subpolygon where Euclidean distances can be used. This reduces the problem to finding a minimum disc that encloses some disks, which Megiddo [80] solved in linear time using the same approach as described

above for linear programming. Lubiw and Naredla [72] (as described in Chapter 5) redid the chord oracle to handle farthest *edges* instead of vertices.

The idea used in the chord oracle algorithm is central to further developments. Expressed in general terms, the goal is to find a point in a domain (either a chord or the whole polygon) that minimizes the maximum distance to a site (a vertex or edge of the polygon). The idea is to first find what we will call a *coarse cover* of the domain by a linear number of elementary regions R (intervals or triangles) with an easy-to-compute convex function f_R defined on each region R, such that for any point x in the domain, the maximum distance from x to a site is the maximum of $f_R(x)$ over regions R containing x. Thus, the goal is to find the point x that minimizes the upper envelope of the convex functions. When the domain is a chord, the chord oracle finds the point x in linear time using Megiddo's technique of pairing the constraints (i.e., the functions) in order to prune away a fraction of them in each round. When the domain is the whole polygon, the innovation of Ahn et al. described in (4) below is to use similar ideas, together with ϵ-net techniques, to obtain a linear time algorithm. However, finding an appropriate coarse cover of the polygon in linear time depends on another breakthrough, which we discuss next.

3. Hershberger and Suri, 1997 [56]. A linear time algorithm to find the farthest vertex from each vertex of a polygon. This important result is the cornerstone of linear time algorithms to find the geodesic diameter and center of a polygon. The problem is transformed to finding row maxima in a matrix of distances between pairs of vertices. The resulting matrix is "totally monotone" so the row maxima can be found with a linear number of matrix accesses using the algorithm of Aggarwal et al. [2]. Hershberger and Suri show how to access the required distance matrix entries in amortized constant time each.

   We show that Hershberger and Suri's algorithm extends to finding the farthest *edge* from each vertex.

4. Ahn, Barba, Bose, De Carufel, Korman, and Oh, 2016 [4]. A linear time algorithm to find the geodesic vertex center of a simple polygon. Using the algorithm of Hershberger and Suri as a starting point, they find a coarse cover of the whole polygon. They then use divide-and-conquer based on ϵ-nets to reduce the domain to a triangle. After that, the vertex center is found using Megiddo-style prune-and-search techniques as described in (1) above.

   Our algorithm uses a similar approach. One difference is that we give a simpler method of finding a coarse cover of the polygon by first finding the geodesic farthest edge Voronoi diagram on the polygon boundary.

Other differences are introduced in order to correct some flaws in the algorithm of Ahn et al. They recurse on subpolygons called "4-cells" that are the intersection of four half-polygons (a half-polygon is the subpolygon to one side of a chord). However, the resulting range space does not in fact have the necessary properties for finding ϵ-nets, if a deterministic algorithm is desired. In addition, their step of partitioning a larger cell into 4-cells is incomplete.

We remedy these issues by recursing on subpolygons that are geodesic hulls of three points or boundary chains of the polygon. We can then prove the requisite ϵ-net properties, and we can partition larger cells into our subpolygons. Our approach works both for the geodesic edge center and for the geodesic vertex center, thus generalizing the result of Ahn et al. For more details, see Section 6.5.

5. Oh, Barba, and Ahn, 2020 [87]. This paper uses the coarse cover of the polygon from (4) above, and finds the farthest vertex Voronoi diagram inside a polygon, although not in linear time. Of relevance here is their first step, which is a linear time algorithm to find the Voronoi diagram restricted to the polygon boundary. We reverse the order of dependence of (4) and (5). First, using the Hershberger-Suri result (3) and the coarse cover of an edge from (6) below, we give a considerably simpler linear time algorithm to find the farthest edge/vertex Voronoi diagram restricted to the polygon boundary—a problem of independent interest. After that, we use the boundary Voronoi diagram to produce a coarse cover of the polygon and then proceed to find the geodesic edge center.

6. Lubiw and Naredla [72] (covered in Chapter 5). An $O(n \log n)$ time algorithm to find the edge center of a simple polygon, based on the algorithm of Pollack et al. described in (1) above. In fact, our algorithm solves the more general problem of finding the center of a set of $O(n)$ half-polygons (edges being the special case where the half-polygons hug the polygon boundary). One ingredient that we will reuse in the present chapter is a linear time chord oracle for the edge center, including finding a coarse cover of a chord.

6.1 Overview of the Algorithm

We proceed to give an overview of the linear-time algorithm for the geodesic edge center. We split the algorithm into two phases:

- **Phase I (The boundary phase)**: Find the farthest-edge Voronoi diagram restricted to ∂P, the boundary of P.

- **Phase II (The search phase)**: Use the restricted Voronoi diagram constructed in the previous phase to find the geodesic edge center.

6.1.1 Phase I (The boundary phase)

In this phase, we construct the geodesic farthest-edge Voronoi diagram restricted to the polygon boundary. We split this into two parts:

Part 1: Finding the Farthest Edge from each Vertex (Section 6.3): We begin by finding some partial information, namely, the farthest edge from each vertex. We show that the algorithm of Hershberger and Suri [56] that finds the farthest *vertex* from each vertex can be modified to find the farthest edge from each vertex. Modifying their algorithm requires some groundwork. Specifically, we need the properties of farthest edges and "separators" developed in Section 6.2.3 and Section 6.2.5.

Part 2: Finding the Farthest Edge Voronoi Diagram Restricted to the Polygon Boundary (Section 6.4): In this part of the boundary phase, we fill in the remaining gaps for the farthest edge Voronoi diagram on the boundary. For a polygon edge with the same farthest edge at both endpoints, every point on the polygon edge has the same farthest edge (Section 6.2.3). For a polygon edge e with different farthest edges at the endpoints—called a "transition" edge—the Voronoi diagram construction requires some work. The idea is to find a coarse cover of transition edge e by intervals, each associated with a simple convex function that captures the correct geodesic distance to a potential farthest edge. The upper envelope of the coarse cover functions gives the farthest Voronoi diagram on e. Several ingredients are needed to make this efficient. The coarse cover of e is constructed from shortest path trees inside a smaller subpolygon called the "hourglass" of e (Section 6.4.1). The tree structure then allows us to incrementally update the upper envelope as we add coarse cover functions (Section 6.4.2).

6.1.2 Phase II (The search phase)

In this phase, we perform a search for the geodesic edge center inside P. The basic plan (described in more detail in Section 6.5) is to first find a coarse cover of the polygon by triangles, each with an associated convex function that captures that correct geodesic distance to a potential farthest edge. The edge center is then the point that minimizes the upper envelope of these functions. To find this point we use divide-and-conquer, reducing in each step to a smaller subpolygon with a constant fraction of the coarse cover elements.

Once the subpolygon is a triangle, the prune-and-search approach of Megiddo's can be applied. Until that point in time, every coarse cover triangle has a boundary chord crossing the subpolygon, and ϵ-net techniques are used to reduce the number of such chords, and hence the number of coarse cover elements. The presentation is split into four parts:

Part 1: Finding a Coarse Cover of the Polygon by Triangles (Section 6.6): In this part, we find a linear-size coarse cover of P using triangles. Section 6.2.6 describes coarse covers in more detail. Each triangle is associated with a simple convex function that captures the correct geodesic distance to a potential farthest edge. The potential farthest edges we need are the edges that have non-empty Voronoi regions on the boundary of P. For each such edge, we construct coarse cover triangles by computing shortest paths from the edge to its Voronoi region on the boundary of P. We prove the defining property of a coarse cover—that for any point x in the domain, the maximum distance from x to a site is the maximum function value over triangles containing x. Thus the problem of finding the edge center is reduced to the problem of finding the point that minimizes the upper envelope of this coarse cover.

Part 2: Divide-and-conquer via ϵ-nets (Section 6.7): We next lay the groundwork for a divide-and-conquer algorithm based on ϵ-nets (reviewed in Section 4.3). Our approach follows that of Ahn et al. [4] but, as mentioned in the previous section, we fill in some gaps. We use "3-anchor hulls" as the subpolygons we recurse on, rather than the "4-cells" of Ahn et al. The subpolygons define a range space whose ground set is the set of polygon chords that bound triangles of the coarse cover, and whose ranges are sets of chords that cross a 3-anchor hull. We prove that our range space has finite VC-dimension, and even more crucially, we give a "subspace oracle" which permits an ϵ-net to be found in *deterministic* linear time, unlike with the 4-cell approach.

Part 3: Geodesic Oracle (Section 6.8): In this section we extend the chord oracle to a *geodesic oracle* that tells us which side of a geodesic path contains the edge center. This is one of the tools we need in the final algorithm.

Part 4: Finding the Geodesic Edge Center (Section 6.9): This last part determines the location of the edge center, which is the point that minimizes the upper envelope of the coarse cover functions. We use divide-and-conquer to narrow the search down to a smaller subpolygon while reducing the number of elements in the coarse cover. In Stage 1 (Section 6.9.1) no triangle of the coarse cover contains the subpolygon, so they all have a chord crossing the subpolygon, and we use ϵ-net techniques and the geodesic oracle to reduce to a smaller subpolygon and eliminate a constant fraction of the crossing chords (and associated coarse cover elements). After Stage 1 we easily reduce to the case where

the subpolygon is a triangle. Then in Stage 2 (Section 6.9.2) we use a modification of the prune-and-search methods of Megiddo [78] (see Section 4.2) to finally determine the edge center of the polygon.

6.2 Preliminaries

We discussed a few preliminary concepts required to define the edge center in the introduction. This section discusses a few other concepts that will be useful in describing and understanding our algorithm. Section 6.2.1 introduces some notation that is used throughout this chapter—notation specific to a section is introduced as needed. In Section 6.2.2, we discuss the general position assumptions on the input. Such assumptions are common in computational geometry and allow for a simpler description of the algorithm. In the preliminary phase of our algorithm, we make repeated use of the properties of farthest edges from points on the polygon boundary. These properties are stated and proved in Section 6.2.3. In Section 6.2.4, we describe a simple extension of the fundamental notion of shortest path trees and maps to the case when geodesic distances to/from edges (rather than vertices) are desired. The notion of separators, introduced in Section 6.2.5, is a crucial tool for finding the farthest edge from each vertex. The final subsection of the preliminaries is Section 6.2.6. It introduces the chord oracle that allows us to apply the divide-and-conquer paradigm to the geodesic edge center problem. It also introduces the idea of the coarse cover that helps us unify the description of our algorithm on the boundary as well as in the interior of the polygon.

6.2.1 Notation and Definitions

We refer to Chapter 2 for the definitions and notation for standard concepts from computational geometry. Some of these were also used in Chapter 5.

The geodesic farthest Voronoi diagram of a set of point sets in a simple polygon was defined in Chapter 2. When the point sets under consideration are the edges of the polygon P, we refer to the diagram as the ***geodesic farthest-edge Voronoi diagram***, denoted $\mathcal{V}(P)$. The restriction of the diagram to the boundary ∂P of the polygon will be denoted $\mathcal{V}(\partial P)$.

6.2.2 General Position Assumptions

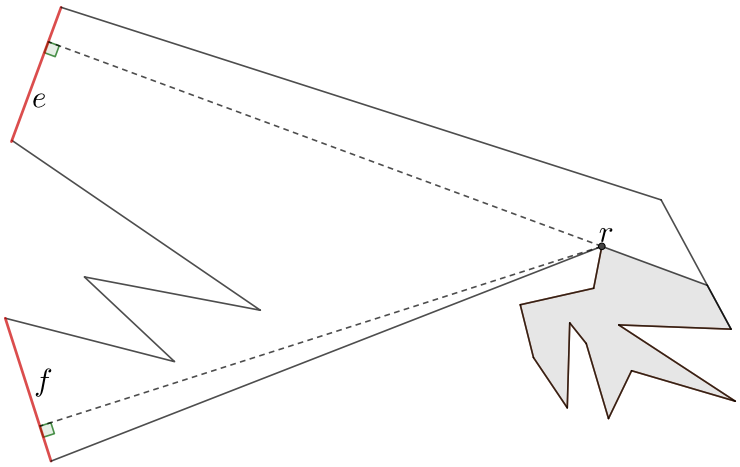


Figure 6.2: Vertex r is equidistant from edges e and e' (colored red), and so is every point in the shaded region. The bisector between the two edges is thus two-dimensional. Assumption 2 forbids this situation.

We make some general position assumptions on the input to simplify the construction of the farthest-edge Voronoi diagram on the boundary. General position assumptions are commonly used in computational geometry to deal with degeneracy in the input. Previous algorithms on geodesic vertex centers and the corresponding Voronoi diagrams [4, 10, 87, 102] used assumptions of a similar flavour. In actual implementations, general position is usually achieved by an infinitesimal perturbation of the input points [41].

Our first assumption is common and basic.

Assumption 1. *No three vertices of the simple polygon P are collinear.*

Our next assumption deals with points whose farthest edge is not unique. For points in the plane, the farthest point Voronoi diagram has: 2-dimensional faces, where the farthest point is unique; 1-dimensional edges, where there is a two-way tie for farthest points; and 0-dimensional vertices, where there are three or more farthest points. We will perturb the polygon vertices to ensure that our geodesic farthest edge Voronoi diagrams have similar properties. In particular, we would like to avoid the situation shown in Figure 6.2 where points in a 2D region have two farthest edges—this situation arises when a vertex (r in the figure) is equidistant from two edges.

We first need a tie-breaking rule because there is a situation that cannot be avoided by perturbing vertices, namely, when a vertex v has two equidistant edges e and f such

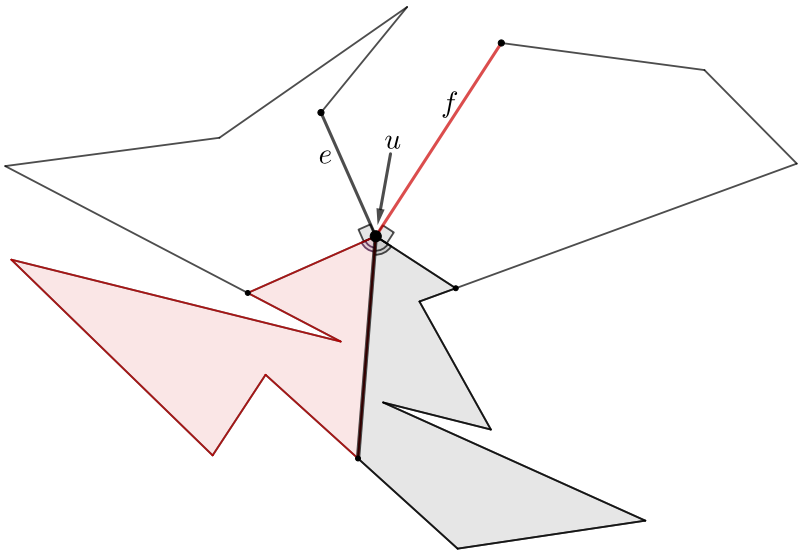


Figure 6.3: From points in the shaded region (both black and red), the geodesic paths to the edges e and f coincide. Points on the angle bisector at u are considered equidistant from e and f. For a point p in the region shaded red (black, resp.), the tie-breaking rule gives $d(p, f) > d(p, e)$ ($d(p, e) > d(p, f)$, resp.). Note that the tie-breaking rule does not apply to the unshaded regions of the polygon.

that the shortest paths $\pi(v, e)$ and $\pi(v, f)$ are the same—which implies that e and f have a common reflex vertex u with $\pi(v, e) = \pi(v, f) = \pi(v, u)$. Here the path itself is unique but the terminal point is on two edges. We break this tie as follows (in a manner similar to Aurenhammer et al. [13]).

Tie-Breaking Rule. Suppose that p is a point of P, e and f are two edges that meet at reflex vertex u, and $\pi(p, e) = \pi(p, f) = \pi(p, u)$. Let line b be the angle bisector of u. For p not on b, break the tie $d(p, e) = d(p, f)$ by saying that the distance to the edge on the opposite side of b is greater. See Figure 6.3.

We make the following additional assumptions, which we claim can be effected by perturbing vertices.

Assumption 2. *After imposing the tie-breaking rule, no vertex is equidistant from two or more edges.*

Based on the Tie-Breaking rule and Assumption 2 we can prove that the set of points with a tie for farthest edge behave nicely.

Lemma 24. *Let D be the set of points in P with more than one farthest edge (after imposing the tie-breaking rule). Then D does not contain a 2-dimensional ball, does not contain a vertex of P, and intersects ∂P in isolated points.*

Proof. It suffices to show that the conditions hold for the set of points equidistant from two edges e and f. Let p be a point with $d(p, e) = d(p, f)$. The paths $\pi(p, e)$ and $\pi(p, f)$ do not share a vertex other than the terminal point, by Assumption 2—in particular, p cannot be a vertex.

If $\pi(p, e)$ and $\pi(p, f)$ share a terminal vertex u, then the Tie-Breaking Rule would apply unless p is on the bisector of the angle at u, which is 1-dimensional and intersects ∂P in a single point because no three vertices are collinear by Assumption 1.

Otherwise, the paths $\pi(p, e)$ and $\pi(p, f)$ diverge at p. Let s_e be the first vertex on the path $\pi(p, e)$—or let $s_e = e$ in case there are no vertices. Define s_f similarly. Then p must be on the weighted bisector between s_e and s_f, which is 1-dimensional, and and intersects ∂P in isolated points. □

We note that Assumption 2 can be effected by perturbing vertices, since, in the $2n$-dimensional space of allowed vertex perturbations, the configurations we must avoid are lower-dimensional.

Finally, to ensure that Voronoi vertices are "nice", we make the following assumption.

Assumption 3. *No point on the polygon boundary has more than two farthest edges. No point in the interior of the polygon has more than a constant number of farthest edges.*

6.2.3 Properties of Farthest Edges

In this section we give some basic properties of shortest paths from points on ∂P to their farthest edges in a polygon, with a focus on when and how such paths cross—more formally, we examine the ordering of the points and their farthest edges around the polygon boundary.

Standard (closest) Voronoi diagrams have the property that if you take two points and the paths to their closest sites, then the paths do not cross. However, for farthest Voronoi diagrams such paths usually do cross.

Let p and q be two points on the polygon boundary. Let $F(p)$ be a farthest edge from p and let $F(q)$ be a farthest edge from q. Note that these farthest edges need not be unique

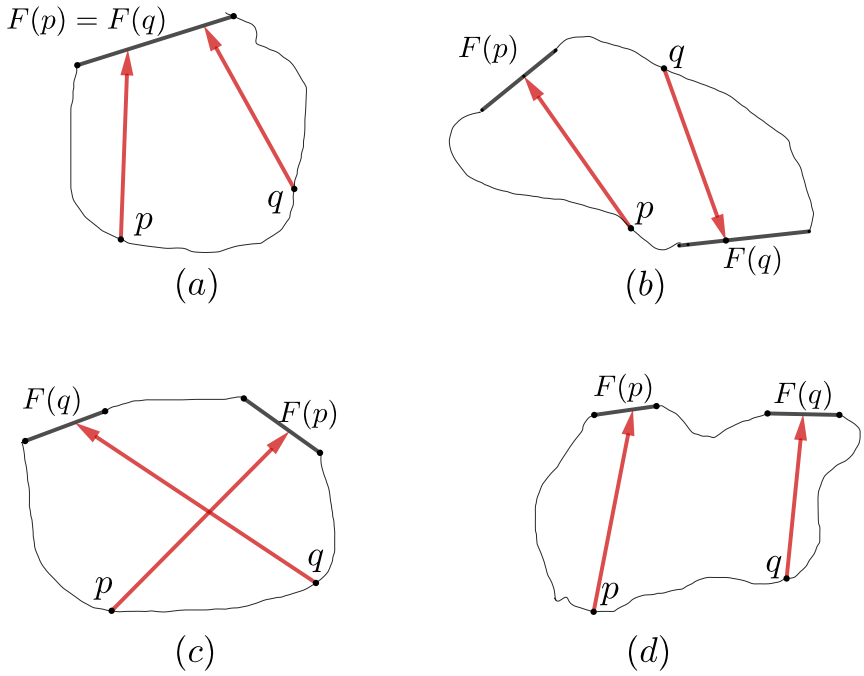


Figure 6.4: Schematics for possible and impossible orderings of points p, q and their farthest edges $F(p)$ and $F(q)$. (a) The only possible ordering if $F(p) = F(q)$. (b),(c) The two possible orderings if $F(p) \neq F(q)$. (d) The impossible ordering if $F(p) \neq F(q)$.

since there are (isolated) points on the polygon boundary with two farthest edges. Let $\pi_p = \pi(p, F(p))$ and let $\pi_q = \pi(q, F(q))$. Let t_p be the terminal point of the path π_p and let t_q be the terminal point of the path π_q. If $F(p) \neq F(q)$, we say that π_p and π_q ***cross*** if the ordering around the polygon boundary (in either clockwise or counterclockwise order) is $p, q, F(p), F(q)$. Note that this allows the possibility that $t_p = t_q$ at a reflex vertex. If $F(p) = F(q)$, we say that π_p and π_q ***cross*** if the ordering around the polygon boundary (in either clockwise or counterclockwise order) is p, q, t_p, t_q, with $t_p \neq t_q$.

Lemma 25. *With the above setup, the paths π_p and π_q have the following properties.*

(P1) *If $F(p) = F(q)$, then π_p and π_q do not cross, i.e., the ordering of points around the boundary of P is p, q, t_q, t_p, possibly with $t_q = t_p$ if the paths merge. (See Figure 6.4(a)).*

(P2) *If $F(p) \neq F(q)$ then the possible orderings are: $p, F(p), q, F(q)$ (see Figure 6.4(b)); or $p, q, F(p), F(q)$, i.e., the paths cross (see Figure 6.4(c)). Equivalently, the only other ordering, $p, q, F(q), F(p)$, cannot occur (see Figure 6.4(d)).*

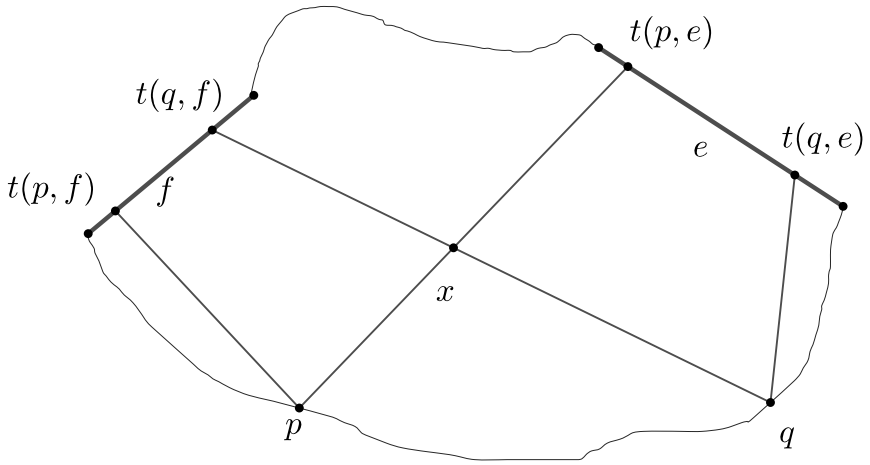


Figure 6.5: Illustration for Lemma 26.

(P3) The Ordering Property. *As p moves clockwise around ∂P, so does $F(p)$.*

(P4) *If the paths π_p and π_q cross, then they do not share a directed polygon chord. The paths may cross at a vertex or at internal points of chords. They may share a chord in opposite directions.*

The Ordering Property **(P3)** was proved by Aronov et al. [10] for the case of farthest vertices. To prove Lemma 25, we will use (as they did) the following basic "triangle inequality" about the lengths of two shortest paths to edges when the paths cross. Later on, we will appeal not only to the Ordering Property and the other parts of Lemma 25, but also to the triangle inequality, since it applies more generally to paths that do not go to farthest edges. In fact, even for the basic problem of finding the farthest vertex from each vertex in a convex polygon, the triangle property is the key to linear time algorithms. The first such linear time algorithm was given by Aggarwal et al. [3] using a technique called matrix searching in a totally monotone matrix. They point out that assuming only the Ordering Property, there is a super-linear lower bound on the time to find all farthest vertices. However, the triangle inequality implies a "totally monotone" matrix and allows a linear time algorithm. The matrix searching technique is discussed further in Section 6.3.

Lemma 26. *Suppose the points p, q and the edges e, f occur in the order p, q, e, f along the polygon boundary ∂P. Then, $d(p,e) + d(q,f) \geq d(p,f) + d(q,e)$.*

Proof. Observe that due to the ordering of p, q, e, f on ∂P, the paths $\pi(p,e)$ and $\pi(q,f)$

must have a common point which we label x. See Figure 6.5. Then:

$$
\begin{aligned}
& d(p,e) + d(q,f) \\
&= d(p, t(p,e)) + d(q, t(q,f)) && \text{[Distance to an edge is distance to the terminal]} \\
&= d(p,x) + d(x, t(p,e)) + d(q,x) + d(x, t(q,f)) \\
&= (d(p,x) + d(x, t(q,f))) + (d(q,x) + d(x, t(p,e))) \\
&\geq d(p, t(q,f)) + d(q, t(p,e)) && \text{[Triangle Inequality]} \\
&\geq d(p, t(p,f)) + d(q, t(q,e)) && \text{[Definition of a terminal]} \\
&= d(p,f) + d(q,e) && \text{[Distance to an edge is distance to the terminal]}
\end{aligned}
$$

□

Corollary 27. *Under the same assumptions, if $d(p,f) > d(p,e)$, then $d(q,f) > d(q,e)$.*

Proof of Lemma 25.

(P1). Suppose the ordering is p, q, t_p, t_q. Then the paths must have a common point x. The shortest path from x to the edge $F(p) = F(q)$ is unique, so the paths π_p and π_q are the same after x. Thus the ordering is p, q, t_q, t_p, possibly with $t_q = t_p$.

(P2). We must show that the ordering $p, q, F(q), F(p)$ cannot occur. Suppose it does. First note that if $t_p = t_q$ then the tie-breaking rule would not allow the ordering $p, q, F(q), F(p)$. Thus we may assume that $t_p \neq t_q$.

Since $F(p)$ is a farthest edge from p, $d(p, F(p)) \geq d(p, F(q))$. Since $F(q)$ is a farthest edge from q, $d(q, F(q)) \geq d(q, F(p))$. By Lemma 26, $d(p, F(q)) + d(q, F(p) \geq d(p, F(p)) + d(q, F(q))$. Therefore $d(p, F(p)) = d(p, F(q))$ and $d(q, F(q)) = d(q, F(p))$.

For the case described above, we have the following claim:

Claim 28. *Both p and q have $F(p)$ and $F(q)$ as tied farthest edges.*

Proof. The distances are the same but we must be careful about the tie-breaking rule. If the tie-breaking rule applies to $\pi(p, F(p))$ and $\pi(p, F(q))$, then these two paths both terminate at a reflex vertex u common to $F(p)$ and $F(q)$, but in this case $\pi(q, F(p))$ must also terminate at u (since it cannot cross $\pi(p, F(p))$). Then $\pi(q, F(q))$ also terminates at u, since we cannot have two equal-length paths from q to different points on edge $F(q)$. Thus the original paths π_p and π_q terminate at the same point, which we already ruled out. □

We claim that any point r that lies on ∂P between p and q also has $F(p)$ and $F(q)$ as farthest edges. Consider any edge e that lies on the polygon chain from r to $F(q)$ (the part containing p). Note that $F(p)$ is one such edge. Applying Lemma 26 to $q, r, e, F(q)$, gives $d(r, F(q)) + d(q, e) \geq d(r, e) + d(q, F(q))$. Since $d(q, F(q)) \geq d(q, e)$ this implies $d(r, F(q)) \geq d(r, e)$. In particular, $d(r, F(q)) \geq d(r, F(p))$. A symmetric argument shows that $d(r, F(p)) \geq d(r, e)$ for any edge e that lies on the polygon chain from r to $F(p)$ (the part containing q). In particular, $d(r, F(p)) \geq d(r, F(q))$. Since all edges e are included in the two ranges, this proves that r has $F(p)$ and $F(q)$ tied for farthest edge. (We can again show that the tie-breaking rule does not apply.) By Lemma 24, only isolated points on ∂P can have $F(p)$ and $F(q)$ tied for farthest edge. Therefore the ordering $p, q, F(q), F(p)$ cannot occur.

(P3). Consider a point p with a farthest edge $F(p)$ and let q be the first point after p moving clockwise around ∂P that has a farthest edge $F(q)$ that is not a farthest edge from p. Note that q comes before $F(p)$. Since the ordering $p, q, F(q), F(p)$ is prohibited, $F(q)$ must lie after $F(p)$ in clockwise order.

(P4). Suppose π_p and π_q cross. We first suppose that $t_p = t_q$. Then the terminal point is a reflex vertex u common to $F(p)$ and $F(q)$. If the paths share a directed chord (a, b), then the paths are identical after vertex a and therefore identical on their last segment which is a chord from some vertex v to u. The tie-breaking rule would not allow p and q to have farthest edges $F(p)$ and $F(q)$ unless v lies on the bisector of u which is excluded by Assumption 2.

Thus we may assume that $t_p \neq t_q$ and the paths share the directed chord (a, b). Consider the portion of π_p from b to $F(p)$ and the portion of π_q from b to $F(q)$. Those are both shortest paths. Because of the general position Assumption 2 (note that b is a vertex), one of the paths must be longer, say the one to $F(q)$. Now we claim that $d(p, F(q)) > d(p, F(p))$, contradiction to $F(p)$ being a farthest edge from p. To show this, construct a path σ from p to $F(q)$ by following π_p from p to a, then traversing chord (a, b), then following π_q from b to $F(q)$. Then σ is longer than π_p. We are done if we can show σ is a shortest path. But the part up to b is locally shortest and the part after a is locally shortest, thus none of the bends in the path can be shortened, so σ is a geodesic path and thus a shortest path. □

6.2.4 Shortest Paths To/From Edges

As basic tools, we need a linear time algorithm to find shortest paths from a given point to all edges of the polygon, and a linear time algorithm to find shortest paths from a

given edge to all vertices of the polygon. In fact, in both cases, we will augment to a ***shortest path map*** that divides the polygon into regions (triangles and trapezoids) in which the shortest paths are combinatorially the same. These augmentation algorithms are not difficult, but we did not find them in the literature.

Shortest Paths from a Point to all Edges and Vertices. For a point p in polygon P, define $\boldsymbol{T_p}$ be the ***shortest path tree*** that consists of shortest paths from p to all the edges and vertices of the polygon. In some situations we will only care about the shortest paths to edges, but we will still use the notation T_p and just clarify what we mean.

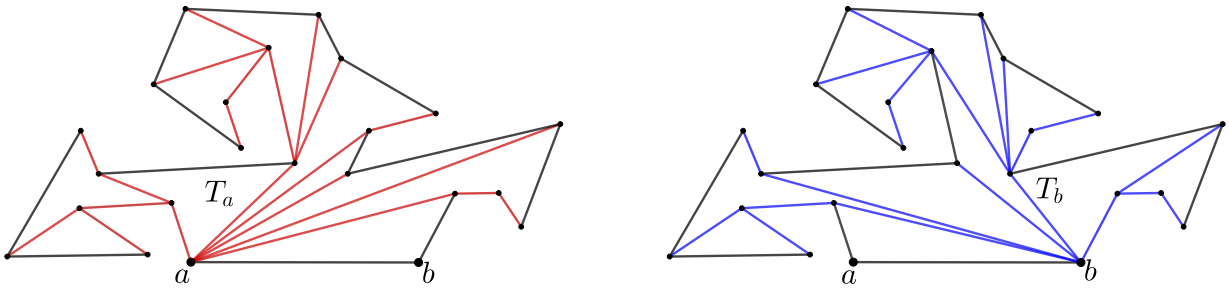
Lemma 29. *There is a linear time algorithm to find, given a point p in a polygon, the shortest path tree T_p and its augmentation to a shortest path map.*

Proof. The idea is simple. Construct the shortest path tree from p to all vertices and augment to the shortest path map using the algorithm by Guibas et al. [49]. Regions of the shortest path map are triangles. Check each triangle in $O(1)$ time to see if it contains the last segment of a shortest path from p to an edge. For further details see [71, 72, Section 4.1.2] , which solves the more general case of shortest paths to a set of chords in a polygon when no two chords nest. Note that these algorithms assume p is on the boundary of the polygon, but we can handle an interior point p by first cutting the polygon at a chord through p in linear time and then finding shortest paths on each side of the chord. □

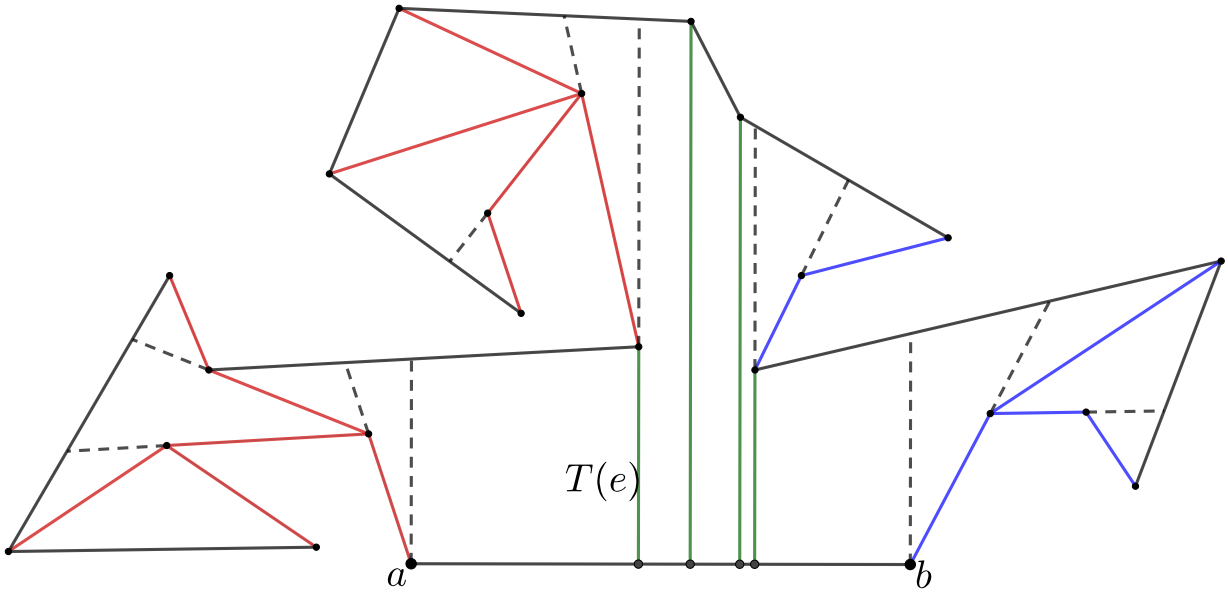
Shortest Paths from an Edge to all Vertices. For an edge $e = ab$ of polygon P, define $\boldsymbol{T(e)}$ to be the forest of shortest paths from e to all vertices of the polygon. See Figure 6.6. We will use the shortest path trees T_a and T_b at the endpoints of e to construct $T(e)$.

A vertex v is ***visible*** from e if there is a line segment xv inside P for some point $x \in e$, and v is ***orthogonally visible*** from e if xv can be orthogonal to e. Pollack et al. [91] note that vertex v is visible from e iff v has different parents $p_a(v)$ and $p_b(v)$ in T_a and T_b. Furthermore, if $p_a(v) \neq p_b(v)$ then v is visible from the interval $[x_a(v), x_b(v)]$ in e where $x_a(v)$ and $x_b(v)$ are the intersections of e with the lines from v to $p_a(v)$ and $p_b(v)$ respectively.

In the following lemma, we augment the forest to obtain the shortest path map with respect to an *edge*. This is different from the shortest path map used in Section 5.3.1.2 (and defined in Chapter 2) that partitions the polygon into regions based on the distances to a *vertex*.



(a) Shortest Path Trees from the endpoints of edge $e = ab$



(b) The resulting shortest path forest for the edge e and its partition into the shortest path map.

Figure 6.6: The shortest path trees from the endpoints of an edge can be used to construct its shortest path forest. The color of an edge indicates the shortest path tree it originates from. Green edges indicate orthogonal visibility from e.

Lemma 30. *There is a linear time algorithm to find, given an edge e of a polygon, the forest $T(e)$ that consists of shortest paths from e to all the vertices of the polygon, and to augment this to the shortest path map.*

Proof. To construct $T(e)$, we define the parent of each vertex of P. If $p_a(v) = p_b(v)$ then v has the same parent in $T(e)$. Otherwise, v is visible from e. If the angles $\angle v x_a(v) b$ and $\angle v x_b(v) a$ are both $\leq \pi/2$ then v is orthogonally visible from e, and we define the parent

of v to be the foot of the perpendicular from v to e. And if one of the angles is obtuse, then the parent of v in $T(e)$ is whichever of $p_a(v)$ or $p_b(v)$ that leads to the obtuse angle. See, for example, the vertex a in Figure 6.6.

We now have the shortest path forest $T(e)$. To augment to the shortest path map, we first begin by constructing the vertex shortest path maps for subtrees rooted at a, b, and all the orthogonally visible vertices. This work takes linear time. Finally, we construct the perpendiculars from the endpoints of e until they intersect ∂P. The extensions from consecutive orthogonally visible vertices split the polygon into trapezoids and triangles giving the required shortest path map.

The algorithm consists of constructing two shortest path trees (rooted at a and b) and iterating over their nodes a constant number of times, thus making the construction of the shortest path forest of an edge a linear time algorithm. □

Refer to Figure 6.6b for the final diagram of the shortest path forest $T(e)$ from the edge e where the color of the subtrees rooted at orthogonally visible vertices is chosen according to the source of the subtree (red for T_a, blue for T_b). The green edges are orthogonally visible.

Note that we can easily modify the algorithm in Lemma 30 to construct the shortest path forest from any line segment in the interior of a given simple polygon in linear time. The shortest path forest corresponding to the segment $s \in P$ is denoted $T(s)$. The forest consists of trees whose roots are points along e. The order of these roots along e is known.

6.2.5 Separators and Funnels

Any geodesic path between two vertices of P separates the boundary of P into two parts, and when we focus on which vertices/edges are in opposite parts, we call the geodesic path a "separator".[2] Separators are a main tool for finding all farthest vertices in a polygon. They were first introduced by Suri [97] (although he called them "connectors" rather than "separators") in his $O(n \log n)$ time algorithm to find farthest vertices of all vertices, and then they were used by Hershberger and Suri [56] who improved the runtime to $O(n)$. Vertex separators (called "separating paths") were also used by Ahn et al. [4], both when they appealed to Hershberger-Suri, and in more direct ways. We need edge separators in similar ways.

The basic properties that Suri [97] proved for separators for farthest vertices are as follows:

[2]These separators are not related to the notion of separators in graph theory.

1. If two vertices x and y are separated by a geodesic path $\pi(a, b)$, then the shortest path from x to y is contained, except for one edge, in the shortest path trees of a and b [97, Lemma 4]. Thus, after constructing the shortest path trees from a and b, it is easy to find shortest paths for any pair x, y that is separated by $\pi(a, b)$.

2. A constant number of separators suffice to separate every vertex from its farthest vertex [97, Section 4].

In this section we develop the analogous theory of separators for farthest edges.

Definition 31. *A **farthest edge separator** is a directed geodesic path $\gamma = \pi(a, b)$ from some vertex a to some vertex b of P such that for every point $p \in \partial P$ to the right of γ, all of p's farthest edges lie to the left of γ.*

Note that we define separators via the strong property that *all* points to one side have their farthest edge on the other side. Although this property is not part of Suri's original definition, his construction produces vertex separators with the property.

In this section we will prove that Suri's two properties hold for our farthest edge separators. We first note an even more basic property that is the main reason for using separators:

Claim 32. *If $\gamma = \pi(a, b)$ is a farthest edge separator and points p and q lie to the right of γ and their farthest edges $F(p)$ and $F(q)$ are distinct, then the paths to their farthest edges cross.*

Proof. The ordering $p, F(p), q, F(q)$ is excluded by the separator. The ordering $p, q, F(q), F(p)$ cannot occur by Property **(P2)**. Thus the ordering must be $p, q, F(p), F(q)$. See also Figure 6.4. □

6.2.5.1 Funnels and Shortest Paths Across a Separator

We first address Suri's property (1) by examining how a shortest path crosses a geodesic $\gamma = \pi(a, b)$. In this subsection the geodesic need not be a farthest edge separator, and the shortest path need not go to a farthest edge. Hershberger and Suri [56] expanded on Suri's result and showed how a shortest vertex-to-vertex path that crosses γ is related to the *funnels* of the vertices. We follow their analysis.

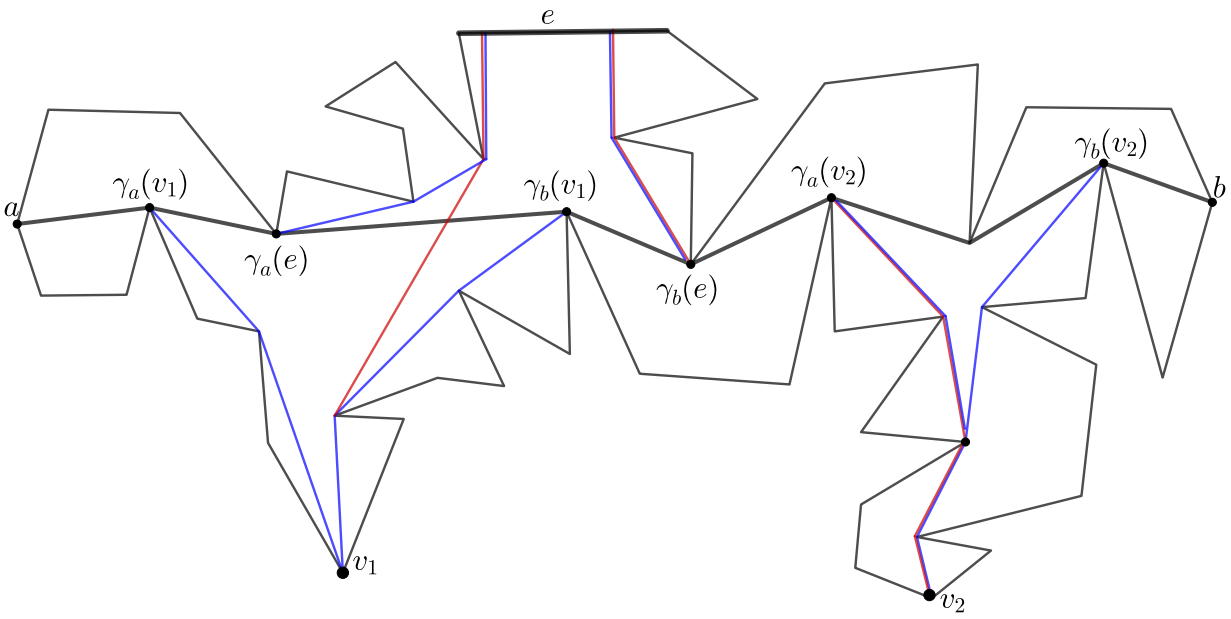


Figure 6.7: A geodesic $\gamma = \pi(a, b)$, the funnels $Y(v_1), Y(v_2)$, and $Y(e)$ (in blue) and the paths $\pi(v_1, e)$ and $\pi(v_2, e)$ (in red).

Suppose that vertex v lies to the right of γ and edge e lies to the left of γ. Then $\pi(v, e)$ crosses γ, either at a single point, or by sharing chords with γ. See Figure 6.7. We show that $\pi(v, e)$ lies in the *funnels* of v and e which are defined in terms of the shortest path trees T_a and T_b. Note that γ is in both T_a and T_b since it is the shortest path from a to b.

The ***funnel of v***, denoted $\boldsymbol{Y(v)}$, is bounded by $\pi(a, v)$, $\pi(b, v)$ and γ, where $\pi(a, v)$ and $\pi(b, v)$ are called the ***walls*** of the funnel. The vertex where $\pi(a, v)$ diverges from γ is $\boldsymbol{\gamma_a(v)}$, defined to be the lowest common ancestor of v and b in the tree T_a. Similarly, the vertex where $\pi(b, v)$ diverges from γ is $\boldsymbol{\gamma_b(v)}$, the lowest common ancestor of v and a in the tree T_b. The vertex where $\pi(v, a)$ diverges from $\pi(v, b)$ is called the ***apex*** of the funnel. Observe that the path between the apex and $\gamma_a(v)$ [or $\gamma_b(v)$] is reflex.

Similarly, the ***funnel of e***, $\boldsymbol{Y(e)}$ is bounded by $\pi(a, e)$, $\pi(b, e)$, γ, together with the piece of e between the terminals $t(a, e)$ and $t(b, e)$ if those terminals are distinct. The lowest common ancestors $\boldsymbol{\gamma_a(e)}$ and $\boldsymbol{\gamma_b(e)}$ and the ***apex*** can be defined analogously, where we allow the apex to be the piece of edge e between $t(a, e)$ and $t(b, e)$ when those terminals are distinct. Funnels have been used in many shortest path algorithms, and there are variations on how they are defined (as a subpolygon or a set of edges; including the edges common to two paths or not, etc.).

The pair of funnels $Y(v), Y(e)$ is ***closed*** if the paths $\pi(\gamma_a(v), \gamma_b(v))$ and $\pi(\gamma_a(e), \gamma_b(e))$ are internally disjoint, see $Y(v_2)$ and $Y(e)$ in Figure 6.7. Otherwise the pair of funnels is ***open***, see $Y(v_1)$ and $Y(e)$ in the figure. Hershberger and Suri dealt with the case where e is replaced by a vertex u. They showed that if the pair $Y(v), Y(u)$ is closed, then the edges of $\pi(v, u)$ are edges of the funnels. In particular, suppose $\gamma_a(u)$ and $\gamma_b(u)$ are closer to a than $\gamma_a(v)$ and $\gamma_b(v)$ (the other ordering is analogous). Then $\pi(v, u)$ consists of the paths $\pi(v, \gamma_a(v))$, $\pi(\gamma_a(v), \gamma_b(u))$, $\pi(\gamma_b(u), u)$. On the other hand, if the pair $Y(v), Y(u)$ is open, then $\pi(v, u)$ consists of part of a wall of $Y(v)$ and part of a wall of $Y(u)$ joined by a ***tangent*** edge $\boldsymbol{\ell(v, u)}$ that crosses γ.

To deal with an edge funnel $Y(e)$, we abuse the notation and say that a segment that meets e at right angles is tangent to $Y(e)$ (this makes sense if we imagine that the edges that meet e at right angles extend off to infinity).

The above results are used to prove the following two lemmas and will also be used in Section 6.3 when we show how to extend Hershberger and Suri's algorithm for finding farthest vertices of all vertices to the case of farthest edges.

Lemma 33. *Consider a geodesic path $\gamma = \pi(a, b)$ with vertex v to the right and edge e to the left. If the pair of funnels $Y(v), Y(e)$ is closed then the edges of $\pi(v, e)$ are contained in the shortest path trees T_a and T_b. If the pair of funnels $Y(v), Y(e)$ is open then the edges of $\pi(v, e)$ are contained in the shortest path trees T_a and T_b, except for one edge $\ell(v, e)$ that crosses γ and is tangent to $Y(v)$ and $Y(e)$.*

Proof. Consider the terminal point $t(v, e)$ of the path $\pi(v, e)$. If $t(v, e)$ is a vertex of P, then the previous results apply. Otherwise, let s be the last segment of the path $\pi(v, e)$. Segment s meets e at a right angle at point $t(v, e)$. Let u be the vertex at the start of s. If u is to the left of γ then s is an edge of $Y(e)$ and the result follows from the previous result for v and u. Otherwise, u is to the right of γ, the pair of funnels is open, and s is the tangent edge that crosses γ. □

Lemma 34. *Let $\gamma = \pi(a, b)$ be a geodesic path. After linear time preprocessing (to compute the trees T_a and T_b and preprocess them for answering lowest common ancestor queries in constant time), the shortest path $\pi(v, e)$ from any vertex v to the right of γ to any edge e to the left of γ can be computed in time proportional to the number of vertices in $\pi(v, e)$.*

Proof. Suppose that the trees have been preprocessed for efficient searching of lowest common ancestors using the algorithm of Harel and Tarjan [53]. Compute the least common ancestors $\gamma_a(v)$, $\gamma_b(v)$, $\gamma_a(e)$, and $\gamma_b(e)$ in constant time. Test if the pair of funnels $Y(v), Y(e)$ is closed or open in constant time using least common ancestor queries. If the

pair is closed, the path $\pi(v, e)$ consists of subpaths that can be found in time linear in the number of vertices.

Otherwise, we must find the tangent edge $\ell(v, e)$ of the two funnels. Assume first that the edge $\ell(v, e)$ does not meet e at right angles. Note that it suffices to search between the apexes of the funnels—to ease notation, we will just suppose that that those apexes are v and e themselves. Then $\ell(v, e)$ is tangent to two reflex curves where one is a wall of the funnel $Y(v)$ and one is a wall of the funnel $Y(e)$. There are four possible choices for the two reflex curves, and for each choice, we are essentially finding the common tangent of two disjoint convex polygons, a very well-solved problem (see [1] for some history). A simple search that walks from the two apexes along the chosen paths towards γ will find the tangent in time proportional to the number of vertices traversed—and those vertices are part of the output path. Doing this in parallel over the four choices, we can find $\ell(v, e)$ and $\pi(v, e)$ in time linear in the number of vertices of $\pi(v, e)$. (This is the same argument as given by Ahn et al. [4, Lemma 3.5].) Finally, to address the possibility that $\ell(v, e)$ meets e at right angles, we can perform a similar search between e and each of the walls of v's funnel. □

6.2.5.2 Constant Number of Separators

We now turn to Suri's property (2)—finding a constant number of separators.

Lemma 35. *There is a set of at most five farthest edge separators such that every point $p \in \partial P$ (and consequently, every edge of P) lies to the right of at least one of the separators. Furthermore, such a set of separators can be found in linear time.*

This lemma is extremely important because it reduces farthest edge problems to a constant number of "bipartite" cases where the source vertices are separated from the target edges. Lemma 35 will be used in Section 6.3 to find farthest edges from all vertices. It will also be used in Section 6.4 to find the Voronoi diagram on ∂P and in Section 6.6 to construct the coarse cover of P.

Proof of Lemma 35. We first note the consequence that for every polygon edge (x, y), one of the five separators has both x and y to its right. This is because separator endpoints are vertices so a farthest edge separator for the midpoint of edge (x, y) must have x and y to its right. Thus it suffices to prove that there are five farthest edge separators such that every point $p \in \partial P$ lies to the right of at least one separator.

The plan in Suri's proof for the case of farthest vertices, was to follow a chain v_1, v_2, v_3, v_4 where v_{i+1} is the farthest vertex from v_i, and argue that $\pi(v_3, v_4)$ crosses $\pi(v_1, v_2)$, and that this provides three separators, namely the three paths. Our plan is similar, but a bit trickier because our paths go from a vertex/point to a farthest edge, so we must then choose a point in the edge to continue the chain.

Take an arbitrary vertex u and find its farthest edge $F(u)$. Note that $F(u)$ is unique by Lemma 24. This can be achieved in linear time by constructing the shortest path tree T_u (Lemma 29) and finding a leaf furthest from u in this tree. Suppose $F(u)$ is the edge e with endpoints e^-, e^+ in clockwise order. Find the farthest edges $F(e^-)$ and $F(e^+)$ in linear time.

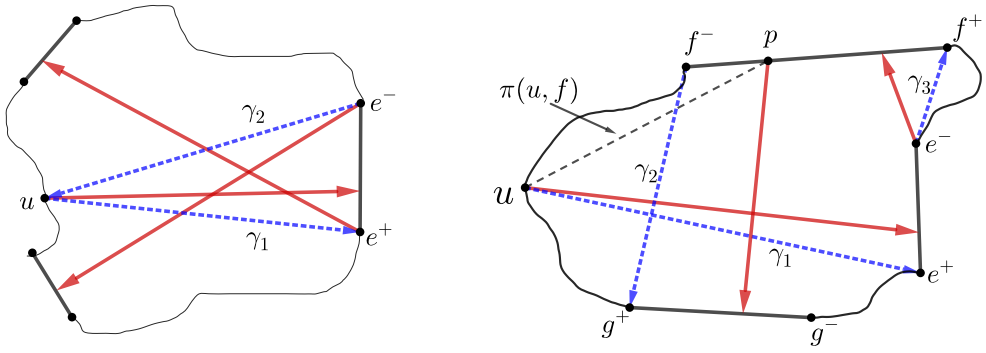
Case 1. First, we suppose that the geodesics $\pi(e^-, F(e^-))$ and $\pi(e^+, F(e^+))$ both cross $\pi(u, e)$. See Figure 6.8. We claim that the geodesics $\gamma_1 = \pi(u, e^+)$, and $\gamma_2 = \pi(e^-, u)$ are farthest edge separators. To prove this, consider a point p to the right of γ_1, i.e., a point in the clockwise chain C from e^+ to u, and suppose that p has a farthest edge $F(p)$ on the same chain. Note that $F(p) \neq e$, since e is not part of the chain. If $F(p)$ occurs before p along the chain C, then $p, u, F(u), F(p)$ occur in that clockwise order and violate Property **(P2)**. Otherwise $F(p)$ occurs after p along the chain C in which case $e^+, p, F(p), F(e^+)$ occur in that clockwise order and violate Property **(P2)**. A symmetric argument shows that γ_2 is a farthest edge separator.

The two geodesics γ_1 and γ_2 separate all points of ∂P from their farthest edges except the points of edge e. We separate those points by adding one more geodesic $\pi(e^+, e^-)$. Note that this kind of degenerate separator is allowed by the definition, and is a farthest edge separator since every point in the edge e has its farthest edge outside e.

This gives a set of three farthest edge separators. Note that they can be found in linear time.

Case 2. Otherwise at least one of the geodesics $\pi(e^-, F(e^-))$ and $\pi(e^+, F(e^+))$ does not cross $\pi(u, e)$. We will consider the case when the geodesic $\pi(e^-, F(e^-))$ does not cross $\pi(u, e)$—the other case is symmetric. Suppose $F(e^-)$ is the edge $f = (f^-, f^+)$ in clockwise order. Find the shortest path from u to edge f, and let point $p := t(u, f)$ be the terminal of that path. Find the farthest edge $g := F(p)$ and suppose $g = (g^-, g^+)$ in clockwise order. We claim that g cannot lie in the clockwise chain from f^+ to e^-. Suppose it does. Then $g \neq e$, which implies that $p \neq u$ (since u has the unique farthest edge e). But then $u, p, F(p), F(u)$ violate Property **(P2)**. Therefore, the edge g lies either: (a) in the clockwise chain from e^- to u, in which case we find separators; or (b) in the clockwise chain from u to f^-, which we prove is impossible. We consider the two cases (a) and (b).

Case 2a. The edge $g = F(p)$ lies in the clockwise chain from e^- to u. The situation



Case 1 Case 2*a*

Figure 6.8: Case 1 and Case 2a from the proof of Lemma 35, showing a schematic of which paths cross.

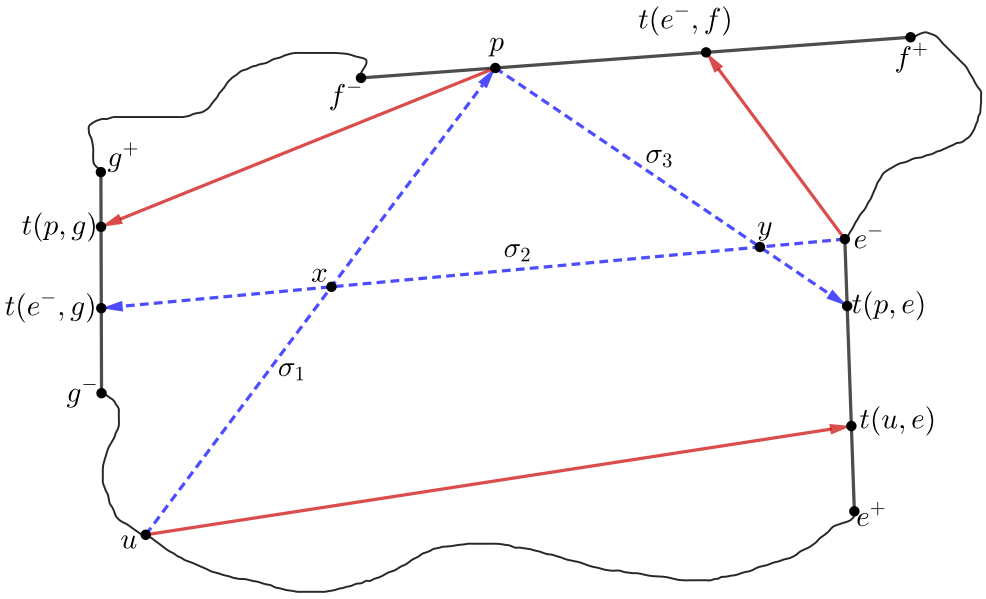


Figure 6.9: Case 2b from Lemma 35. An abstract representation of ∂P is provided, with edges shown explicitly where relevant. Rays drawn in red are the geodesics joining a vertex to its farthest edge. Dashed rays in blue represent geodesics joining vertices to other edges. We prove in the lemma that this case cannot occur, since it contradicts the triangle inequality.

is depicted in Figure 6.8. We claim that the geodesics $\gamma_1 = \pi(u, e^+)$, $\gamma_2 = \pi(f^-, g^+)$ and $\gamma_3 = (e^-, f^+)$ are farthest edge separators. Note that γ_2 is redundant if $f^- = u$, and γ_1 is redundant if $g = e$. To prove that γ_1 is a farthest edge separator, note that because of the "anti-parallel" pair $\pi(u, e)$ and $\pi(e^-, f)$, no point $p \in \partial P$ to the right of γ_1 has a farthest edge to the right of γ_1 (otherwise the path from p to such a farthest edge must go in the same direction as one of $\pi(u, e)$ and $\pi(e^-, f)$, thus violating Property **(P2)**). Similarly, γ_2 is a farthest edge separator because of the anti-parallel pair $\pi(p, g)$ and $\pi(e^-, f)$, and γ_3 is a farthest edge separator because of the same anti-parallel pair.

The three geodesics γ_1, γ_2 and γ_3 separate all points of ∂P from their farthest edges except the points of edges e and f. We can separate those points by adding the geodesics $\pi(e^+, e^-)$ and $\pi(f^+, f^-)$.

This gives a set of five farthest edge separators. Note that they can be found in linear time.

Case 2b. The edge $g = F(p)$ lies in the clockwise chain from u to f^-. See Figure 6.9. We will prove that this case cannot occur. To show this, consider the geodesic paths $\sigma_1 := \pi(u, f) = \pi(u, p)$, $\sigma_2 := \pi(e^-, g)$ and $\sigma_3 := \pi(p, e)$. Note that because e is the unique farthest edge from u, $d(u, e) > |\sigma_1|$. Similarly, $d(e^-, f) > |\sigma_2|$, and $d(p, g) \geq |\sigma_3|$ (p need not have a unique farthest edge). Adding these together, and noting that we have used each part of each $\sigma_i, i = 1, 2, 3$ exactly once, we obtain

$$d(u, e) + d(e^-, f) + d(p, g) > |\sigma_1| + |\sigma_2| + |\sigma_3|. \tag{6.1}$$

Recall that for vertex v and edge h, $t(v, h)$ is the terminal point of the path $\pi(v, h)$. Observe that in clockwise order $p \leq t(e^-, f)$ on edge f, and $t(e^-, g) \leq t(p, g)$ on edge g. Let x be the intersection point of σ_1 and σ_2 (possibly at one of their endpoints). Let y be the intersection point of σ_2 and σ_3 (possibly at one of their endpoints). Observe that along σ_2, y precedes (or is equal to x). See Figure 6.9. We get the following inequality from the definition of a terminal and the triangle inequality:

$$\begin{aligned} d(u, e) &= d(u, t(u, e)) \leq d(u, t(p, e)) \\ &\leq d(u, x) + d(x, y) + d(y, t(p, e)) \end{aligned}$$

Reasoning as above, we also get the following two inequalities:

$$\begin{aligned} d(e^-, f) &\leq d(e^-, y) + d(y, p) \\ d(p, g) &\leq d(p, x) + d(x, t(e^-, g)) \end{aligned}$$

Adding the three inequalities and re-arranging yields:

$$d(u, e) + d(e^-, f) + d(p, g) \leq |\sigma_1| + |\sigma_2| + |\sigma_3|.$$

which contradicts Equation 6.1. □

6.2.6 Chord Oracles and Coarse Covers

In this section we describe the chord oracle results that we need from previous work, and we give a unified explanation of those algorithms and our current algorithm in terms of coarse covers.

We also recall from Chapter 5 the definition of a chord oracle.

Definition 36. *A **chord oracle** decides, given a chord K, whether the geodesic edge center lies to the left or right (or on) the chord.*

Pollack et al. [91] gave a linear time chord oracle for the geodesic vertex center, which is at the heart of all further geodesic center algorithms. In Chapter 5 (also see Lubiw and Naredla [72]), we extended the chord oracle to the case of the geodesic *edge* center (and in fact to the more general case where the sites are half-polygons). In this context, linear-time means linear in the size of the polygon.

In both cases, a main step is finding the ***relative center***, which is a point c_K on K that minimizes the maximum distance from c_K to any site (vertex or edge). The relative center problem is precisely the "one-dimension down" version of the geodesic center problem.

The algorithms to find the relative center on a chord and to find the center of a polygon depend on a crucial convexity property. Define the ***geodesic radius function***, $r(x)$, for $x \in P$ to be the maximum geodesic distance from x to a site (a vertex or edge). Thus the center is the point x that minimizes $r(x)$.

Recall from Chapter 2 that a function is *geodesically convex* on P if the function is convex on every geodesic path in P. A subset $Q \subseteq P$ is a *geodesically convex set* in P if, for any two points a and b in Q, $\pi(a, b)$ in P is contained in Q.

Corollary 14.1 from Chapter 5 is repeated below:

Corollary 14.1. *The geodesic radius function $r(x, \mathcal{H})$ is geodesically convex. The geodesic ball $B(t, H) := \{x \in P : d(x, H) \leq t\}$ for any half-polygon H, and the geodesic ball $B(t) := \{x \in P : r(x, \mathcal{H}) \leq t\}$ are geodesically convex.*

Note that edges are special cases of half-polygons and hence the geodesic radius function for edges is geodesically convex—a fact crucial to the results in the previous chapter and the current one.

More explanation of the role of geodesic convexity will be given below, but basically, it implies that a local optimum is a global optimum.

As mentioned at the beginning of this chapter, the algorithms to find the relative center of a chord, and the algorithms to find the center of a polygon follow the same pattern, which we formalize via the concept of a *coarse cover* of the chord/polygon. Chapter 5 introduced coarse covers for a chord, while here we generalize the notion to a polygon. A precise definition is given below, but the main idea is that a *coarse cover* for a domain (a chord/polygon) is a set of elementary regions R (intervals/triangles) covering the domain, where each region R has an associated easy-to-compute convex function f_R, and such that for any point x in the domain, the maximum distance from x to a site (i.e., $r(x)$) is the maximum of $f_R(x)$ over regions R containing x. Thus the [relative] center problem breaks into two subproblems: (1) find a coarse cover; and (2) find the point x that minimizes the upper envelope of the coarse cover functions. The high-level idea for solving step (2) in linear time (for a chord or polygon domain) is to recursively reduce the domain (the search space) to a subinterval or subpolygon while eliminating elements of the coarse cover whose functions are strictly dominated by others.

We give a precise definition of the coarse cover for the case of farthest edges.

Definition 37. *A **coarse cover** of a chord K [or of polygon P] is a set of triples (R, f, e) where:*

1. *R is a subinterval of K [or a triangle of P], f is a function defined on domain R, and e is an edge of P.*
2. *For all $x \in R$, $f(x) = d(x, e)$ and has one of the following forms:*
   - *$f(x) = d_2(x, v) + \kappa$ where d_2 is Euclidean distance, κ is a constant, v is a vertex of P, and the segment xv is the first segment of the path $\pi(x, e)$.*
   - *$f(x) = d_2(x, \bar{e})$, where d_2 is Euclidean distance, $\bar{e}$ is the line through e, and the path $\pi(x, e)$ is the straight line segment from x to $\bar{e}$ (meeting $\bar{e}$ at right angles).*
3. *For any point $x \in K$ [or P], and any edge e that is farthest from x, there is a triple (R, f, e) in the coarse cover with $x \in R$.*

   In particular, this implies that the upper envelope of the functions of the coarse cover is the geodesic radius function.

The triangles of the coarse cover for polygons will have some additional properties that will be discussed after the algorithm for constructing them (Section 6.6).

We note that the definition in Chapter 5 allows an exception to condition (3): in case two triples have identical R and f (which implies that for all $x \in R$ the paths to two half-polygons have the same length and the the same first segment xv), then we may eliminate one of them. We do not need that exception here because we assumed that no vertex v is equidistant from two or more edges (Assumption 2).

With the above definition of a coarse cover, we can give the following high-level version of the $O(n)$ time chord oracle algorithms of Pollack et al. [91, Section 3] for the vertex center and the extension to the geodesic center of half-polygons described in Section 5.3.1 of Chapter 5. The algorithm for the chord oracle given below is equivalent to the one described in the previous chapter. Later (in Section 6.8), we will examine step (1) in more detail to eliminate the dependence on n when the chord oracle is called during the divide-and-conquer step (this is the primary reason for revisiting this algorithm).

Chord Oracle
Input: a chord K of polygon P on n vertices.
Output: whether the center of P lies left/right/on K.
1. Find a coarse cover of K.
2. Find the point on K that minimizes the upper envelope of the coarse cover functions—this is the relative center c_K.
3. Examine the maximum values of the coarse cover functions at c_K to determine whether the center of P lies left/right/on K.

Step 1 (described in Section 5.3.1.3) takes $O(n)$ time to produce a coarse cover $\mathcal{T}$ of size $O(n)$. Step 2 (described in Section 5.3.1.4) runs in time $O(|\mathcal{T}|)$. Finally, Step 3 (described in Section 5.3.1.1) also takes $O(|\mathcal{T}|)$ time. The correctness of Steps 2 and 3 depends on Corollary 14.1. Complete details of this version of the chord oracle were given in Chapter 5.

We repeat Lemma 19 from Section 5.3.1 below (with the statement modified since the half-polygons here are the n edges):

Lemma 38. *There is a linear-time (i.e. $O(n)$ time) chord oracle for the geodesic center of the edges of an n vertex simple polygon.*

We use the result of step 1—a coarse cover of a chord—in Phase I of our algorithm. However, we do not use the entire chord oracle until the part of Phase II where we have a

coarse cover of the polygon and are searching for a point in the polygon that minimizes the upper envelope of the coarse cover functions. We then make calls to the chord oracle as we use divide-and-conquer to restrict the search space to a subpolygon and eliminate elements of the coarse cover. As noted by Ahn et al., a coarse cover $\mathcal{T}$ of the polygon provides a coarse cover $\mathcal{T}_K$ of any chord K, simply by intersecting the triangles of $\mathcal{T}$ with K. This idea is fleshed out in Section 6.8, where we describe the type of subpolygons we recurse on, and what it means to have a coarse cover of such a subpolygon. We also generalize the chord oracle to a *geodesic oracle* that tests which side of a geodesic path contains the center of the polygon.

6.3 Phase I, Part 1: Finding the Farthest Edge from each Vertex

Phase I is to find the farthest edge Voronoi diagram restricted to the polygon boundary. In this section we give the first step of Phase I:

Theorem 39. *There is a linear time algorithm to find the farthest edge from each vertex of a simple polygon.*

Hershberger and Suri [56] gave a linear time algorithm to find the farthest *vertex* from each vertex in linear time. We show that their algorithm extends to finding the farthest *edge* from each vertex in linear time. Hershberger and Suri build upon an algorithm called SMAWK due to Aggarwal et al. [3] that finds row maxima in a totally monotone matrix in linear time. The SMAWK algorithm immediately solves the problem of finding the farthest vertex from each vertex in a convex polygon in linear time, but Hershberger and Suri need substantial new ideas to extend to general simple polgons. In order to extend Hershberger and Suri's algorithm to find the farthest edge from each vertex, we must examine their algorithm in more detail.

We structure this section as follows:

1. Use separators to reduce the farthest vertex/edge problem to a problem of finding all row maxima in a totally monotone matrix. The matrix is given implicitly—each entry in the matrix represents the distance from one vertex to a vertex/edge, and this distance is computed only when needed.

2. An overview of the SMAWK algorithm to find row maxima in a totally monotone matrix. Together with item 1, this solves the problem of finding the farthest vertex from each vertex in a *convex* polygon, because then each matrix entry (the distance between two vertices) can be computed in constant time.
3. An overview of the Hershberger-Suri algorithm that solves the problem of finding the farthest vertex from each vertex in a general simple polygon. To do this, they show how to compute each matrix entry needed in the SMAWK algorithm in constant amortized time.
4. The modifications required for finding the farthest *edge* from each vertex.

Reducing farthest vertices/edges to row maxima in a matrix. We use the notion of separators from Section 6.2.5. Suri [97] proved that there are a constant number of separators that separate every vertex from its farthest vertex. In Lemma 35 we extended this result to the farthest *edge* from each vertex. Thus, in either case, to find the farthest vertex/edge from each vertex it suffices to solve the following problem: given a separator $\pi(a, b)$, find, for each vertex to the right of the separator, the farthest vertex/edge that lies to the left of the separator.

Consider a ***distance matrix*** M with rows indexed by the vertices to the right of the separator in counterclockwise order and columns indexed by either the vertices or the edges to the left of the separator in counterclockwise order, and with $M(v, s)$ defined to be the geodesic distance from vertex v to vertex/edge s. Then we seek the maximum in each row of the matrix.

A matrix M is ***totally monotone*** if for any 2×2 submatrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, if $b > a$ then $d > c$.

Hershberger and Suri prove that the distance matrix for farthest vertices is totally monotone. We prove the analogous result for farthest edges.

Claim 40. *The distance matrix M for farthest edges as described above is totally monotone.*

Proof. Consider a 2×2 submatrix with rows indexed by vertices u, v and columns indexed by edges e, f:

$$\begin{array}{c} \\ u \\ v \end{array} \begin{array}{c} \begin{array}{cc} e & f \end{array} \\ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \end{array}$$

Because the row order and column order are counterclockwise, and because u, v are to the right of the separator and e, f are to the left of the separator, u, v, e, f occur in counterclockwise order around the polygon. By Corollary 27, if $d(u, f) > d(u, e)$ then $d(v, f) > d(v, e)$, i.e., if $b > a$ then $d > c$. □

Thus the problem of finding the farthest vertex/edge from each vertex is reduced in linear time to the problem of finding row maxima in a totally monotone distance matrix (where we must take into account the time required to access matrix entries).

The SMAWK algorithm to find row maxima in a totally monotone matrix. Let M be an $n \times m$ totally monotone matrix. Break ties for the maximum value in a row by choosing the leftmost maximum. The positions of these row maxima progress rightward and downward—more precisely, if the maximum in row i occurs in column k, then the maximum in row $j > i$ occurs in column $l \geq k$. The SMAWK algorithm [3] finds the (leftmost) maximum in each row as follows:

1. Delete columns (without eliminating any row maxima) to reduce to an $n \times m'$ matrix M', where $m' \leq n$. This is accomplished by a routine called REDUCE that accesses $2m - n$ matrix entries.
2. Let M'' consist of the even numbered rows of M'. Recursively find the row maxima in M''. This gives us the row maxima for all even-numbered rows of M'.
3. Fill in the row maxima for the odd numbered rows of M'. Observe that the column of the maximum in row number $2i + 1$ occurs between the columns of the maxima in row numbers $2i$ and $2i + 2$, which means that this step accesses $n + m'$ matrix entries, where the next access is below or to the right of the current one.

Aggarwal et al. [3] prove that the SMAWK algorithm runs in time $O(n + m)$ assuming that matrix entries can be accessed and compared in constant time. The number of recursive calls ("phases") is $O(\log n)$. An important property is that in step 1 and step 3 each successive matrix entry access is to the right, or up, or down from the current one—in particular there are no left moves. This is stated as Equation (2.3) by Hershberger and Suri [56].

The Hershberger-Suri Algorithm and Its Extension to Farthest Edges. As noted above, the SMAWK algorithm gives a linear time algorithm to find the farthest vertex from each vertex in a convex polygon, because in that case each entry in the distance matrix

can be computed in constant time. However, for a general simple polygon, each matrix access involves finding the distance between two vertices v and u on opposite sides of the separator. Hershberger and Suri show that this can be done in $O(1)$ amortized time per matrix access. Their algorithm relies on the order of matrix accesses in the SMAWK algorithm as mentioned above, and on the properties of shortest paths that cross the separator $\gamma(a, b)$, as discussed in Section 6.2.5.1. We use the terminology and notation from Section 6.2.5.1. The shortest path $\pi(v, u)$ consists of edges of the funnels $Y(v)$ and $Y(u)$, with one additional tangent edge $\ell(u, v)$ in case the pair of funnels is open. The shortest path trees T_a and T_b can be preprocessed in linear time to allow constant time queries for least common ancestors, and for lengths of paths to a or b. Then the length of $\pi(u, v)$ can be found in constant time if the pair of funnels $Y(v), Y(u)$ is closed. The same applies to our case of the shortest path from vertex v to edge e—for example, in Figure 6.7, the funnels $Y(v_2)$ and $Y(e)$ are closed and $d(v_2, e) = (d(a, v_2) - d(a, \gamma_a(v_2)) + (d(b, e) - d(b, \gamma_a(v_2))$.

When a pair of funnels is open the only hard part is finding their tangent edge. Given the tangent edge, the length of the path can be found in constant time. For example, in Figure 6.7, the funnels $Y(v_1)$ and $Y(e)$ are open with tangent edge $\ell(v_1, e) = (x, y)$ of length $d_2(x, y)$ so $d(v_1, e) = (d(b, v_1) - d(b, x)) + (d(a, e) - d(a, y)) + d_2(x, y)$. Hershberger and Suri give a data structure to find the tangent between a pair of open funnels in constant amortized time by storing and maintaining the walls of the funnels $Y(v)$ and $Y(u)$ during each phase of the algorithm as v moves counterclockwise and u moves in either direction. In fact, it suffices to maintain the parts of the walls from the apex of the funnel to γ.

Binary search along the walls can be used to find the tangent edge but this is too inefficient for a linear-time algorithm. Therefore, a more complex data structure that modifies the shortest path trees (T_a and T_b) at each phase is used. Paths in the trees are broken into subpaths, and each subpath is represented by a *supernode* that supports fast searching. Supernodes are stored as binary trees with the original polygon vertices at their leaves, and internal nodes representing the edge joining the subtrees below. Any path of the shortest path tree in the k-th phase is a list of supernodes connected by superedges, such that every supernode has at most 2^k vertices of the original polygon. Finally, Hershberger and Suri provide a method for obtaining the supernode representation for the trees before the k-th phase in time $O(kn/2^k)$. This takes $O(n)$ time for all the $O(\log n)$ phases and also ensures that tangents between open funnels in the k-th phase can be determined efficiently. The maintenance of the supernode representation between phases is quite involved and we do not describe more details here. The data structure permits them to find the tangent edge $\ell(v, u)$ and to update the funnels, in $O(1)$ amortized time per operation. Their algorithm and its analysis depend on a lemma about the difference between two funnels. For two sites (vertices/edges) s_i and s_j on the same side of the separator $\gamma(a, b)$, the

funnel-difference is the set of edges in $Y(s_i)$ that do not occur in $Y(s_j)$. We observe that their result about funnel differences [56, Lemma 3.3] extends to our situation and is crucial for the amortized analysis.

Lemma 41. *The funnel difference of s_i, s_j forms a path that includes the apex of $Y(s_i)$, and is edge-disjoint from $Y(s_k)$, for any vertex/edge s_k that appears in the order s_i, s_j, s_k on the same side of the separator.*

In summary, the Hershberger-Suri algorithm extends in a straightforward manner to farthest edges. The only modifications needed are the extension to edge funnels (instead of funnels based on vertices) and the different number of separators.

6.4 Phase I, Part 2: Finding the Farthest Edge Voronoi Diagram Restricted to the Polygon Boundary

In the previous section we showed how to find the farthest edge from each vertex of the polygon. In this section we build on that result and show how to find $\mathcal{V}(\partial P)$, the farthest edge Voronoi diagram restricted to the polygon boundary.

Our construction of $\mathcal{V}(\partial P)$ allows us to determine the geodesic edge center if it lies on ∂P. If it does, we are done. If not, we can safely assume in Phase II (starting from the next section) that the edge center lies in the interior of P.

Theorem 42. *There is an $O(n)$ time algorithm to obtain the restriction $\mathcal{V}(\partial P)$ of the farthest edge Voronoi diagram to the boundary ∂P.*

Based on the general position assumptions in Section 6.2.2, the boundary of P consists of chains with a single farthest edge, separated by isolated points (not vertices) that have two farthest edges. Our goal is to find these points.

If two consecutive polygon vertices a and b have the same farthest edge, i.e., $F(a) = F(b)$, then all points on the edge ab have the same farthest edge, by the Ordering Property **(P2)** of Lemma 25. However, if $F(a) \neq F(b)$ then we must do more work to find the farthest edge Voronoi diagram on the edge ab. A polygon edge ab is a ***transition edge*** if $F(a) \neq F(b)$. See Figure 6.10.

We will find the farthest edge Voronoi diagram on one transition edge in linear time. To handle *all* the transition edges in linear time, we will show that for each transition edge

ab we can restrict our attention to a subpolygon $H(a, b)$ called the ***hourglass*** of ab. See Figure 6.10 and the definition in Section 6.4.1 below.

The proof of Theorem 42 then has two parts: in Section 6.4.1 we show that the hourglasses of all transition edges can be found in linear time and that the sum of their sizes is linear; and in Section 6.4.2 we show how to find the farthest edge Voronoi diagram on a transition edge inside its hourglass in time linear in the size of the hourglass. For the second step, we use a coarse cover (see Section 6.2.6) of the transition edge.

We briefly discuss related work. Hourglasses were first used in algorithms for shortest paths [49, 50, 26], and then used in algorithms to find the farthest vertex geodesic Voronoi diagram (Aronov et al. [10]) and in algorithms to find the geodesic [vertex] center (Ahn et al. [4]).

As explained at the beginning of this chapter, there is a linear time algorithm to find the farthest *vertex* Voronoi diagram restricted to the boundary of a polygon due to Oh, Barba and Ahn [87, Section 3]. Our approach is roughly similar to theirs (with farthest edges in place of farthest vertices), but there are some significant differences. First of all, we will use the Voronoi diagram on the polygon boundary to easily find a coarse cover of the polygon, whereas previous papers did things in reverse—Ahn et al. had a more complicated way of finding a coarse cover of the polygon which Oh et al. then used to find the boundary Voronoi diagram. Another difference is that our algorithm to find the Voronoi diagram on one transition edge is considerably simpler—see Section 6.4.2 for details.

6.4.1 Hourglasses

In this section we show that to find the Voronoi diagram on a transition edge ab it suffices to look at the *hourglass* of ab, and we show that all the hourglasses can be found in linear time, and the sum of their sizes is linear.

Let ab be a transition edge directed counterclockwise. Note that $\pi(a, F(a))$ and $\pi(b, F(b))$ cross each other by the Ordering Property **(P2)** of Lemma 25. The ***hourglass*** $H(a, b) = H(e)$ is the subpolygon of P bounded by ab, $\pi(a, F(b))$, $\pi(b, F(a))$ and the clockwise portion of ∂P between the terminals $t(a, F(b))$ and $t(b, F(a))$. See Figure 6.10. Recall that every vertex of P has a unique farthest edge by Assumption 2; and $t(a, F(b))$ is the terminal point of the geodesic path from vertex a to the edge $F(b)$. The geodesics $\pi(a, F(b))$ and $\pi(b, F(a))$ are called the ***walls*** of the hourglass $H(a, b)$, and the part of ∂P clockwise from $t(b, F(a))$ to $t(a, F(b))$ is called the ***chain*** of the hourglass. The ***size*** of an hourglass is its number of vertices.

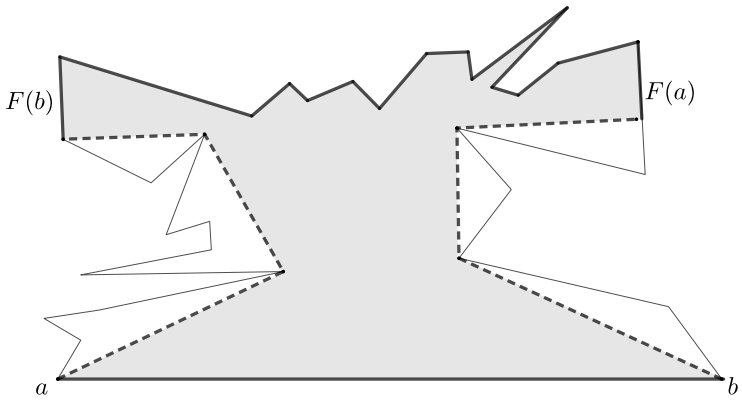


Figure 6.10: The hourglass for a transition edge ab. The walls $\pi(a, F(b))$ and $\pi(b, F(a))$ are represented by dashed polylines.

The following lemma justifies restricting attention to the hourglass of a transition edge ab in order to find the farthest edge Voronoi diagram restricted to ab. It is a consequence of the Ordering Property from Lemma 25.

Lemma 43. *Let p be a point on the transition edge ab and let e be a farthest edge from p in P. Then e lies in the chain of the hourglass $H(a, b)$.*

Proof. Suppose e lies in the clockwise chain of ∂P from $F(a)$ to b. Then the clockwise ordering around ∂P is $p, a, F(a), e$ in contradiction to the Ordering Property (Property **(P2)** of Lemma 25). Similarly, if e lies in the clockwise chain ∂P from a to $F(b)$, the clockwise ordering $b, p, e, F(b)$ contradicts the Ordering Property. □

Let $\mathcal{H}$ be the set of hourglasses of all the transition edges of P. In the remainder of this subsection we show how to find $\mathcal{H}$ in linear time.

Lemma 44. *All the hourglasses of $\mathcal{H}$ can be constructed in $O(n)$ time. In particular, the sum of their sizes is $O(n)$.*

Proof. Recall that by Lemma 35 there are five farthest edge separators such that for every edge ab, one of the separators has a and b to its right and $F(a)$ and $F(b)$ to its left. Let $\gamma = \pi(p, q)$ be a farthest edge separator and let $\mathcal{H}_\gamma$ be the set of hourglasses of transition edges that lie to the right of γ. It suffices to prove the lemma for one set $\mathcal{H}_\gamma$.

Each hourglass in $\mathcal{H}_\gamma$ consists of a transition edge ab to the right of γ, two walls, and a chain to the left of γ. In Section 6.3, we found the farthest edge from each vertex in linear

time, so we know $F(a)$ and $F(b)$. Because the hourglass chains are internally disjoint, we just need to show that we can find all the walls of the hourglasses in $\mathcal{H}_\gamma$ in linear time.

Each wall is a shortest path from a vertex to the right of the separator $\gamma = \pi(p, q)$ to an edge to the left of γ, so by Lemma 33 each wall consists of edges of the shortest path trees T_p and T_q, except for at most one edge crossing γ. The set of crossing edges has size $O(n)$ because there are $O(n)$ hourglasses. By Lemma 29 the shortest path trees T_p and T_q can be found in time $O(n)$ and have size $O(n)$. By Lemma 34 we can find each wall in time proportional to the size of the wall. Thus we can find all the walls in time $O(n)$ so long as we show that each polygon chord is in $O(1)$ walls. (Note that walls of hourglasses are not paths to farthest edges, so we cannot simply apply Property **(P4)** that crossing paths to farthest edge do not share directed chords.)

Claim 45. *Any chord of the polygon is in $O(1)$ walls of hourglasses of $\mathcal{H}_\gamma$.*

Proof. Let $t_1, \ldots, t_k$ be the transition edges to the right of γ in clockwise order. If two transition edges are close together in this ordering, then their walls may have common chords, but we will show that if t_i and t_j are separated by at least three transition edges, i.e., $j - i \geq 4$, then the walls of the hourglasses $H(t_i)$ and $H(t_j)$ have no common chords. Note that this proves the claim.

So, consider t_i and t_j with $j - i \geq 4$, and suppose for a contradiction that a wall of $H(t_i)$ and a wall of $H(t_j)$ share a common chord f. Take the intermediate transition edge $t_k = (u, v)$, where $k = i + 2$. Then the farthest edges of the endpoints of t_i, t_j and t_k are all distinct (this is the reason for choosing i, j, k as we did), and all lie to the right of γ. We will show that the paths $\pi(u, F(u))$ and $\pi(v, F(v))$ also share the chord f. This means that we have crossing paths to distinct farthest edges and the paths share a chord, which contradicts Property **(P4)** from Lemma 25.

It remains to show that $\pi(u, F(u))$ uses the chord f. (The case of $\pi(v, F(v))$ is exactly the same.) The idea is that this path is "squashed" between the two walls that use f. Suppose that the wall $\pi(p_1, e_1)$ of $H(t_i)$ and the wall $\pi(p_2, e_2)$ of $H(t_j)$ both use chord f. See Figure 6.11. Here p_1 and p_2 are distinct vertices on the right of the separator γ and e_1 and e_2 are distinct edges on the left of γ. Vertex u lies between p_1 and p_2 in clockwise order, and $F(u)$ lies between e_1 and e_2 in clockwise order, and all are distinct. Let $f = xy$ where x and y are vertices of the polygon. Because shortest paths to the same destination do not cross, the shortest path from u to y goes through x. Similarly, the shortest path from x to the edge $F(u)$ goes through y. The union of these two shortest paths is a geodesic (locally shortest) path from u to $F(u)$ and must therefore be the shortest path from u to $F(u)$. Thus $\pi(u, F(u))$ uses the edge f. □

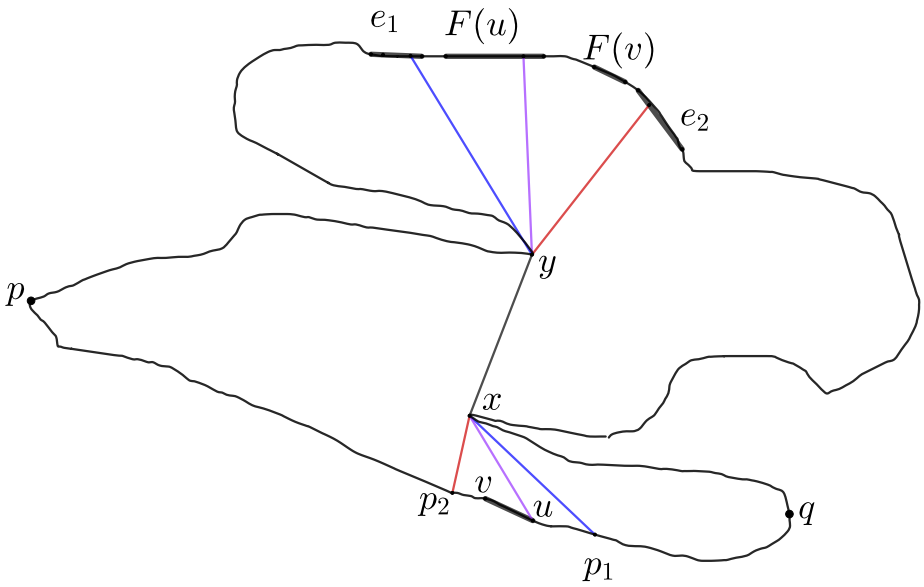


Figure 6.11: The walls $\pi(p_1, e_1)$ and $\pi(p_2, e_2)$ share the chord $f = xy$, which forces the path $\pi(u, F(u))$ to also use f.

This completes the proof of Lemma 44. □

6.4.2 Finding the Farthest Edge Voronoi Diagram on One Edge

In this section, we show how to construct the farthest edge Voronoi diagram along an edge inside a polygon in time linear in the size of the polygon. We can apply this to find the farthest edge Voronoi diagram along a transition edge ab inside the hourglass polygon $H(a, b)$ in time linear in the size of $H(a, b)$, though our algorithm is more general and does not use any properties of hourglasses. We use the ***coarse cover*** (Definition 37) of the edge ab, which can be found in linear time (see Lubiw and Naredla [72], or Section 5.3.1.3 of this thesis). Recall that the coarse cover is a set of triples (I, f, e) where I is a subinterval of ab and $f(x) = d(x, e)$ for any $x \in I$. The upper envelope of the convex functions f is the distance function to the farthest edge.

It is easy to compute the upper envelope of the functions f, and thus the Voronoi diagram on ab, if we first sort the endpoints of the intervals I—but we cannot afford to sort. Instead, we will add the coarse cover elements (I, f, e) one by one, maintaining a list M of [pairwise internally] disjoint subintervals of ab together with an associated distance function $f_M(x)$. When we consider (I, f, e) we will insert into M the initial subinterval of

I where $f(x) > f_M(x)$. We must be able to quickly find the left endpoint of I in M and to ensure that one subinterval of I is all we need to add. This requires a particular ordering of the coarse cover elements, for which we must delve more deeply into how the coarse cover was defined. Specifically, elements of the coarse cover of edge ab are associated with edges of the shortest path trees T_a and T_b (that consist of the shortest paths from a and b, respectively, to all the edges of P), and we will use the ordering of the trees as embedded in the plane.

As mentioned above, Oh, Barba, Ahn [87] gave a linear time algoirthm to find the farthest *vertex* Voronoi diagram on the boundary of P. Their algorithm also works by adding coarse cover elements to obtain the upper envelope of the distance functions. Their coarse cover is obtained differently, and their algorithm is quite complicated—they iterate over the sites (the vertices, in their case), adding all coarse cover elements associated with each site. In order to find the correct insertion point on ab, they sweep back and forth maintaining a shortest path from the current point on ab to the current vertex. Their argument that updating the path costs (amortized) linear time depends in the final analysis on properties of the coarse cover that are not made explicit (see Lemma 7 in their paper). Our algorithm is much simpler, both in its description and its analysis, because we use the relationship between our coarse cover and shortest path trees, which allows us to iterate over edges of the shortest path trees rather than iterating over the sites.

6.4.2.1 Coarse Covers and Shortest Path Trees

We begin by describing how the coarse cover (see Definition 37) of an edge ab is constructed, and how it is related to the shortest path trees T_a and T_b. These results were described in detail in Chapter 5. We recall the concept here with slightly modified notation since we deal with the special case when all the half-polygons are edges. Direct the edges of T_a from the root a to the leaves (which correspond to the edges of the polygon). Similarly, direct the edges of T_b from the root b to the leaves. For any node v in T_a define $\boldsymbol{\ell_a(v)}$ to be the maximum length of a directed path in T_a from v to a leaf node representing a terminal point on some polygon edge, and define $\boldsymbol{F_a(v)}$ to be that polygon edge. Define functions $\boldsymbol{\ell_b}$ and $\boldsymbol{F_b}$ similarly. As discussed in Chapter 5, the ℓ and F values can be computed in linear time.

Define $\boldsymbol{p_a(u)}$ and $\boldsymbol{p_b(u)}$ to be the parents of node u in T_a and T_b, respectively. As noted by Pollack et al. [91], a vertex u is visible from some point on ab if and only if $p_a(u) \neq p_b(u)$. Furthermore, we note that if u is visible from some point on ab, then extending the edge from u through $p_a(u)$ reaches a point $\boldsymbol{x_a(u)}$ on ab from which u is visible. Similarly, extending the edge from u through $p_b(u)$ reaches a point $\boldsymbol{x_b(u)}$ on ab

from which u is visible. The intervals of the coarse cover will be defined using the points $x_a(u)$ and $x_b(u)$.

Note that, as a special case, we add a 0-length edge to T_a at the root a to account for the path (of length 0) from a to the polygon edge incident to a. The 0-length edges ensure that every polygon edge is associated with a unique leaf of T_a [and of T_b]. Recall that some other 0-length segments were already appended to the shortest path trees in order to address corner cases in Section 5.3.1.3. We can now define the coarse cover.

The coarse cover $\mathcal{T}$. Next, we describe the different types of elements in a coarse cover. Note that here we have fewer types of coarse cover elements for edges since, unlike half-polygons, an edge cannot properly intersect a chord. The coarse cover $\boldsymbol{\mathcal{T}}$ consists of elements of the following three types (Note that elements of type 0 used in Section 5.3.1.3 for half-polygons can never occur for edges):

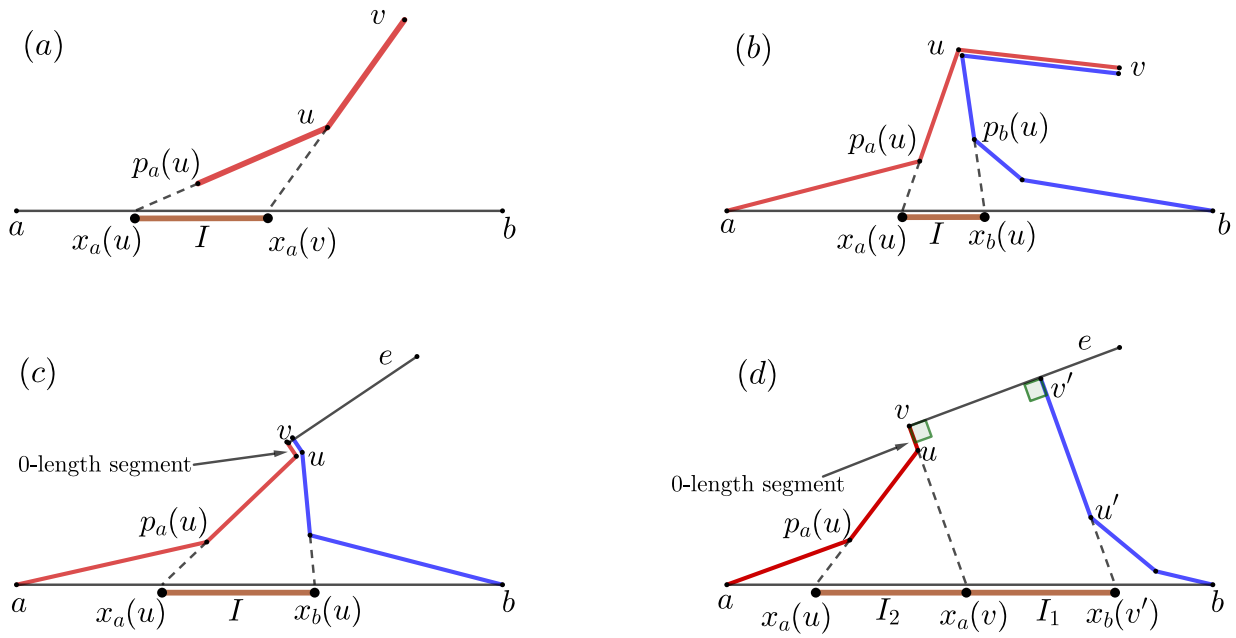


Figure 6.12: Elements of the coarse cover (this figure is the same as Figure 5.8 with half-polygons replaced by edges): (a) I is the base of an a-side triangle associated with edge uv; (b) I is the base of a central triangle associated with edge uv; (c) I is the base of a central triangle associated with 0-length edge uv; (d) I_1 is the base of a central trapezoid associated with terminal points v and v' on e; I_2 is the base of an a-side triangle associated with 0-length edge uv.

1. For each directed edge (u, v) in T_a where $u \neq a$, and u and v are both visible from ab, there is an **a-side triangle** with an associated coarse cover element. The **b-side triangles** and their associated coarse cover elements are defined symmetrically.
2. For each edge (u, v) that is common to T_a and T_b where u is visible from ab and v is not (i.e., $u = p_a(v) = p_b(v)$) there is a **central triangle** with a coarse cover element associated with it.
3. For each polygon edge e such that the terminal points $\boldsymbol{t(a, e)}$ of $\pi(a, e)$ and $\boldsymbol{t(b, e)}$ of $\pi(b, e)$ are distinct, there is a **central trapezoid** and an associated coarse cover element.

In Chapter 5 (specifically, Lemma 22), we proved that $\mathcal{T}$, as defined above, is a coarse cover and can be found in linear time (in fact, the proof works for the more general case of farthest half-polygons).

Coarse cover elements of Type 1 and Type 2 (a-side triangles, b-side triangles, and central triangles) are associated with edges (u, v) of T_a and T_b with at least one node visible from ab. Elements of Type 3 (central trapezoids) are associated with polygon edges, and thus with leaves of T_a.

Observation 46. *If uv and vw are edges of T_a that have associated coarse cover elements (I_1, f, e) and (I_2, f', e'), then the right endpoint of I_1 is $x_a(v)$ and the left endpoint of I_2 is $x_a(v)$, i.e., I_1 and I_2 appear in that order along ab and intersect in a single point. This observation is also true for an edge uv and a central trapezoid at v if v happens to be a leaf.*

A similar property holds for T_b.

Lemma 47. *Suppose edge uv of T_a has an associated coarse cover element $(I, f, F_a(v))$. Then:*

1. *On the path $\pi(a, u)$ all edges except the first one have associated coarse cover elements.*
2. *On the path $\pi(v, F_a(v))$ let x be the last vertex visible from ab. All edges on $\pi(v, x)$ have associated coarse cover elements for the polygon edge $F_a(v)$. Furthermore, if x is a leaf then there is a central trapezoid associated with $F_a(v)$, and otherwise there is an edge xy in $\pi(v, F_a(v))$ and it is associated with a central triangle for $F_a(v)$.*

A similar property holds for T_b.

From Lemma 47 and Observation 46, we get:

Corollary 48. *For any edge e of P, let $C(e)$ be the set of coarse cover elements (I, f, e) for e. If $C(e)$ is nonempty, then its elements correspond to a (possibly empty) path in T_a directed towards a leaf, followed by a central triangle or trapezoid, followed by a (possibly empty) path in T_b directed towards the root. Furthermore, the corresponding intervals on ab appear in order, are [internally] disjoint, and their union is an interval.*

Proof of Lemma 47. The first statement just depends on the fact that if u is visible from ab (i.e., has different parents in T_a and T_b) then the same is true for every vertex on the path $\pi(a, u)$.

For the second statement, note that $F_a(v)$ is the farthest edge from v in the subtree of v. Let w be any vertex on $\pi(v, x)$, and e' any polygon edge for which the terminal $t(a, e')$ lies in the subtree of w. We have $d(v, e') = d(v, w) + d(w, e')$. Since $d(v, F_a(v)) > d(v, e')$, the previous equality implies $d(w, F_a(v)) > d(w, e')$. So $F_a(w) = F_a(v)$ and the coarse cover for the tree edge joining w to its parent $p_a(w)$ is also associated with $F_a(v)$.

If x is a leaf, the terminals $t(a, F_a(v))$ and $t(b, F_a(v))$ are distinct (due to the introduction of 0-length segments). From the definition of the coarse cover elements, this means there is a central trapezoid associated with $F_a(v)$. If x is not a leaf, let xy be the first edge on the path $\pi(x, F_a(v))$ (the latter path is a subpath in $\pi(v, F_a(v))$). The vertex (or terminal point) y is not visible from ab and xy is a tree edge in both T_a and T_b. The coarse cover element for xy is a central triangle associated with $F_a(v)$. □

6.4.2.2 A DFS-based Algorithm for the Farthest Edge Voronoi Diagram

In this section we give the final algorithm to find the farthest edge Voronoi diagram on polygon edge ab. We first construct a tree T whose edges correspond to coarse cover elements of ab. Then we incrementally construct the farthest edge Voronoi diagram on ab by adding coarse cover elements in a depth first search (DFS) order of T.

Constructing a tree of coarse cover elements. We construct a tree T whose edges are in one-to-one correspondence with the coarse cover elements. For ease of terminology, we will refer to, e.g., an a-side triangle as a coarse cover element, though to be precise, we should say "the coarse cover element associated with the a-side triangle." In brief, T is constructed as follows: we attach each central trapezoid to the associated leaf vertex of T_a; add the path of b-side triangles for each polygon edge e after the central triangle or trapezoid for e; and contract original edges of T_a that are not associated with coarse cover elements. We give more detail of these steps.

1. For each central trapezoid, say associated with polygon edge e, there is a leaf l of T corresponding to e. Attach a new edge in T descending from l and associate the central trapezoid with it.
2. For each polygon edge e that has b-side triangles associated with it, those triangles correspond to a path π in T_b, that is directed in leaf-to-root order, see Corollary 48. The tree T currently has an edge, say g, associated with the central triangle/trapezoid for e. Attach the path π at end of the edge g.
3. Finally, we contract any original edge of T_a that is not associated with a coarse cover element. These are edges uv such that u is not visible from ab plus the original edges incident to a.

The resulting tree T can be constructed in linear time and its edges are in one-to-one correspondence with the coarse cover elements. Observation 46 extends to the tree T:

Observation 49. *If uv and vw are edges of T, then the intervals I_1 and I_2 of the corresponding coarse cover elements appear in that order along ab and intersect in a single point.*

DFS Algorithm for the Voronoi Diagram. We process the coarse cover elements one by one, ordered according to a depth first search of the tree T in which we explore children of a vertex in clockwise order (i.e., with the most counterclockwise child first). We maintain a list M of interior disjoint subintervals of ab whose union is an interval starting at a. With each subinterval in M we record the coarse cover element it came from. We define f_M to be the distance function determined by the intervals of M. In the end M will be the farthest edge Voronoi diagram on ab.

During the depth first search of T, when we traverse an edge uv with an associated coarse cover element (I, f, e), we ensure that any part of this element that defines the final upper envelope is inserted into M. (Note the subtlety that M need not be the upper envelope of the coarse cover elements examined so far.)

In fact, the algorithm will insert only one subinterval of I beginning at its left endpoint. If all of I is inserted, then the algorithm recursively continues the depth first search on vertex v, and otherwise the whole subtree rooted at v is abandoned. There are two issues: correctness, which is addressed later; and efficiency, which we discuss here. In interval $I = [l, r]$ we must compare the distance function f to the distance function f_M determined by the intervals of M. We begin at l, the left endpoint of I. To compare $f(l)$ with $f_M(l)$, we must locate l in M at unit cost. If u is the root of T, then $l = a$, and we begin at

the start of M. Otherwise, the edge of T from the parent of u to u has an associated coarse cover element with an interval I'. By Observation 49, the right endpoint of I' is l. Thus, we will store with vertex u a pointer p_u to the interval of M that contains the right endpoint of I'. This allows us to access $f_M(l)$ at unit cost.

We define a recursive routine, `Insert`(u, p_u) where u is a vertex of T and p_u is a pointer into the list M as described above. The goal is to insert into M the portions of coarse cover elements that are associated with the subtree of T rooted at u and that determine the final upper envelope. Initially M has a single endpoint a and pointer p_a points to it. At the top level we call `Insert`(a, p_a).

`Insert`(u, p_u) # u is a node of T and p_u is a pointer to an interval of M

for each child v of u in clockwise order **do**
  (I, f, e) := the coarse cover element associated with the edge uv of T
  l := left endpoint of I; r := right endpoint of I
  Invariant: p_u points to an interval of M that contains l
  if $f(l^+) > f_M(l^+)$ where l^+ is just to the right of l **then**
    replace intervals of M starting at p_u with a subinterval of I ending at the "cross-over" point $t < r$ where f_M starts to dominate f, or at r
    if f dominates until r and v is not a leaf of T **then**
      call `Insert`(v, p_v), where p_v is a pointer to interval I in M

Checking $\mathbf{f(l^+) > f_M(l^+)}$. We now explain how to check for the condition $f(l^+) > f_M(l^+)$ from Step 2 in the above algorithm. If $f(l) > f_M(l)$ or $f(l) < f_M(l)$, i.e. the inequality is strict, then the comparison between $f(l^+)$ and $f_M(l^+)$ gives the same result.

So we focus on the case when $f(l) = f_M(l)$. Here, we need to check the distance function d_l stored in the interval of M whose left endpoint is l. Recall that d_l is either the square root of a quadratic function in x and y (when it is the distance to a point plus some constant) or a linear function in x and y (when it is the distance to a line). We take $f(l^+) > f_M(l^+)$ to be true if (and only if): (i) the distance function $d_l = -\infty$, or, (ii) the derivative of d_l at point l is strictly less than the derivative of f at point l.

Finding a "cross-over" point. Each node of the list M stores a distance function (distance to point plus a constant or the distance to a line). We can therefore compare the functions to check if the function stored in M becomes greater than the value stored in the function f within the domain where both functions are defined. The point where this happens is identified as the "cross-over" point.

Runtime: Each edge of T is handled once, and causes at most one new interval to be inserted into M, so the total number of endpoints inserted into M is $O(n)$. We can access $f_M(l^+)$ in constant time using the pointer p_u. Then the endpoints of intervals of M that we traverse as we do the insertion vanish from M. The runtime is thus $O(n)$.

Correctness: By definition, the upper envelope of the distance functions of the coarse cover elements of ab is the geodesic edge radius. We will prove that our algorithm only discards cover elements (partially or fully) in intervals where their function value is exceeded by some other element. The discarded parts are thus not relevant for finding the upper envelope. At any point during the execution of the algorithm, M is a list of coarse cover elements with disjoint intervals on ab whose union is an interval starting at a. All the coarse cover elements that remain in the end are part of M and hence M must be the upper envelope. The following lemma completes the proof of correctness:

Lemma 50. *The algorithm only discards pieces of coarse cover elements that do not form part of the final upper envelope.*

Proof. We examine the behaviour of the algorithm for edge uv of T with associated coarse cover element (I, f, e), where $I = [l, r]$. We insert the subinterval $[l, t]$ into M (or no subinterval). Because $f(x) \geq f_M(x)$ for $x \in [l, t]$, any subintervals of M that are removed due to the insertion do not determine the upper envelope, so their removal is correct.

If we insert all of interval I into M and recursively call Insert(v, p_v), then this is correct by induction. So suppose we insert a proper subinterval of I or none of I. We must prove that no later part of I, and no element of the coarse cover associated with edges of the subtree rooted at v determines the upper envelope. Let t^+ be a point just to the right of t (or just to the right of l if we insert no part of I). Then $f_M(t^+) > f(t^+)$. Number the polygon edges $e_1, e_2, \ldots, e_m$ clockwise from a to b. Suppose that $e = e_i$, so $f(x) = d(x, e_i)$ for $x \in I$, in particular, $f(t^+) = d(t^+, e_i)$. Suppose that $f_M(t^+) = d(t^+, e_k)$. Then $d(t^+, e_k) > d(t^+, e_i)$.

Now consider the edges of T descended from v plus the edge uv. Consider the corresponding coarse cover elements, C_v, and let e_j be any polygon edge associated with any element in C_v. Note that the intervals on ab associated with coarse cover elements of C_v lie to the right of r, except for I associated with uv. We will prove that for any point $x \in ab$ to the right of t^+, $d(x, e_k) > d(x, e_j)$, which implies that none of the coarse cover elements in C_v determines the upper envelope, nor does any part of I to the right of t. Thus the algorithm is correct to discard them.

We first prove the result for $x = t^+$. If uv corresponds to a central triangle/trapezoid for e_i or a b-side triangle, then T has a single path descending from v, all of whose edges

are associated with e_i, i.e., $j = i$. Otherwise, by the definition of an a-side coarse cover element, e_i corresponds to the farthest leaf of T_a descended from v, which implies that $d(x, e_i) \geq d(x, e_j)$ for all $x \in I$, and in particular for $x = t^+$. Thus, in either case we have $d(t^+, e_k) > d(t^+, e_i) \geq d(t^+, e_j)$.

We next claim that $k < j$. The current f_M values arise from tree edges already processed. These consist of: (1) edges on the path from a to u; and (2) edges of T counterclockwise from this path. Edges on the path from a to u have coarse cover intervals on ab to the left of l, by Observation 49. Thus type (1) edges do not determine $f_M(t^+)$. By the depth-first-search order, type (2) edges have coarse cover elements corresponding to polygon edges counterclockwise from e_j. Thus $k < j$.

To complete the proof of Lemma 50, consider any point $x \in ab$ to the right of t^+. The clockwise ordering around the polygon boundary is x, t^+, e_k, e_j, so by Lemma 26 and the fact that $d(t^+, e_k) > d(t^+, e_j)$, we get $d(x, e_k) > d(x, e_j)$, as required. □

6.4.2.3 Examples of the Algorithm

Example 1. Let us explain our Voronoi construction algorithm for Figure 6.13. We have here a convex polygon with edges e_1, e_2, e_3 and ab. The shortest path trees rooted at a and b are shown.

The enhanced shortest path tree T_a in Figure 6.14 has edges a_0, a_1, a_2, a_3, where a_0 and a_2 are the added 0-length edges. T_a has three leaves corresponding (in clockwise order) to polygon edges e_3, e_2, e_1. The enhanced shortest path tree T_b has edges b_0, b_1, b_2, b_3, where b_0 and b_2 are the added 0-length edges. T_b has three leaves corresponding (in clockwise order) to polygon edges e_3, e_2, e_1. The perpendiculars to e_2 at its endpoints v and u meet ab at p_1 and p_2 respectively. The angle bisectors of $\angle avu$ and $\angle buv$ meet ab at t_2 and t_1 respectively.

In this example, we denote the coarse cover elements by c_l where l is either the tree edge for which the element is defined (for a-side triangles, b-side triangles, or central triangles) or a polygon edge (for central trapezoids).

First we list the elements of the coarse cover for Figure 6.14:

1. a-side triangles:

   $\{c_{a_2} = (I_{a_2} = ap_1, f = d_2(x, v), e_2)\}$
2. b-side triangles:

   $\{c_{b_2} = (I_{b_2} = p_2b, f = d_2(x, u), e_2)\}$

3. central triangles:

   $\{\}$

4. central trapezoids:

   $\{c_{e_3} = (I_{e_3} = ab, f = d_2(x, \bar{e}_3), e_3), c_{e_2} = (I_{e_2} = p_1p_2, f = d_2(x, \bar{e}_2), e_2),$
   $c_{e_1} = (I_{e_1} = ab, f = d_2(x, \bar{e}_1), e_1)\}$

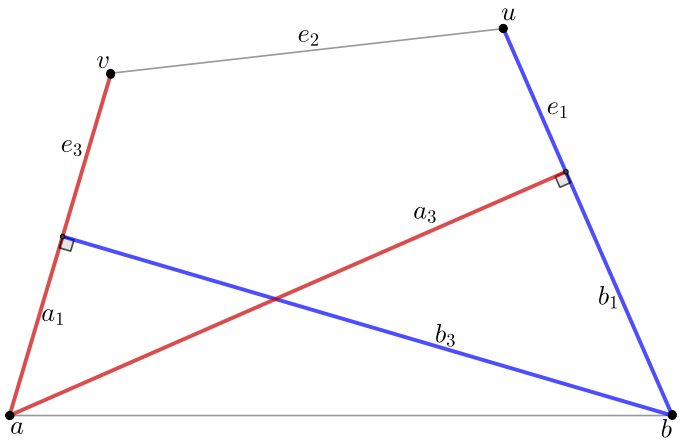


Figure 6.13: A simple example for which we demonstrate the algorithm for the upper envelope. The underlying polygon is a convex 4-gon. The shortest path trees rooted at a and b are shown in red and blue respectively.

Next, as in Corollary 48, we denote by $C(e)$ the set of coarse cover elements (I, f, e) for the edge e. Ordering these sets as in Corollary 48 we have:

1. $C(e_3) = \{c_{e_3}\}$
2. $C(e_2) = \{c_{a_2}, c_{e_2}, c_{b_2}\}$
3. $C(e_1) = \{c_{e_1}\}$

Finally, the coarse cover elements are traversed in the following DFS order according to tree T (see Figure 6.15):

1. c_{e_3}: Since M is initially empty, c_{e_3} is added to the entire interval ab. After adding this element,

   $M = (ab, d_2(x, \bar{e}_3), e_3)$.

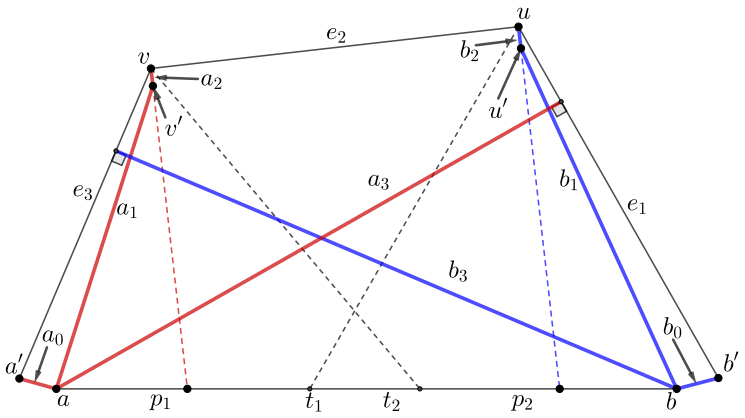


Figure 6.14: The enhanced shortest path trees for our DFS based algorithm. The underlying polygon shown in Figure 6.13 is a convex 4-gon, but this figure looks more complex since we explicitly show the 0-length edges a_0, a_2, b_0, and b_2.

2. c_{a_2}: This replaces the initial part of M. As the angle bisector of $\angle avu$ meets ab at t_2, points in the interval ap_1 are farther away from e_2 than from e_3. After adding c_{a_2},

   $M = (ap_1, d_2(x, v), e_2)$
   $(p_1b, d_2(x, \bar{e}_3), e_3)$.

   As all of c_{a_2} is added, we continue on to c_{e_2}, with the pointer pointing to the element of M at p_1.

3. c_{e_2}: Starting out at p_1, this replaces some part of the element $(p_1b, d_2(x, \bar{e}_3), e_3)$ of M. Note however that the angle bisector of e_3 and e_2 meets ab at t_2. As points on the angle bisector are equidistant from both sides, t_2 is a cross-over point. After handling c_{e_2},

   $M = (ap_1, d_2(x, v), e_2)$
   $(p_1t_2, d_2(x, \bar{e}_2), e_2)$
   $(t_2b, d_2(x, \bar{e}_3), e_3)$.

   As we do not add c_{e_2} completely, we halt the DFS at t_2.

4. c_{b_2}: Not examined.

5. c_{e_1}: This element has its left endpoint at a and is added until the crossover point with e_2 is reached. By the construction of Figure 6.14, this crossover point t_1 is reached at the second element of M. So, the first element of M is completely replaced, while the interval of the second element is split. After handling c_{e_1},

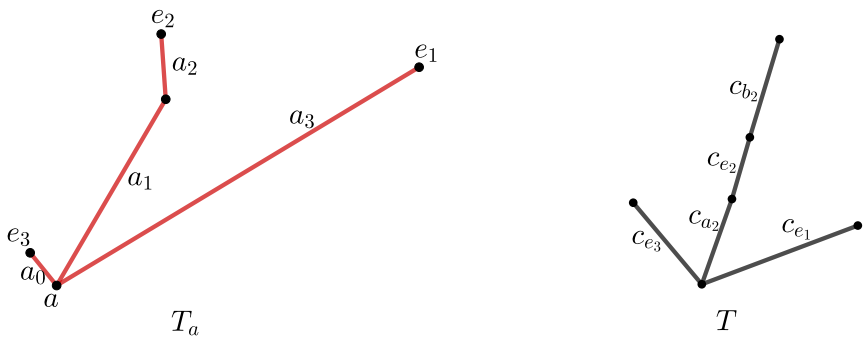


Figure 6.15: The tree in red is T_a, the shortest path tree from Figure 6.14 rooted at a. The tree in black is the tree T that decides the DFS order of traversal for the coarse cover elements. See the algorithm in Section 6.4.2.2.

$M = (at_1, d_2(x, \bar{e}_1), e_1)$
$(t_1t_2, d_2(x, \bar{e}_2), e_2)$
$(t_2b, d_2(x, \bar{e}_3), e_3)$.

The execution of the algorithm is now complete and the farthest edge Voronoi diagram on ab is stored in the list M, whose contents are shown below:

$(at_1, d_2(x, \bar{e}_1), e_1)$
$(t_1t_2, d_2(x, \bar{e}_2), e_2)$
$(t_2b, d_2(x, \bar{e}_3), e_3)$.

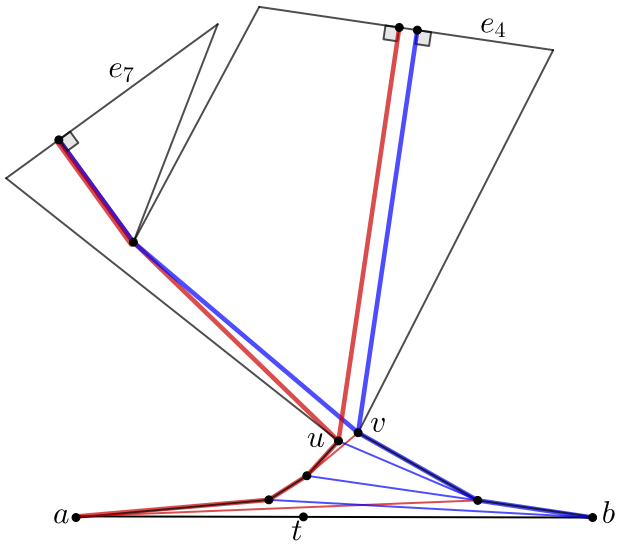


Figure 6.16: Another simple example for the algorithm for constructing the upper envelope. The shortest path trees rooted at a and b are shown in red and blue respectively. We have constructed the diagram so that the farthest edge from points on ab is either e_4 or e_7. Tree edges not on the paths to these two edges will be ignored. The crossover point t on ab is at equal geodesic distance from e_4 and e_7. Our Voronoi diagram algorithm will assign the interval at to e_4 and tb to e_7.

Example 2. We now explain our algorithm for constructing the farthest edge Voronoi diagram for Figure 6.16. It is constructed so that the only farthest edges from points on ab are e_4 and e_7. The crossover point t is also shown in the figure. The farthest edge from u in its subtree in T_a is e_4, i.e., $F_a(u) = e_4$. The farthest edge from v in its subtree in T_b is e_7, i.e., $F_b(v) = e_7$.

Only the edges e_4 and e_7 have coarse cover elements relevant to the current example. Therefore, we will ignore some coarse cover elements to simplify the description.

We show the intervals for the coarse cover elements in Figure 6.17. We avoid writing the actual distance functions and only indicate those as $d(x, e_4)$ or $d(x, e_7)$. The coarse cover elements under consideration will be:

1. a-side triangles:

   $\{c_{a_2} = (I_{a_2}, d(x, e_4), e_4), c_{a_3} = (I_{a_3}, d(x, e_4), e_4), c_{a_4} = (I_{a_4}, d(x, e_7), e_7), c_{a_6} = (I_{a_6}, d(x, e_4), e_4)\}$

2. b-side triangles:

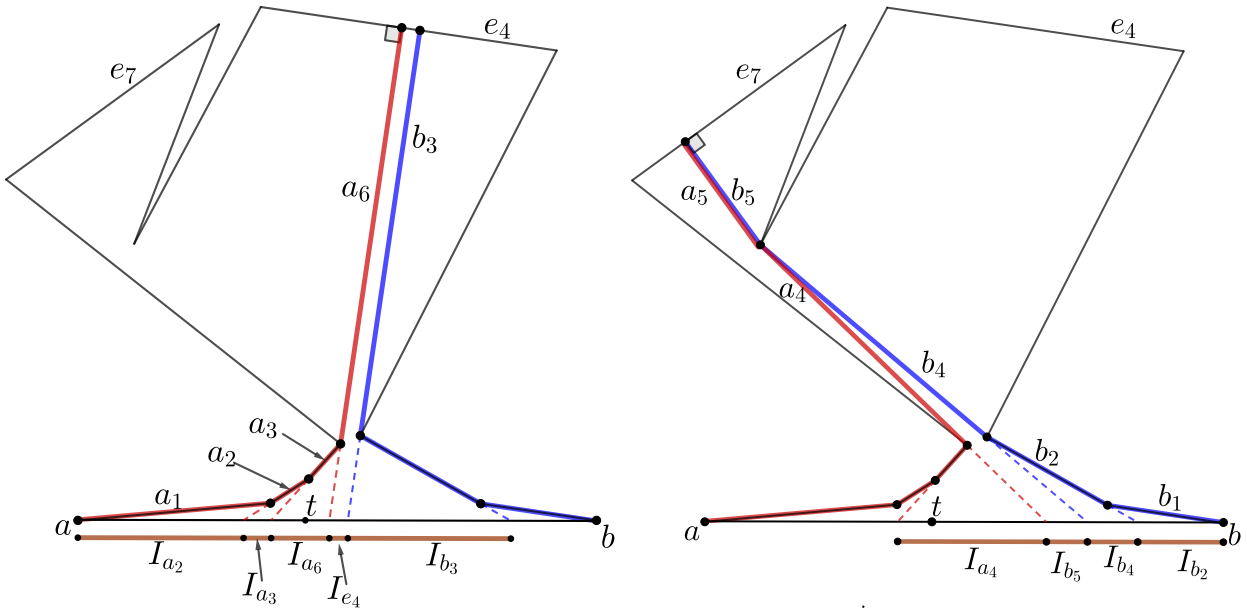


Figure 6.17: (a) Coarse cover elements for e_4. The crossover point t lies in I_{a_6}. Only the relevant parts of the shortest path trees are shown. (b) Coarse cover elements for e_7. The crossover point t lies in I_{a_4}. Only the relevant parts of the shortest path trees are shown.

$$\{c_{b_2} = (I_{b_2}, d(x, e_7), e_7), c_{b_3} = (I_{b_3}, d(x, e_4), e_4), c_{b_4} = (I_{b_4}, d(x, e_7), e_7)\}$$

3. central triangles:

   $\{c_{b_5} = (I_{b_5}, d(x, e_7), e_7)\}$

4. central trapezoids:

   $\{c_{e_4} = (I_{e_4}, d_2(x, \bar{e_4}), e_4)\}$

Next, we list the coarse cover elements by the polygon edge using notation and ordering borrowed from Corollary 48:

1. $C(e_7) = \{c_{a_4}, c_{b_5}, c_{b_4}, c_{b_2}\}$
2. $C(e_4) = \{c_{a_2}, c_{a_3}, c_{a_6}, c_{e_4}, c_{b_3}\}$

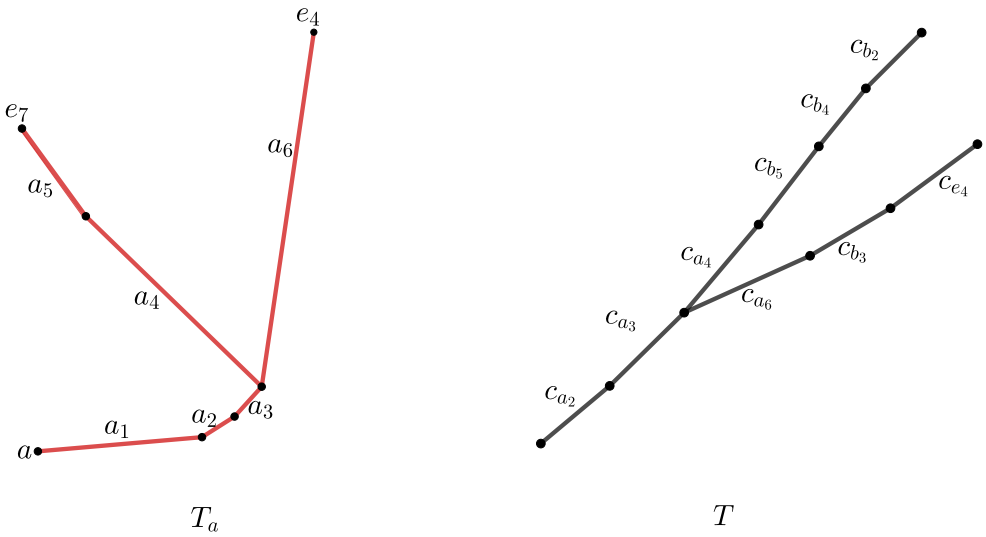


Figure 6.18: The tree in red is T_a, the shortest path tree from Figure 6.16 rooted at a. The tree in black is the tree T that decides the DFS order of traversal for the coarse cover elements. See the algorithm in Section 6.4.2.2.

Finally, the coarse cover elements are traversed in the following DFS order given by tree T (see Figure 6.18):

1. c_{a_2}: After handling c_{a_2},
   $M = (I_{a_2}, d(x, e_4), e_4)$.
2. c_{a_3}: After handling c_{a_3},
   $M = (I_{a_2}, d(x, e_4), e_4)$
   $(I_{a_3}, d(x, e_4), e_4)$.
3. c_{a_4}: The left endpoint of I_{a_4} coincides with the right endpoint of I_{a_3}. The coarse cover element corresponds to e_7 and is directly appended to the existing list M. After handling c_{a_4},
   $M = (I_{a_2}, d(x, e_4), e_4)$
   $(I_{a_3}, d(x, e_4), e_4)$
   $(I_{a_4}, d(x, e_7), e_7)$.
4. c_{b_5}: After handling c_{b_5},

$M = (I_{a_2}, d(x, e_4), e_4)$
$(I_{a_3}, d(x, e_4), e_4)$
$(I_{a_4}, d(x, e_7), e_7)$
$(I_{b_5}, d(x, e_7), e_7)$.

5. c_{b_4}: After handling c_{b_4},

   $M = (I_{a_2}, d(x, e_4), e_4)$
   $(I_{a_3}, d(x, e_4), e_4)$
   $(I_{a_4}, d(x, e_7), e_7)$
   $(I_{b_5}, d(x, e_7), e_7)$
   $(I_{b_4}, d(x, e_7), e_7)$.

6. c_{b_2}: After handling c_{b_2},

   $M = (I_{a_2}, d(x, e_4), e_4)$
   $(I_{a_3}, d(x, e_4), e_4)$
   $(I_{a_4}, d(x, e_7), e_7)$
   $(I_{b_5}, d(x, e_7), e_7)$
   $(I_{b_4}, d(x, e_7), e_7)$
   $(I_{b_2}, d(x, e_7), e_7)$.

7. c_{a_6}: For the interval I_{a_6} of coarse cover element c_{a_6}, the left endpoint is the same as I_{a_4}. Thus we can directly start the comparison in list M at $(I_{a_4}, d(x, e_7), e_7)$. The crossover point lies in both I_{a_4} and I_{a_6}. Let the left and right endpoints of I_{a_4} be t' and t'' respectively. Note that the left endpoint of I_{a_6} must also be t'. As the crossover point occurs in the interior of I_{a_6}, the tree T will not be examined further. After handling c_{a_6}, the list is:

   $M = (I_{a_2}, d(x, e_4), e_4)$
   $(I_{a_3}, d(x, e_4), e_4)$
   $(t't, d(x, e_4), e_4)$
   $(tt'', d(x, e_7), e_7)$
   $(I_{b_5}, d(x, e_7), e_7)$
   $(I_{b_4}, d(x, e_7), e_7)$
   $(I_{b_2}, d(x, e_7), e_7)$.

8. c_{e_4}: Not examined.

9. c_{b_3}: Not examined.

The final list M (and also the Voronoi diagram on ab for this example) is:

$(I_{a_2}, d(x, e_4), e_4)$
$(I_{a_3}, d(x, e_4), e_4)$
$(t't, d(x, e_4), e_4)$
$(tt'', d(x, e_7), e_7)$
$(I_{b_5}, d(x, e_7), e_7)$
$(I_{b_4}, d(x, e_7), e_7)$
$(I_{b_2}, d(x, e_7), e_7)$.

6.5 Phase II: Overview

In Phase I we found the farthest edge Voronoi diagram on the boundary of P. In Phase II we find the geodesic edge center. Phase I finds the edge center if it lies on ∂P. Therefore, we assume in this phase that the geodesic edge center lies in the interior of P. At a high-level we follow the algorithm of Ahn et al. [4] though with some significant differences. The first step is to find a coarse cover (see Definition 37) of the polygon by triangles. The details of this step are in Section 6.6. The triangles of the coarse cover will be bounded by two chords and one piece of a polygon edge. The problem of finding the center is thus reduced to finding a point that minimizes the upper envelope of the functions of the coarse cover. We then use a divide-and-conquer algorithm to solve subproblems of the following form: Given a subpolygon and a subset of the functions (each defined on a triangle of the original polygon P), find a point in the subpolygon to minimize the upper envelope of the functions. At each stage of the divide-and-conquer algorithm the size of the subproblem (the size of the subpolygon and the number of functions) is reduced by a constant fraction. We use ϵ-net techniques to recursively solve subproblems until the size of the subpolygon becomes constant, at which time we switch to a prune-and-search technique.

In the remainder of this overview section, we give some more details of the ϵ-net techniques that are used for subproblems with large subpolygons, and how our approach differs from that of Ahn et al. The subpolygons we work with are "3-anchor hulls"— the geodesic convex hulls of at most three points or subchains of the polygons boundary.

We prove that if the size of a 3-anchor hull Q is greater than 6, then no triangle of the coarse cover contains it. Thus any coarse cover triangle that intersects Q has a bounding chord that intersects Q, and we can focus on the chords $\mathcal{K}$ that define triangles of the coarse cover.

To apply ϵ-nets we define a "3-anchor range space" whose ground set is $\mathcal{K}$, and whose ranges are sets of chords that intersect 3-anchor hulls. The idea is to find an ϵ-net, which is a constant-sized set $N \subseteq \mathcal{K}$ such that any 3-anchor hull not intersected by a chord of N is intersected by at most a constant fraction of the chords of $\mathcal{K}$—this is the important property that allows us to discard a fraction of the coarse cover triangles that properly intersect the subpolygon. The set of chords N forms an arrangement that partitions the current subpolygon into cells. We can use the chord oracle to determine which cell contains the center, but in order to recurse, we must subdivide this cell into a constant number of 3-anchor hulls and figure out which of these 3-anchor hulls contains the center point. The ingredients needed to carry out this plan are as follows.

1. Prove that the 3-anchor range space has finite VC-dimension (to show that an ϵ-net of constant-size exists). Provide a "subspace oracle" (to find an ϵ-net N in deterministic linear time). These ingredients are given in Section 6.7.
2. Subdivide a cell of the arrangement of N into a constant number of 3-anchor hulls. Find a simple 3-anchor hull that contains the center using a geodesic oracle (an extension of the chord oracle) to test which side of a geodesic path contains the center point. The geodesic oracle is given in Section 6.8. The divide-and-conquer algorithm, including details of how to subdivide a cell into 3-anchor hulls) is given in Section 6.9.

For now, we expand on the aspects of the algorithm that differ from the approach of Ahn et al. [4] (beyond the obvious difference that we deal with farthest edges rather than farthest vertices.). Instead of 3-anchor hulls, their algorithm works with 4-*cells*, formed by taking the intersection of at most four half-polygons, where a half-polygon is the part of P to one side of a chord. Figure 6.19 gives an example of a 4-cell and a 3-anchor hull. The number (three versus four) is not significant, but we bound our regions by geodesics instead of chords in order to obtain the following two results.

1. The 3-anchor range space has finite VC-dimension. This implies that constant-sized ϵ-nets exist. Furthermore, there is a "subspace oracle" that allows us to find an ϵ-net N in deterministic linear time [99, Chapter 47, Theorem 47.4.3]. For further background see Section 6.7.2.

   Ahn et al. claim that their range space (of chords crossing 4-cells) has finite VC-dimension but their proof is flawed. Our proof shows that their range space does in fact have finite VC-dimension. They do not mention subspace oracles, without which their algorithm runs in expected linear time rather than deterministic linear time as claimed. We expand on these aspects in Section 6.7.1 below.

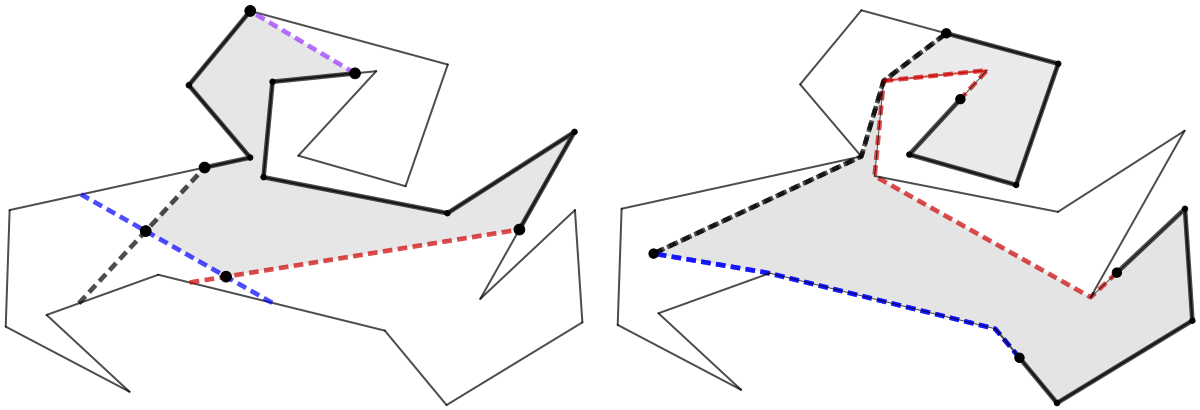


Figure 6.19: (left) The shaded region shows a 4-cell in P. The dashed segments are the four chords that define half-polygons whose intersection is the 4-cell. The 4-cell is the hull of two polygon chains (shown in bold) and two points in the interior. (right) The shaded region shows a 3-anchor hull. It is the geodesic convex hull of the two polygon boundary chains shown in bold and a point in the interior.

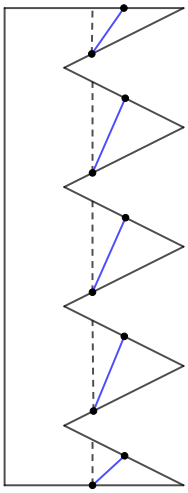


Figure 6.20: Figure outlining the problem with Ahn et al.'s proposed vertical decomposition of the polygon into 4-cells. The blue chords represent an ϵ-net of their range space. Using their decomposition scheme, the polygon is not partitioned into 4-cells (the subpolygon bounded by the five vertical dashed chords is a 5-cell), contrary to their claim.

2. A cell of the arrangement of N can be partitioned into constantly many 3-anchor hulls.

   The method used by Ahn et al. to subdivide a cell of N into 4-cells by adding a constant number of chords is incomplete. From the intersection points and endpoints of the chords in N, they shoot vertical rays up and down until either a chord of N or the boundary of the outer 4-cell is reached. They claim that this subdivides each face into a constant number of 4-cells. It is true that there are a constant number of regions, but not true that the regions are 4-cells. Figure 6.20 shows a counterexample. There are five chords in N, and their construction leaves a 5-cell. The partition step can be fixed with an alternate approach but we find 3-anchor hulls more natural.

6.6 Phase II, Part 1: Finding a Coarse Cover of the Polygon by Triangles

This section contains the first step of Phase II, which is to construct in linear time a ***coarse cover*** of the polygon as specified in Definition 37. In later sections we show how to search the coarse cover to find the geodesic edge center.

Let X be the set of vertices of the farthest edge Voronoi diagram on the boundary of P. These are points on ∂P that each have two farthest edges. The points of X partition ∂P into ***chains*** $\boldsymbol{C(e)}$, where $C(e)$ consists of the points on ∂P whose farthest edge is e. We begin by expanding each chain $C(e)$ to a subpolygon $Y(e)$ that contains the Voronoi region of the edge e. After that, we will partition each polygon $Y(e)$ into triangles to obtain the coarse cover.

Suppose the chain $C(e)$ goes clockwise from $p(e) \in X$ to $q(e) \in X$. The ***edge funnel*** $\boldsymbol{Y(e)}$ is the polygon bounded by the chain $C(e)$, the ***walls*** $\pi(p(e), e)$ and $\pi(q(e), e)$, and the ***base*** $b(e)$, which is the part of e between the terminal points $t(p(e), e)$ and $t(q(e), e)$. See Figure 6.21. The ***size*** of the edge funnel $Y(e)$ is its number of vertices. By Property **(P1)**, the walls of an edge funnel may merge but never cross, so each edge funnel is a weakly simple polygon.

Edge funnels are an extension of the well-studied [vertex] funnels that are used for computing shortest paths (see [66, 49]), and that were used by Ahn et al. [4] to compute the geodesic vertex center. To build their coarse cover of the polygon, Ahn et al. needed hourglasses as well as [vertex] funnels, so their method was more complicated. By contrast, our coarse cover is constructed from edge funnels alone because we did extra work ahead

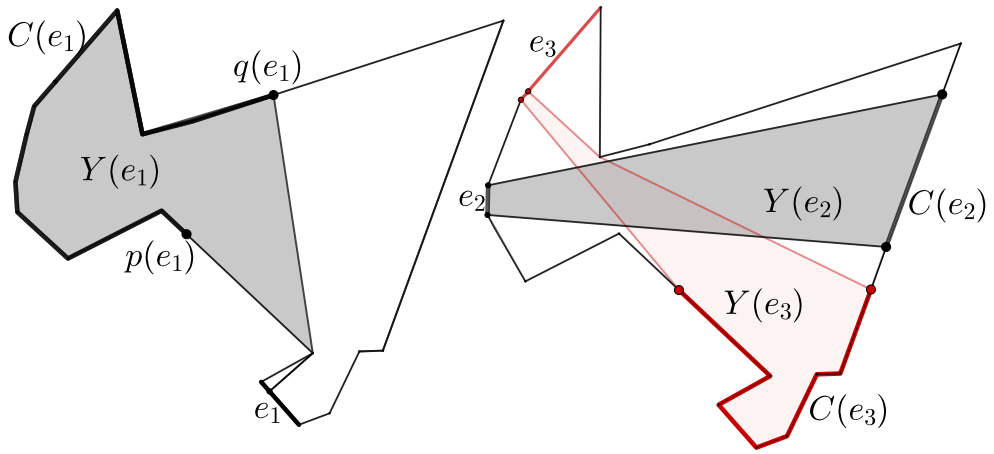


Figure 6.21: Edge funnels $Y(e_i)$ for some edges e_i of a polygon. The chains $C(e_i)$ on ∂P are indicated in bold.

of time using hourglasses to compute the farthest edge Voronoi diagram restricted to the polygon boundary.

Lemma 51. *For any point $p \in P$, if e is a farthest edge from p, then $p \in Y(e)$.*

Proof. Consider the shortest path $\pi(p, e)$ from the point p to the edge e and extend the first segment of the path backwards until it intersects the boundary ∂P at point p'. Since the result is a locally shortest path, it must be the shortest path from p' to e. Thus the distance from p' to e is $|p'p| + d(p, e)$. We now show that e is a farthest edge from p'. Consider any other edge e'. We have $d(p, e') \leq d(p, e)$. Then $d(p', e') \leq |p'p| + d(p, e') \leq |p'p| + d(p, e) = d(p', e)$. Thus e is a farthest edge from p', so p' lies on the chain $C(e)$. By Property **(P1)** of Lemma 25, $\pi(p', e)$ does not cross the walls of the edge funnel $Y(e)$, so it lies inside $Y(e)$. Therefore, the point p lies in $Y(e)$ since p is a point on $\pi(p', e)$. □

Lemma 52. *The set of edge funnels $Y(e)$ corresponding to all the edges e of the polygon can be constructed in $O(n)$ time. The sum of all their sizes is $O(n)$.*

Proof. The farthest edge Voronoi diagram on ∂P gives us the chains $C(e)$, so we only need to find the walls of the edge funnels, which are the shortest paths from the endpoints of $C(e)$ to e. Equivalently, we must find, for each Voronoi vertex $p \in X$, the shortest paths to p's farthest edges. Note that there are $O(n)$ points in X, and by Lemma 24, each $p \in X$ has two farthest edges.

Recall that by Lemma 35 there is a linear time algorithm to find a set of five farthest edge separators such that for every point $p \in \partial P$, one of the separators has p to its right and, by definition of a farthest edge separator, has all farthest edges of p to the left. It therefore suffices to focus on one of these farthest edge separators $\gamma = \pi(a, b)$, and give a linear time algorithm to find the shortest path from each point $p \in X$ that lies to the right of γ to each of p's farthest edges. By Lemma 33, each such path $\pi(p, F(p))$ is contained (except for one edge) in the shortest path trees $T(a)$ and $T(b)$. By Lemma 34, after a linear time preprocessing of the trees $T(a)$ and $T(b)$, each path $\pi(p, F(p))$ can be found in time proportional to its size $|\pi(p, F(p))|$. Finally, we note that the two walls of one edge funnel may share edges, but we claim that walls of different edge funnels do not share edges if they originate from points in X to the right of γ. Consider $p, q \in X$ and the paths $\pi(p, F(p))$, $\pi(q, F(q))$ with $F(p) \neq F(q)$. By Claim 32, the paths cross, and then Property **(P4)** implies that the paths do not share any edges (chords).

We therefore have $\sum |\pi(p, F(p))| \in O(|T(a)| + |T(b)| + n)$, where the last term accounts for the one edge of each path that is not in the trees. Thus the total run time to find all the shortest paths is $O(n)$. □

Defining the coarse cover $\mathcal{T}$ of polygon P. The idea is to partition each funnel $Y(e)$ into triangles in time linear in $O(|Y(e)|)$, and then take the union over all funnels. We first use Lemma 30 to partition $Y(e)$ in linear time into its *shortest path map* from its base edge $b(e)$. Recall that the shortest path map partitions $Y(e)$ into regions such that shortest paths to e from points in the same region are combinatorially the same. In addition, if a region of the shortest path map contains any vertex v whose shortest path to e splits the region, then we subdivide the region at the path. All these subdivisions can be found in linear time, and the resulting subdivided regions T are either triangles or trapezoids. See Figure 6.6b.

Next we define distance functions on the triangles and trapezoids. If T is a triangle, then the shortest path to e from any point $p \in T$ goes through an apex v of the triangle, and the distance from p to e is $f_T(p) := d_2(p, v) + \kappa$ where d_2 is Euclidean distance (ignoring P) and κ is $d(v, e)$ which is independent of p. If T is a trapezoid, then the shortest path to e from any point p of T is a straight line segment meeting e at right angles, and the distance from p to e is $f_T(p) := d_2(p, \bar{e})$, where $\bar{e}$ is the line through e and d_2 is Euclidean distance (ignoring P). For the convenience of not having to say "triangles and trapezoids," we will further partition each trapezoid into two triangles, each inheriting a distance function of the form $d_2(p, \bar{e})$.

Define $\boldsymbol{\mathcal{T}(e)}$ to contain the triple (T, f_T, e) for each triangle T in the partition of $Y(e)$. Along with the triples we store the following:

1. Each triangle T is given by its three sides: one side is a subsegment of an edge and the other two are chords (recall that a chord may include, or be, an edge of P). A chord is given by its two endpoints and the vertex/edge containing each endpoint.
2. Furthermore, we store a list of chords used as sides of triangles of $\mathcal{T}(e)$, and for each chord, list the one or two triangles it is a side of. Each chord is given by its two endpoints on ∂P.

So, by Lemma 30, we have:

Claim 53. *$\mathcal{T}(e)$ can be computed in time $O(|Y(e)|)$, and has size $O(|Y(e)|)$.*

Define $\boldsymbol{\mathcal{T}}$ to be $\bigcup_e \mathcal{T}(e)$. We prove that $\mathcal{T}$ is a coarse cover of P according to Definition 37. (Note that a chord may appear as a side of a triangle in more than one $\mathcal{T}(e)$. We could, in fact, identify these, but instead we simply allow duplicates in the list of chords.)

Lemma 54. *The set $\mathcal{T}$ is a coarse cover of P. Furthermore, $\mathcal{T}$ has size $O(n)$ and can be computed in time $O(n)$.*

Proof. From Lemma 52, we can construct all the edge funnels $Y(e)$ in time $O(n)$, and from Claim 53, we can compute $\mathcal{T}(e)$ in time $O|Y(e)|)$. Thus we can compute $\mathcal{T}$ in time $O(n)$.

To prove that $\mathcal{T}$ is a coarse cover, first observe that the functions f_T have the correct form. By Lemma 51, for any point $p \in P$, if e is a farthest edge from p, then p is in the edge funnel $Y(e)$ so p is contained in some triangle T in the partition of $Y(e)$, and is therefore contained in a triple (T, f_T, e) of $\mathcal{T}(e)$.

□

Next, we make two observations about the triangles of the coarse cover we have constructed. The first one follows from our construction using the shortest path maps from edges, and is stated without proof.

Observation 55. *Each coarse cover triangle has three sides—one is a subsegment of an edge of P, and two are chords of P. Additionally, each chord is a side of one or two coarse cover triangles.*

Lemma 56. *Each triangle of the coarse cover contains at most three vertices of P.*

Proof. First, observe that no vertex of P is internal to T. Since P does not have 3 collinear vertices (by Assumption 1), each side of T contains at most 2 vertices of P. Furthermore, one side of T—call it s_1—is a subsegment of an edge of P, so it cannot contain vertices in its interior. Triangles of the coarse cover either have a vertex of P as an apex opposite s_1, or arise from subdividing a trapezoid (see Section 6.6). In the first case, T has at most one more vertex on each side incident to the apex for a total of at most three vertices of P. In the second case, T has a side that is a diagonal of a trapezoid and contains no vertices in its interior, though it may have a vertex of P at its intersection with s_1, and the third side of T has at most two vertices of P, for a total of at most three vertices of P. Thus T contains at most three vertices of P. □

6.7 Phase II, Part 2: Divide and Conquer via ϵ-Nets

At this point, we have, from Section 6.6, a coarse cover of the polygon P of size $O(n)$. The problem of finding the geodesic edge center reduces to the problem of finding the point in P that minimizes the upper envelope of the functions given by the coarse cover. As described in Section 6.5, we now use divide-and-conquer to solve subproblems where we have a subpolygon whose interior contains the center (intially all of P) and a coarse cover of the subpolygon. At each step the size of the subproblem is reduced by a constant fraction. The algorithm has two stages. The first stage applies while no triangle of the coarse cover contains the subpolygon (this is true initially). It uses ϵ-net techniques. The second stage applies once the subpolygon reaches constant size—only then can coarse cover triangles contain the subpolygon—and uses a Megiddo-style prune-and-search algorithm. The overall algorithm is in Section 6.9.

In this section we develop the necessary machinery of ϵ-nets for use in the first stage of the algorithm. A basic overview of ϵ-nets for geometric divide-and-conquer was given in Section 4.3. In Section 6.7.1, we give the specific range space for which ϵ-nets are determined, the "3-anchor range space", and its properties. All of our results apply to the vertex center and remedy the gaps in the algorithm of Ahn et al. [4]. These gaps and the differences between their approach and ours are described in further detail in Subsection 6.7.2.

6.7.1 The 3-Anchor Range Space

In this subsection we define our range space and prove the properties necessary for efficiently constructing constant size ϵ-nets via Lemma 8, namely finite VC-dimension and the existence of a subspace oracle.

Roughly speaking, the ground set of our range space is the set of chords that form boundary edges of the triangles in the coarse cover $\mathcal{T}$ from Section 6.6, and the ranges are the subsets of these chords that intersect certain "cells" of the polygon.

More precisely, consider the set of all the triangles of the coarse cover. By Observation 55, each triangle has three sides where one is a subsegment of an edge of P and the other two are chords of P. Let $\boldsymbol{\mathcal{K}}$ be the set of these chords. The ***ground set*** of our range space is $\mathcal{K}$. Observe that $\mathcal{K}$ has size $O(n)$ since there are $O(n)$ triangles in the coarse cover. Recall that we know the two endpoints of each chord of $\mathcal{K}$, and which edge/vertex of ∂P contains the endpoint.

The subpolygons we work with are **3-*anchor hulls*** defined as follows. An ***anchor*** is a point inside P, or a subchain of ∂P. A **3-*anchor hull*** is the geodesic convex hull of at most three anchors. These are weakly simple in general, but we only recurse on ***simple*** 3-anchor hulls.

We make the following observations and definitions about 3-anchor hulls.

Observation 57. *Let Q be a 3-anchor hull.*

1. *Q is a closed connected weakly-simple polygon, and is **geodesically convex in P**, meaning that for any two points a and b in Q, the geodesic path from a to b in P is contained in Q. This implies that the intersection of Q with a chord [geodesic] of P is a chord [geodesic] of Q.*
2. *The boundary of Q consists of: the at most three anchors that are subchains of ∂P; and at most three geodesic paths between pairs of anchors.*
3. *If v is a vertex of Q but not a vertex of P, then v is a point anchor or the endpoint of an anchor chain that is interior to an edge of P. In either case, v is a convex vertex of Q.*

We define the **3-*anchor range space*** as follows. For the ground set $\mathcal{K}$, and for each 3-anchor hull H of P there is a range $\mathcal{K}(H)$ consisting of all chords of $\mathcal{K}$ that ***cross*** H. Here a chord ***crosses*** a set if both its open half-polygons contain points of

the set (See Figure 6.22). Thus, the set of ***ranges*** on $\mathcal{K}$ is defined as $\boldsymbol{\mathcal{R} = \{\mathcal{K}(H) \mid H \text{ is a 3-anchor hull}\}}$. The ***3-anchor range space*** is $(\mathcal{K}, \mathcal{R})$.

Finite VC-Dimension. We prove that the 3-anchor range space has finite VC-dimension in the following lemma.

Lemma 58. *The 3-anchor range space has VC-dimension less than 259.*

Proof. We will prove that the shattering dimension is 6 and then apply the result that a range space with shattering dimension d has VC-dimension bounded by $12d \ln(6d)$ (Lemma 7). For $d = 6$ this is < 259.

We must show that for a set $\mathcal{K}$ of chords with $|\mathcal{K}| = m$, the number of distinct ranges is $O(m^6)$. We prove that the range space for $\mathcal{K}$ is the same if we replace 3-anchor hulls by "expanded 3-anchor hulls" that are defined in terms of $\mathcal{K}$, more precisely, in terms of the arrangement $A(\mathcal{K})$ of the chords $\mathcal{K}$ plus the edges of P. The arrangement $A(\mathcal{K})$ has $O(m^2)$ internal faces and one external face. By an "internal" edge/vertex, we mean an edge or vertex on a chord of $\mathcal{K}$ (i.e., we exclude the segments of ∂P.) There are $O(m^2)$ internal edges and vertices. Define an ***expanded anchor*** to be an internal face, edge, or vertex of $A(\mathcal{K})$, or a polygon chain with endpoints in $V(\mathcal{K})$, the set of endpoints of chords $\mathcal{K}$. An ***expanded 3-anchor hull*** is the geodesic convex hull of at most three expanded anchors.

Lemma 59. *The set of ranges $\mathcal{R} = \{\mathcal{K}(Q) \mid Q$ is a 3-anchor hull $\}$ is the same as the set of ranges $\overline{\mathcal{R}} = \{\mathcal{K}(Q) \mid Q$ is an expanded 3-anchor hull $\}$.*

Proof. To prove $\mathcal{R} \subseteq \overline{\mathcal{R}}$, consider a 3-anchor hull Q. Replace any point anchor p by the smallest (by containment) internal vertex, edge, or face of $A(\mathcal{K})$ that contains p. See Figure 6.22. Replace any polygon chain anchor C by the smallest chain of ∂P containing C and with endpoints in $V(\mathcal{K})$. Let $\psi(Q)$ be the geodesic convex hull of these expanded anchors. Then $\psi(Q)$ is an expanded 3-anchor hull that contains Q, and it is straightforward to prove that $\mathcal{K}(Q) = \mathcal{K}(\psi(Q))$

Claim 60. *Let Q be a 3-anchor hull and $\psi(Q)$ be the corresponding expanded 3-anchor hull for Q. Then a chord of $\mathcal{K}$ crosses Q if and only if it crosses $\psi(Q)$, i.e., $\mathcal{K}(Q) = \mathcal{K}(\psi(Q))$.*

Proof. One direction of the proof is simple: If a chord crosses Q, it must cross $\psi(Q)$ since $Q \subseteq \psi(Q)$.

For the other direction we prove that if a chord $K \in \mathcal{K}$ does not cross the 3-anchor hull Q, then it does not cross the expanded 3-anchor hull $\psi(Q)$. Suppose that a chord $K \in \mathcal{K}$

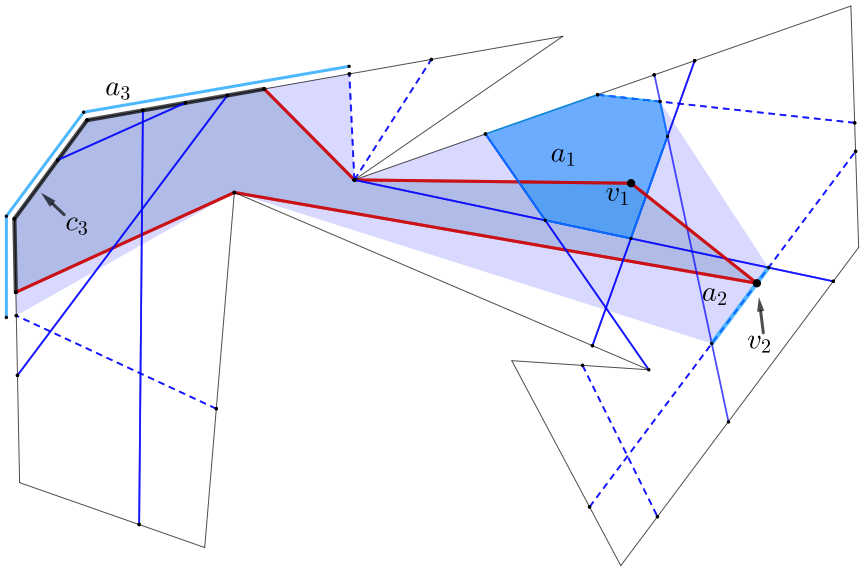


Figure 6.22: A simple 3-anchor hull Q with anchors v_1, v_2 and the polygon chain c_3. Solid blue chords cross Q, while dashed blue chords do not. The expanded 3-anchor hull $\psi(Q)$ (lightly shaded) has expanded anchors: a_1, the face in the chord arrangement containing v_1; a_2, the edge with v_2 in its interior; and a_3, the polygon chain extending c_3 to chord endpoints. The same chords cross Q and $\psi(Q)$.

does not cross Q. Then Q is contained in one of the closed half-polygons, say H, defined by K. This implies that the anchors of Q are contained in H. Since the corresponding expanded anchors were defined to not cross chords of $\mathcal{K}$, they are contained in H. Thus $\psi(Q)$, being the geodesic convex hull of sets in H, is also in H. So K does not cross $\psi(Q)$. □

To prove $\overline{\mathcal{R}} \subseteq \mathcal{R}$, let Q be an expanded 3-anchor hull. Replace an expanded anchor that is a face or edge of $A(\mathcal{K})$ by a point anchor in the interior of that face, edge. An expanded anchor that is a vertex or a polygon chain remains unchanged. Let $\gamma(Q)$ be the geodesic convex hull of the resulting anchors. Observe that $\gamma(Q)$ is a 3-anchor hull and $\psi(\gamma(Q)) = Q$. Thus by Claim 60, $\mathcal{K}(Q) = \mathcal{K}(\gamma(Q))$. □

To complete the proof of Lemma 58 we claim that the number of expanded 3-anchor hulls of $\mathcal{K}$ is $O(m^6)$. An expanded anchor may be an internal vertex, edge, of face of $A(\mathcal{K})$. It was shown above that the number of vertices, edges, and faces in $A(\mathcal{K})$ is $O(m^2)$. Otherwise, an expanded anchor is a chain of ∂P between vertices of $V(\mathcal{K})$. A chain between two points in $V(\mathcal{K})$ may be specified by its endpoints, again giving a total of $O(m^2)$ possibilities. Thus the number of expanded 3-anchor hulls is $O((m^2)^3) = O(m^6)$. □

Our 3-anchor range space has a subspace oracle as we prove in the next lemma.

Lemma 61. *The 3-anchor range space has a subspace oracle.*

Proof. We must provide a deterministic algorithm that, given a subset $\mathcal{K}' \subseteq \mathcal{K}$ with $|\mathcal{K}'| = m$, computes the set of ranges $\mathcal{R} = \{\mathcal{K}'(Q) \mid Q \text{ is a 3-anchor hull}\}$ in time $O(m^{d+1})$, where $d = 6$ is the shattering dimension of the 3-anchor range space.

We use the equivalence of the 3-anchor range space and the expanded 3-anchor range space (Lemma 59). In Lemma 58 we proved that the number of expanded 3-anchor hulls, Q, is $O(m^6)$. We must find these, and find, for each Q, the set of chords of $\mathcal{K}'$ that cross it.

Recall that $A(\mathcal{K}')$ is the arrangement of the chords of $\mathcal{K}'$ plus the edges of P. This is an arrangement of line segments, with the special property that all segment endpoints are on the outer face. Recall also that $V(\mathcal{K}')$ denotes the endpoints of the chords in $\mathcal{K}'$. If $\mathcal{K}'$ has size m, then $A(\mathcal{K}')$ has $O(m^2)$ faces, $O(m^2)$ internal vertices and edges, and $n + 2m$ external vertices and edges on the boundary of P. In particular, the external vertices are the vertices of P union $V(\mathcal{K}')$. For the algorithm we will avoid the dependence on n by

working with a combinatorial version of $A(\mathcal{K}')$ in which each minimal chain along ∂P with endpoints in $V(\mathcal{K}')$ is represented by a single "dummy edge". Note that the number of dummy edges is at most $2m$. Let $\boldsymbol{G(A(\mathcal{K}'))}$ denote this planar graph, which has $O(m^2)$ vertices, edges, and faces.

We compute $G(A(\mathcal{K}'))$ as follows. Compute the arrangement of the m line segments $\mathcal{K}'$ in time $O(m^2)$. Then traverse the outer face of the arrangement, adding dummy edges corresponding to subchains of ∂P between vertices of $V(\mathcal{K}')$. We thus compute $G(A(\mathcal{K}'))$ in time $O(m^2)$.

Next, we enumerate all of the possible expanded anchors: the $O(m^2)$ internal vertices, edges, and faces of $G(A(\mathcal{K}'))$, and the $O(m^2)$ polygon chains, each represented by two endpoints in $V(\mathcal{K}')$.

For each of the m chords K of $\mathcal{K}'$ we enumerate the anchors that lie in each of the two closed half-polygons H defined by K. In particular, we can traverse $G(A(\mathcal{K}'))$ in time $O(m^2)$ to find the vertices, edges, and faces that lie in H. We can also decide which of the $O(m^2)$ polygon chains lie entirely in H, based on where the endpoints lie. This takes time $O(m^2)$ per chord, for a total of $O(m^3)$.

Finally, we can enumerate all the $O(m^6)$ choices of at most three expanded anchors that determine an expanded 3-anchor hull Q. For each choice we spend $O(m)$ time to find the set of chords crossing Q—begin with all of $\mathcal{K}'$ and eliminate chords that have all three anchors on the same side, since these are precisely the chords do not cross Q. This gives us the set of chords crossing Q. □

6.7.2 Comparison to Ahn et al.

Our ϵ-net approach is based on the ϵ-net approach that Ahn et al. [4] used to find the geodesic center of the vertices of a polygon. In this subsection we describe their approach and the differences between their approach and ours. In particular, we explain some gaps in their approach and how we remedy them.

Their coarse cover of the polygon by "apexed triangles" is different from ours, because we deal with the *edge* center, but those differences do not matter for the ϵ-net. They define a ***4-cell range space***: the ground set is the same, namely the chords that define the triangles in the coarse cover, but they define a range as the chords that intersect a ***4-cell***, where a 4-cell is the intersection of at most four half-polygons, and a half-polygon is the part of the polygon to one side of a chord. By contrast, we defined ***3-anchor hulls*** to be the geodesic hull of at most three points or boundary chains of P.

Observation 62. *If Q is a 4-cell, then it is a 4-anchor hull.*

Proof. Around the boundary of Q, there are four chords (or segments of chords), with two consecutive ones joined by a polygon chain or meeting at a point. Q is the geodesic hull of these ≤ 4 polygon chains and points. □

The two issues with the 4-cell range space approach of Ahn et al. are as follows:

1. Their proof of finite VC-dimension is flawed.
2. They claim a deterministic algorithm to construct an ϵ-net but do not show that their range space has a subspace oracle. Furthermore, it is unlikely that their range space has a subspace oracle.

We expand on these issues and our remedies. For issue (1) they prove that the 1-cell range space has VC-dimension at most 65,535. They note that a 4-cell is the intersection of four 1-cells, and then claim in their Lemma 9.1 that this implies finite VC-dimension for the 4-cell range space. As justification, they refer to Proposition 10.3.3 of Matousek's text [77], which states that the VC-dimension is bounded for any family whose sets can be defined by a formula of Boolean connectives (union, intersection, set difference). However, Matousek's proposition cannot be applied in this situation because, although a 4-cell is the intersection of four 1-cells, it is not true that a 4-cell *range* is the intersection of four 1-cell *ranges*. In particular, a chord can intersect two 1-cells, but not intersect the intersection of the two 1-cells. For example, a line of slope -1 can intersect the $+x$ half-plane and the $+y$ half-plane without intersecting the $+x, +y$ quadrant.

We address issue (1) by defining ranges in terms of geodesic-cells, proving equivalence with anchor-cells, and proving finite shattering dimension (hence finite VC-dimension) for the anchor-cell ranges. Our proof of Lemma 58 works equally well for 4-anchor hulls—the bound becomes 372. A 4-cell is a special case of a 4-anchor hull (Observation 62), so our proof implies finite VC-dimension (≤ 372) for the 4-cell range space, which repairs the claim by Ahn et al.[3]

Regarding issue (2), designing a subspace oracle for the 4-cell range space of Ahn et al. seems problematic (compare Figures 6.23 and 6.24). However, the same proof as for Lemma 61 shows that the 4-anchor range space has a subspace oracle. Thus constant-sized ϵ-nets can be found in deterministic linear time. An ϵ-net for the 4-anchor range space is an ϵ-net for the 4-cell range space. This repairs the approach of Ahn et al., modulo their partitioning step that divides a 4-cell into a constant number of 4-cells (Figure 6.20).

[3] In response to our enquiries, Eunjin Oh independently suggested a similar remedy.

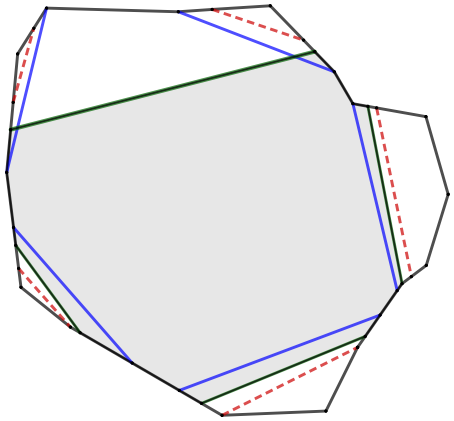


Figure 6.23: In this figure, we can construct a 4-cell (shaded) using the green chords that is crossed by only the blue chords (and not the dashed red chords). So, in the 4-cell range space, the set of ranges contains the subset consisting solely of the blue chords.

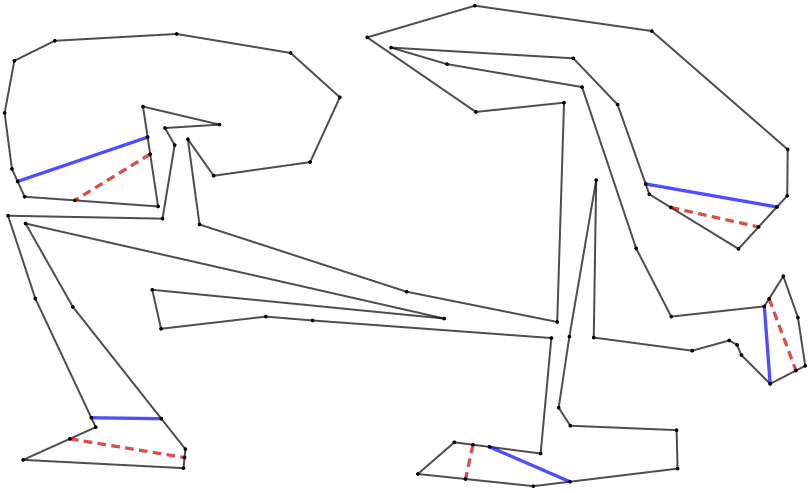


Figure 6.24: In this figure, we cannot construct a 4-cell that is crossed by only the blue chords (and not the dashed red chords). In the 4-cell range space, the set of ranges does not contain the subset consisting solely of the blue chords, even though this figure is combinatorially the same as Figure 6.23.

6.8 Phase II, Part 3: Geodesic Oracle

Recall that the chord oracle tells us which side of a chord contains the geodesic edge/vertex center of a polygon. The algorithm that implements the chord oracle in $O(n)$ time was described in Section 6.2.6. Recall that the first step is to find a coarse cover of the chord.

In this section we give two extensions of the chord oracle that we use in Phase II. Recall (see Section 6.5) that in Phase II we recurse on subpolygons that are simple 3-anchor hulls. The first extension of the chord oracle is a version that works on a chord of a subpolygon (a 3-anchor hull) and uses the coarse cover of the subpolygon to get a coarse cover of the chord. The runtime will be linear in the size of the subpolygon's coarse cover, which decreases during the divide-and-conquer algorithm. The second extension is a generalization of the chord oracle to a *geodesic oracle* that tells us which side of a geodesic path contains the center of the polygon. A geodesic path divides a polygon into regions, and the geodesic oracle tells us which region contains the center. We need this because our subpolygons are bounded by geodesics.

Let Q be a 3-anchor hull in P. Recall from Observation 57 that Q is geodesically convex in P so the intersection of Q with a chord [geodesic] of P is a chord [geodesic] of Q.

We say that a subset $\mathcal{T}$ of the coarse cover of P is a ***coarse cover of Q in P*** if condition 3 of the definition of a coarse cover (Definition 37) holds for all points in the interior of Q, i.e., for any point x in the interior of Q and any edge e of P that is farthest from x, there is a triple (R, f, e) in the coarse cover with $x \in R$. (In particular, we get a coarse cover of Q by taking all the coarse cover elements whose triangles intersect the interior of Q.)

Recall that each triangle T of the coarse cover of P is bounded by a segment of an edge of P and two chords, and we store the endpoints of the chords on ∂P. As we recurse on subpolygons Q we will maintain the endpoints of these chords on ∂Q.

To obtain a linear-time algorithm for the geodesic edge center, we need a strengthening of the linear-time chord oracle (Lemma 38 in Section 6.2.6). The main idea is that our existing chord oracle always takes $O(n)$ time where n is the size of the simple polygon. The following modifications allow the chord oracle to run in time proportional to the size of the coarse cover of the chord under consideration. Roughly speaking, this improves the runtime of a logarithmic number of applications from $O(n \log n)$ to $O(n)$ (since the size of the coarse cover is reduced by at least a constant fraction between successive applications).

Lemma 63. *For the geodesic edge center problem, there is an algorithm that takes as input a simple 3-anchor hull Q known to contain the center of P in its interior, a coarse cover*

$\mathcal{T}$ of Q in P, and a chord K of Q, and decides whether the center lies left/right/on K. The runtime is $O(|\mathcal{T}|)$.

Proof. We first construct a coarse cover $\mathcal{T}_K$ of K. For each triple (T, f, e) in $\mathcal{T}$, let T_K be the intersection of triangle T with K. Note that T_K is a subsegment of K and can be found in constant time from the boundary chords of T. Add the triple (T_K, f, e) to $\mathcal{T}_K$. The resulting set $\mathcal{T}_K$ is a coarse cover of K of size at most $O(|\mathcal{T}|)$.

Next, we follow Steps 2 and 3 of the chord oracle algorithm—Step 2 finds the relative center on K, and Step 3 decides whether the center lies to the left or right (or on) K. As noted in Section 6.2.6, each step takes time $O(|\mathcal{T}_K|)$. □

Next we generalize Lemma 63 to a geodesic path.

Lemma 64. *For the geodesic edge center problem, there is an algorithm that takes as input a simple 3-anchor hull Q known to contain the center of P in its interior, a coarse cover $\mathcal{T}$ of Q in P, and a geodesic $\gamma = \pi(a, b)$ in Q with $a, b \in \partial Q$, and finds which subregion formed by γ contains the center, or if the center lies on the geodesic. The runtime is $O(|Q| + |\mathcal{T}|)$.*

The idea is similar to that of Lemma 63. We must first describe how to intersect the triangles of the coarse cover of Q with the geodesic γ. Since coarse cover triangles are bounded by chords, we can use the following result.

Lemma 65. *There is an algorithm that takes as input a subpolygon Q, a geodesic $\gamma = \pi(a, b)$ in Q with $a, b \in \partial Q$, and a set of chords $\mathcal{K}$ of Q, and finds the intersections of the chords of $\mathcal{K}$ with γ. Each chord of $\mathcal{K}$ is given by its endpoints together with the identity of the edge of Q containing the endpoint. The runtime is $O(|Q| + |\mathcal{K}|)$.*

Proof. Suppose the geodesic γ has g segments. Then it divides the boundary of Q into $g+1$ subchains, and we can traverse ∂Q once to identify, for each edge of Q, which subchain contains it.

Direct γ from a to b. This also directs the subchains of Q. Identify each segment $s = uv$ of γ with the subchain c_s that ends at v. The subchain c_s is unique except for the last segment incident to vertex b where two subchains end—use one of the two subchains and ignore the other one.

Observe that if chord $K \in \mathcal{K}$ crosses segment s, then K has an endpoint in c_s. Thus we can iterate through the chords K, finding which subchain contains each endpoint, and testing whether the associated segment of γ intersects K.

The total time is $O(|Q| + |\mathcal{K}|)$. □

With Lemma 65 in hand, we can prove Lemma 64.

Proof of Lemma 64. For each triangle of the coarse cover $\mathcal{T}$, the endpoints of its defining chords on ∂Q are known. Denoting the set of these defining chords by $\mathcal{K}$, we apply Lemma 65 to determine the intersections of the chords in $\mathcal{K}$ with γ. This takes $O(|Q|+|\mathcal{K}|)$ time, or equivalently, $O(|Q|+|\mathcal{T}|)$ time.

From the chord intersections, we can determine the intersections of the triangles of the coarse cover $\mathcal{T}$ with the segments of γ. Each segment s of γ is a chord of Q. Let $\mathcal{T}_s$ be the coarse cover elements whose triangles intersect the interior of s. Each intersection is an interval of s and the set of these intersections gives a coarse cover of s of size $O(|\mathcal{T}_s|)$. The chord oracle of Lemma 63 then determines whether the edge center lies left/right/on the segment s in time $O(|\mathcal{T}_s|)$. Running the algorithm for *all* the segments of γ will take $O(|\mathcal{T}|)$ time in total because each triangle of $\mathcal{T}$ intersects the interior of at most one segment of γ. If the center lies on one of the segments, then it lies on γ. Otherwise, since the segments partition Q into disjoint regions, knowing which side of each segment contains the center tells us the region that contains the center.

The algorithm takes $O(|Q|+|\mathcal{T}|)$ time. □

6.9 Phase II, Part 4: Finding the Geodesic Edge Center

In this section we give the final divide-and-conquer algorithm to find the edge center. Our approach was outlined in Section 6.5. We use the coarse cover $\mathcal{T}$ from Section 6.6, and we seek the point in P that minimizes the upper envelope of the functions of the coarse cover $\mathcal{T}$. We consider P as our initial 3-anchor hull defined by an anchor that is a boundary chain excluding one edge. At each step of the algorithm we find a smaller 3-anchor hull Q that contains the edge center, and we eliminate a constant fraction of the coarse cover. Each recursive step takes time linear in the size of Q plus the size of its coarse cover.

The algorithm has two stages. In the first stage (Section 6.9.1) no triangle of the coarse cover contains Q (this is true initially), so every triangle has a chord crossing Q and we can use the ϵ-net techniques of Section 6.7 to reduce the size of the subproblem. Once Q is contained in a triangle of the coarse cover, we show that Q must have constant size (Lemma 66)—in fact, we will exit stage 1 as soon as Q has constant size. It is then easy to reduce Q to a triangle. After that, we switch to the second stage (Section 6.9.2) of the

algorithm where we use a Megiddo-style prune-and-search technique to recursively reduce the size of the problem by a constant fraction.

6.9.1 Stage 1: Algorithm for Large Q

In this section, we explain our algorithm for large Q—the case where no triangle of the coarse cover can contain the 3-anchor hull Q. We begin by specifying the subproblems solved by the algorithm at each step. Each subproblem consists of the following.

1. Q, a simple 3-anchor hull of P that contains the geodesic edge center in its interior.
2. The set $\boldsymbol{\mathcal{T}(Q)}$ of all coarse cover elements whose triangles intersect the interior of Q. Each triangle of the coarse cover is specified by its two defining chords of P and the subsegment of an edge of P that forms its third side. Note: In this section we will refer to triangles of the coarse cover elements of $\mathcal{T}(Q)$ as "triangles of $\mathcal{T}(Q)$."
3. The set $\mathcal{K}_T(Q)$ of defining chords of triangles of $\mathcal{T}(Q)$, each given by its endpoints on the boundary of P, and each recording the one or two triangles of $\mathcal{T}(Q)$ that it is a side of. We also maintain the subset $\mathcal{K}(Q)$ of chords that cross Q, each given by its endpoints on the boundary of Q (as well as its endpoints on the boundary of P).

To solve a subproblem in Stage 1 means finding a point in Q that minimizes the upper envelope of the functions of the coarse cover $\mathcal{T}(Q)$, or reducing to $|Q| < 6$.

Define $\boldsymbol{t(Q)} := |\mathcal{T}(Q)|$. The ***size of a subproblem*** is $|Q| + t(Q)$, where $|Q|$ is the number of vertices of Q (as a polygon). Initially, Q is P, $\mathcal{T}(Q)$ is all of $\mathcal{T}$, and $\mathcal{K}_T(Q)$ and $\mathcal{K}(Q)$ are all of $\mathcal{K}$. The size of the initial problem is $O(n)$, because $\mathcal{T}$ has linear size by Lemma 54. Our goal is to spend linear time in the size of a subproblem to reduce the size by a constant fraction.

We need some results about the size of Q.

Lemma 66. *If a simple* 3*-anchor hull* Q *is contained in a triangle of the coarse cover then* $|Q| \leq 6$.

Proof. Let T be a triangle of $\mathcal{T}(Q)$ that contains Q. We claim that the boundary of Q has at most three edges that are subsegments of edges of P. Any such segment must lie on the boundary of T, and each of the three sides of T can contain at most one such segment by our assumption that no three vertices of P are collinear (Assumption 1).

We next claim that each of the at most three geodesic chains on the boundary of Q consists of a single segment. This is because an internal vertex v of a geodesic chain is a vertex of P, which must then be on the boundary of T (since no point on the boundary of P lies in the interior of T). But then the internal angle of Q at v is $\leq \pi$, so v is not an internal vertex of a geodesic path.

Thus Q has at most six edges. □

Claim 67. *If Q is a simple 3-anchor hull, then $|Q| \leq 3t(Q) + 6 \leq 9t(Q)$.*

Proof. Because Q is simple, every vertex v of Q has interior points of Q in its neighbourhood, so v must be contained in some triangle of $\mathcal{T}(Q)$ since $\mathcal{T}(Q)$ is a coarse cover of Q. By Observation 57, all but 6 of the vertices of Q are vertices of P.

Lemma 56 tells us that each triangle T of the coarse cover contains at most three vertices of P. This shows that $|Q| \leq 3t(Q) + 6$. For the second part of the inequality, note that $t(Q) \geq 1$. □

We note that the above claim depends on the assumption that Q is a 3-anchor hull of P. If we constructed 3-anchor hulls of 3-anchor hulls, then the number of vertices that are not vertices of P would grow.

We also need the following relationships between the number of chords and the number of coarse cover triangles.

Claim 68. *$|\mathcal{K}(Q)| \leq |\mathcal{K}_T(Q)| \leq 2t(Q)$. If Q is not contained in a triangle of $\mathcal{T}(Q)$, then $t(Q) \leq 2|\mathcal{K}(Q)|$.*

Proof. For the first inequality, $\mathcal{K}(Q) \subseteq \mathcal{K}_T(Q)$ and Observation 55 tells us that every triangle of the coarse cover has two chords (the third side is a piece of a polygon edge). For the second inequality, since no triangle of $\mathcal{T}(Q)$ contains Q, each one has at least one chord that crosses Q, and each chord comes from the coarse cover $\mathcal{T}(e)$ of a funnel $Y(e)$ and is a side of one or two coarse cover triangles in $\mathcal{T}(e)$. (If a chord arises from more more than one $Y(e)$, we duplicate it in $\mathcal{K}$, see the definition of $\mathcal{T}(e)$ in Section 6.6.) □

Next, we give an algorithm to handle a subproblem corresponding to a subpolygon Q (a simple 3-anchor hull) with $|Q| > 6$ and its associated sets $\mathcal{T}(Q)$, $\mathcal{K}_T(Q)$, and $\mathcal{K}(Q)$. By Lemma 66, no triangle of $\mathcal{T}(Q)$ contains Q, so every triangle of $\mathcal{T}(Q)$ has a chord in $\mathcal{K}(Q)$. The algorithm either finds the edge center or reduces to a subproblem with $|Q| \leq 6$ which is handled in Section 6.9.2. The idea was described in the main text.

1. For $\epsilon = \frac{1}{80}$, construct an ϵ-net N for the 3-anchor range space with ground set $\mathcal{K}(Q)$. The range space is defined with respect to 3-anchor hulls of P.
2. Compute the arrangement A of the chords N inside Q, and use the Chord Oracle of Lemma 63 to find the face F of A that contains the edge center.
3. Partition face F into a constant number of 3-anchor hulls of P.
4. Use the Geodesic Oracle (Lemma 64) to find which of these 3-anchor hulls contains the edge center, and to reduce it to a simple 3-anchor hull Q'.
5. If $|Q'| \leq 6$ then test each triangle of $\mathcal{T}(Q)$ to find $\mathcal{T}(Q')$ and $\mathcal{K}_T(Q')$, and switch to Stage 2 in the next subsection.
6. Otherwise $|Q'| > 6$. Find $\mathcal{K}(Q')$ and $\mathcal{T}(Q')$, and recurse on the subproblem for Q'.

We elaborate on these steps and their run-times below, but first we justify that our choice of ϵ in Step 2 guarantees that the size of the subproblem we recurse on is reduced by a fraction. Recall that the size of the subproblem for Q is $|Q| + t(Q)$.

Lemma 69. *For $\epsilon = \frac{1}{80}$, if $|Q'| > 6$, then $|Q'| + t(Q') \leq \frac{1}{2}(|Q| + t(Q))$.*

Proof. Since $|Q'| > 6$, no triangle of $\mathcal{T}(Q')$ contains Q'. Thus, since $\mathcal{K}(Q') \cap N = \emptyset$, the defining property of ϵ-nets (Equation 4.1), ensures that $|\mathcal{K}(Q')| \leq \frac{1}{80}|\mathcal{K}(Q)|$, which we relate to the subproblem sizes as follows.

$$
\begin{aligned}
|Q'| + t(Q') &\leq 9t(Q') + t(Q') && \text{by Claim 67} \\
&= 10t(Q') \leq 20|\mathcal{K}(Q')| && \text{by Claim 68 (no triangle of } \mathcal{T}(Q') \text{ contains } Q') \\
&\leq \tfrac{20}{80}|\mathcal{K}(Q)| = \tfrac{1}{4}|\mathcal{K}(Q)| && \text{by the } \epsilon\text{-net property} \\
&\leq \tfrac{1}{2}t(Q) && \text{by Claim 68} \\
&\leq \tfrac{1}{2}(|Q| + t(Q))
\end{aligned}
$$

□

We now fill in more details of the steps of the algorithm, and justify that the runtime is $O(|Q| + t(Q))$.

1. Construct an ϵ-net. Lemma 58 proves that the 3-anchor range space has bounded VC-dimension, and Lemma 61 proves that a subspace oracle exists. This implies (see

Lemma 8) that we can find a constant sized ϵ-net for this range space in time proportional to the size of the ground set, which is $O(|\mathcal{K}(Q)|)$ in our case. By Claim 68 this is $O(t(Q))$.

2. Compute the arrangement of A in Q and find the face F that contains the edge center. Once the constant sized ϵ-net N is determined, we can construct the arrangement of the chords in $O(|N|^2) = O(1)$ time, using the algorithm of Edelsbrunner et al [42]. Note that we know the endpoints of each chord of N on ∂Q. We run the chord oracle of Lemma 63 on each chord of N inside polygon Q to determine the face F that contains the edge center (halting if we find the center on one of the chords). This takes $O(t(Q))$ time for each chord of N. Since N has constant size, this step takes $O(t(Q))$ time.

3. Partition F into 3-anchor hulls. The boundary of F consists of $O(1)$ segments of chords in N, $O(1)$ subchains of the geodesics bounding Q, and $O(1)$ subchains of the polygon P. Let $V = \{v_0, \ldots, v_t\}$ be the points in order around ∂F that join successive segments/subchains. Then V has size $O(1)$. Find shortest paths $\gamma_i = \pi(v_0, v_i)$, $i = 1, \ldots, t$ in F. This takes time $O(|F|)$, which is $O(|Q|)$.

Let Γ be the set of these $O(1)$ shortest (geodesic) paths. Because F is geodesically convex, each shortest path $\gamma_i \in \Gamma$ is a geodesic path in P (the shortest path in P from v_0 to v_i lies inside F, and thus is equal to γ). We claim that the paths of Γ subdivide F into a constant number of 3-anchor hulls (which need not be simple). If the boundary of ∂F between v_i and v_{i+1}, $i = 1, \ldots, t-1$, is a segment of a chord of N or a subchain of a geodesic bounding Q, then take the 3-anchor hull that is the geodesic hull of the three point anchors v_0, v_i, v_{i+1}. If the boundary of ∂F between v_i and v_{i+1} is a subchain of ∂P, then take the 3-anchor hull that is the geodesic hull of v_0 and the polygon chain. Finally, if the boundary of ∂F between v_0 and v_1 or between v_t and v_0 is a subchain of ∂P, then take the 3-anchor hull of the polygon chain.

4. Find a simple 3-anchor hull $Q' \subseteq F$ that contains the edge center. Call the Geodesic Oracle (Lemma 64) in Q for each of the $O(1)$ geodesics of Γ. Halt if we find the center on one of the geodesics. Otherwise, the geodesic oracle tells us which region of the partition by Γ contains the edge center in its interior, and this gives us a ***simple*** 3-anchor hull Q' with the edge center in its interior. Each of the constant number of calls to the geodesic oracle takes time $O(|Q| + t(Q))$.

5. If $|Q'| \leq 6$, find $\mathcal{T}(Q')$ and $\mathcal{K}_T(Q')$. Since Q' has constant size, we can find its intersection with each triangle in $\mathcal{T}(Q)$ in constant time, so we can find $\mathcal{T}(Q')$ and $\mathcal{K}_T(Q')$ in time $O(t(Q))$.

6. If $|Q'| > 6$ find $\mathcal{K}(Q')$ and $\mathcal{T}(Q')$. We first find $\mathcal{K}(Q')$ by checking which chords of $\mathcal{K}(Q)$ cross Q'. By Observation 57, the 3-anchor hull Q' is bounded by at most three

polygon chains and three geodesic chains. A chord of $\mathcal{K}(Q)$ crosses Q' if and only if it has an endpoint interior to one of polygon chains of Q', or crosses one of the geodesic chains of Q'. We can test the former in constant time per chord because we know the endpoints of each chord on ∂P (including knowing which edge of P contains the endpoint). We can test the latter by finding the intersections of the chords of $\mathcal{K}(Q)$ with each of the at most three geodesics bounding Q' using Lemma 65 in Q. The runtime is $O(|Q| + |\mathcal{K}(Q)|) = O(|Q| + t(Q))$.

Note that these tests also determine the endpoints of each chord of $\mathcal{K}(Q')$ on $\partial Q'$.

Finally, since each chord of $\mathcal{K}(Q)$ records the triangles of $\mathcal{T}(Q)$ that it bounds, we set $\mathcal{T}(Q')$ to be the triangles that are bounded by a chord of $\mathcal{K}(Q')$. Note that this gives all triangles that intersect the interior of Q' since no triangle contains Q' by Lemma 66. This step takes $O(t(Q))$ time.

6.9.2 Stage 2: Algorithm for Q a Triangle

In this section we outline the algorithm to solve a subproblem for a subpolygon Q with $|Q| \leq 6$ and its associated sets $\mathcal{T}(Q)$ and $\mathcal{K}_T(Q)$. Some of the triangles of $\mathcal{T}(Q)$ may contain Q. We can triangulate Q in constant time and apply the chord oracle to determine which triangle contains the center. Thus we will assume that Q is a triangle.

We must find the point that minimizes the upper envelope of the functions of the coarse cover $\mathcal{T}(Q)$. We crucially use the properties that the upper envelope is a geodesically convex function (Corollary 14.1 from Chapter 5) and that Q is convex—together these imply that the upper envelope is a convex function. We use a Megiddo-style prune-and-search technique, following the same approach as Ahn et al. [4, Section 7] but modified to deal with the edge center rather than the vertex center.

Each triangle T of the coarse cover is the domain of a distance function to some edge e of P. Definition 37 tells us that functions associated with coarse cover elements have two different forms. Accordingly, we partition $\mathcal{T}(Q)$ into:

1. $\mathcal{T}_1$: Coarse cover elements whose associated functions have the form $d_2(x, v) + \kappa$, where v is a vertex and κ is a constant.
2. $\mathcal{T}_2$: Coarse cover elements whose associated functions have the form $d_2(x, \bar{e})$, where $\bar{e}$ is the line through edge e.

To determine the edge center, we must locate a point $x = (x_1, x_2)$ and a value ρ to solve the following optimization problem:

$$\begin{array}{lll} \text{minimize} & \rho & \\ \text{subject to} & x \in Q & \\ & d_2(x, v) + \kappa \leq \rho & x \in T \cap Q;\ v,\ \kappa, \text{ and } T \text{ from an element of } \mathcal{T}_1 \\ & d_2(x, \bar{e}) \leq \rho & x \in T \cap Q;\ \bar{e} \text{ and } T \text{ from an element of } \mathcal{T}_2 \end{array} \tag{6.2}$$

We show how to solve the optimization problem (6.2) in linear time when the upper envelope of the coarse cover functions is convex. (Without this condition our methods do not apply since we then have unrelated convex constraints defined on different subdomains T.)

The constraints corresponding to $\mathcal{T}_1$ will be referred to as *disk constraints*. The constraints corresponding to $\mathcal{T}_2$ will be referred to as *half-plane constraints*. Ahn et al. [4, Section 7] solve Problem (6.2) when there are no half-plane constraints. Following their approach, we first describe previous work that handles the case when all triangles of the coarse cover contain Q.

Special Case: All Triangles Contain Q. Note that in this case there is no need to assume that the upper envelope of the coarse cover functions is convex, since this follows immediately from the fact that each constraint is convex on Q. A linear-time algorithm for this case is given in the solution of the visibility center problem in Chapter 5. The idea is to pair up the half-plane constraints and separately pair up the disk constraints. After computing the bisector of each pair, the prune-and-search approach described above will prune away a constant fraction of both types of constraints in linear time.

General Case. The new complication is that each constraint applies only in a triangular subdomain. The idea for the solution one dimension down (with interval subdomains on a line) comes from the linear-time chord oracle of Pollack et al. [91]. This was extended by Ahn et al. [4] to two dimensions. They dealt only with disk constraints, but we can extend the approach to handle both disk constraints and half-plane constraints, by pairing each constraint with another of the same type.

Relevant background is provided in Section 4.2. We outline the approach of Ahn et al. [4, Section 7.1]. The basic idea is to add the subdomain boundary lines to the set of bisectors. Each triangle of the coarse cover is bounded by two chords of P. A pair of constraints (of the same type) then involves five linear constraints (a "plane-set"): two for

each triangular subdomain plus one bisecting plane. Using cuttings and a "side-decision" algorithm (the hyperplane oracles from Section 4.2) we can in linear time restrict our search to a constant sized convex region $Q' \subseteq Q$ such that some constant fraction of the pairs of constraints have the property that no member of their plane-set intersects Q'. The claim is that at least one of each such pair can be eliminated. If Q' is outside either of the two triangular domains, then the corresponding constraint is irrelevant Otherwise, Q' is inside both the domains. In this case, we use the fact that it lies on one side of the bisector plane. One constraint dominates over the other on this side of the bisector plane, and the other one may be ignored. The last remaining ingredient is the hyperplane oracle which involves solving Problem (6.2) restricted to a plane—this is the same problem down a dimension—and then testing whether this solution is a local (hence global) solution and if not, finding which side of the plane contains the optimum.

This completes the outline for solving Problem 6.2 in linear-time.

6.10 The Farthest Edge Voronoi Diagram of a Polygon

In the previous sections we gave a linear time algorithm to find the geodesic edge center of a simple polygon. The center is either a vertex of the geodesic farthest edge Voronoi Diagram, $\mathcal{V}(P)$, or lies on one of its edges. This section considers applications to the construction of the diagram $\mathcal{V}(P)$. We continue to make general position assumptions from Section 6.2.2 to ensure that the diagram is 'nice'. For instance, these assumptions ensure that there are no two-dimensional bisectors of the type shown in Figure 6.2).

Convex Polygons. The farthest-vertex Voronoi diagram for a convex polygon can be constructed in linear time, using an algorithm of Aggarwal et al. [2]. It is an open question to construct the farthest-edge diagram $\mathcal{V}(P)$ in *deterministic* linear time, even for the special case of a convex polygon. Aurenhammer et al. [13] explicitly ask whether the farthest-edge Voronoi diagram for a convex polygon can be constructed in faster than $O(n \log n)$ time. The algorithm in Aggarwal et al. [2] does not easily adapt to the farthest edge case. This was observed by Khramtcova and Papadopoulou [62], who give an $O(n)$ expected time algorithm for constructing a farthest segment Voronoi diagram in the Euclidean plane, once the sequence of faces at infinity are known. For a *convex* polygon, this sequence coincides with the order of the edges.

Now, suppose that we wish to determine the farthest edge Voronoi diagram restricted to a convex polygon P (instead of the whole plane). Using Theorem 42, we can determine

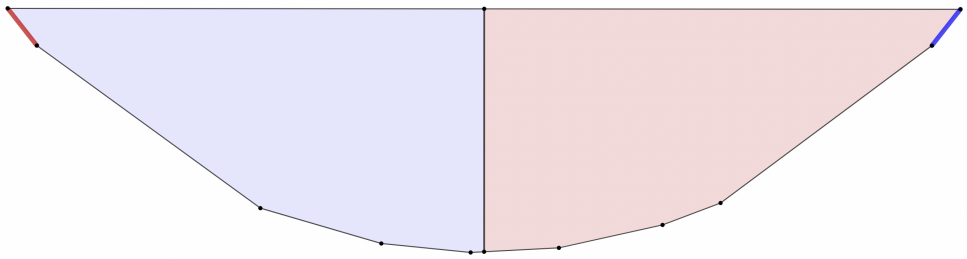


Figure 6.25: An extreme example where the complexity of the farthest edge Voronoi diagram *within* a convex polygon is much lower than the complexity of the unbounded farthest edge Voronoi diagram for all the edges which is $O(n)$. Only the edges colored red and blue have Voronoi regions within the polygon (the Voronoi region of the edge is shaded with the same color as the edge).

$\mathcal{V}(\partial P)$, i.e. $\mathcal{V}(P)$ restricted to ∂P in $O(n)$ time. Let H be the set of edges with non-empty Voronoi regions in $\mathcal{V}(\partial P)$. These are the only edges with non-empty Voronoi regions inside P. We can now use H to construct $\mathcal{V}(P)$ in $O(|H|)$ expected time using the algorithm of Khramtcova and Papadopoulou [62], . The size of the set H can be much smaller than n, as seen in Figure 6.25. In fact, if $|H|$ is in $O(\frac{n}{\log n})$, we can construct $\mathcal{V}(P)$ in deterministic linear time using the algorithm from Aurenhammer et al. [13]. We are not aware of any other work that can determine the set H in deterministic linear time.

Simple Polygons with the Geodesic Metric. The geodesic nearest edge Voronoi diagram is simply the medial axis of the polygon, a well-studied structure with numerous applications. The medial axis can be constructed in linear time using an algorithm of Chin et al. [29].

When the sites are not points, we are not aware of any work on geodesic farthest site Voronoi diagrams (although this is a well-studied notion in the Euclidean metric [14, 28]). The geodesic farthest edge Voronoi diagram $\mathcal{V}(P)$ is an extension of the well-studied geodesic farthest vertex Voronoi diagram [10, 102, 87]. To see this, observe that if we consider the vertices to be edges of infinitesimal length then the diagram of this augmented edge set coincides with the geodesic farthest vertex Voronoi diagram. This is a consequence of the following fact: from any point in the polygon, one of the endpoints of any of the original edges must be geodesically farther than the edge itself.

Almost all the farthest site Voronoi diagrams construct the diagram at infinity (or on the polygon boundary) and then construct the interior using a *sweep* algorithm. Constructing the diagram on the boundary is usually the more involved part of the algorithm.

Theorem 42 should find applicability in extending the algorithms that construct geodesic farthest vertex Voronoi diagram to obtain efficient algorithms for constructing the diagram for farthest edges.

6.11 Conclusions and Open Problems

In this chapter, we described a linear-time algorithm for finding the geodesic edge center of the edges of a simple polygon. This generalizes the well-studied geodesic center and (coupled with the results from Chapter 5) is the first work on the geodesic center of non-point sites in polygons.

Open Problems. We pose the following open problems:

1. Find an efficient algorithm to construct the geodesic farthest edge Voronoi diagram of a simple polygon. There are efficient algorithms for determining the geodesic farthest Voronoi diagrams for point sites ([87, 15, 86]), but we found virtually no literature on efficient constructions for non-point sites.

2. Find the geodesic center of arbitrary polygonal shapes inside a polygon. Note that in Chapter 5, we determined the geodesic center of a set of visibility polygons. But the natural problem of finding the center of arbitrary polygons, although well studied under the Euclidean metric, has not been addressed in the geodesic setting.

Chapter 7

Conclusions and Future Directions

> ...for the greatest fool may ask more than the wisest man can answer.
>
> Charles Caleb Colton

In this final chapter, we summarize the results of this thesis and propose directions for future research.

The contributions of this work are:

1. We proposed a simple geometric transformation to solve the dispersion problem on intervals in linear-time. Our solution guarantees not just max-min dispersion but lexicographic dispersion—a strictly stronger property.

2. We introduced the notion of the visibility center of a simple polygon—a point in the polygon starting from which the maximum distance to visibility to any of k given points in the polygon is minimized. Chapter 5 described an $O((n+k)\log(n+k))$ time algorithm to determine the visibility center of k point sites in an n vertex polygon. Our solution idea involves determining the geodesic center of half-polygons inside a polygon.

3. We also described an optimal linear time algorithm to determine the geodesic edge center of a simple polygon (a special case of the geodesic center for half-polygons). This is the very natural problem of determining the point in a simple polygon that minimizes the distance to the farthest edge.

7.1 Directions for Future Research

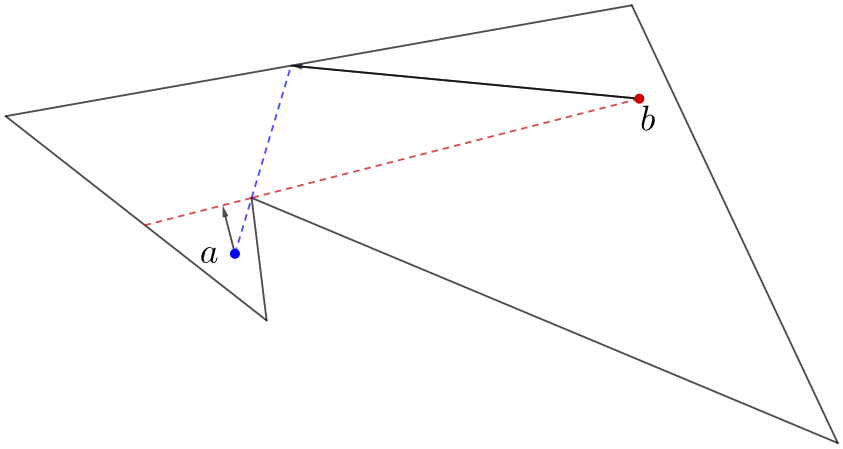


Figure 7.1: The distances to visibility $d_V(a, b)$ and $d_V(b, a)$ are not equal in general.

Inverse Visibility Center. The distance to visibility is not symmetric: $d_V(a, b) \neq d_V(b, a)$. See Figure 7.1. This motivates the question of locating a point in a simple polygon that is "easy-to-view" from any starting point. In other words, we wish to find a point such that the maximum distance to view it (from any starting point in the polygon) is minimized. This problem seems to be of a slightly different flavour from the problems of locating the visibility center and the edge center.

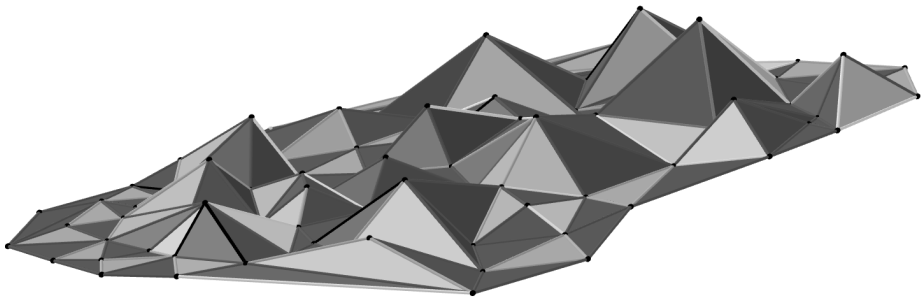


Figure 7.2: A polyhedral terrain

Center Problems on Polyhedral Terrains. Another interesting problem to consider is an algorithm for the visibility center of viewpoints on a polyhedral terrain (see Figure 7.2).

Literature on center problems on polyhedral terrains is very sparse—we are only aware of the paper by Aronov et al. [11]. There is also prior work on terrain visibility that addresses the construction of structures similar to the visibility polygon for a simple polygon (see Cole and Sharir [31], Hurtado et al. [57]). In order to see a viewpoint starting from some point on the terrain, the motion we could allow could be along shortest paths for air travel (drones), land travel (cars), or a combination of the two. An understanding of such questions, although more challenging, seem very applicable to facility location problems in real-life situations.

Geodesic Segment Centers. There is prior work on finding a line in the plane that minimizes the maximum Euclidean distance to a set of points or minimizes the average Euclidean distance to the line (See the papers by Morris and Norbach [82], Lee and Wu [64]). Finally, Imai et al. [59] solve the following problem: given a set of points in the plane, find a placement for a given line segment so as to minimize the distance to the farthest input point. Efrat and Sharir [43] gave a near-linear time algorithm for this problem. It is very interesting to consider the geodesic variants of these problems. The problem is the following: find a chord (or segment of given length) in a simple polygon that minimizes the maximum (or average) geodesic distance to an input set of points in the polygon. There can be variants based on allowing unweighted or weighted distances.

References

[1] Mikkel Abrahamsen and Bartosz Walczak. Common tangents of two disjoint polygons in linear time and constant workspace. *ACM Transactions on Algorithms (TALG)*, 15(1):1–21, 2018. doi:10.1145/3284355.

[2] Alok Aggarwal, Leonidas J Guibas, James Saxe, and Peter W Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989. doi:10.1007/bf02187749.

[3] Alok Aggarwal, Maria M Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1-4):195–208, 1987. doi:10.1007/bf01840359.

[4] Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56(4):836–859, 2016. doi:10.1007/s00454-016-9796-0.

[5] Oswin Aichholzer, Wolfgang Aigner, Franz Aurenhammer, Thomas Hackl, Bert Jüttler, Elisabeth Pilgerstorfer, and Margot Rabl. Divide-and-conquer for Voronoi diagrams revisited. *Computational Geometry*, 43(8):688–699, 2010. doi:10.1016/j.comgeo.2010.04.004.

[6] Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D. Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, aug 2017. doi:10.1007/s00454-017-9918-3.

[7] Tetsuya Araki, Hiroyuki Miyata, and Shin-ichi Nakano. Dispersion on intervals. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, page 2021DMP0004, 2022. doi:10.1587/transfun.2021dmp0004.

[8] Tetsuya Araki and Shin-ichi Nakano. Max–min dispersion on a line. *Journal of Combinatorial Optimization*, pages 1–7, 2020. doi:10.1007/s10878-020-00549-5.

[9] Esther M Arkin, Alon Efrat, Christian Knauer, Joseph S B Mitchell, Valentin Polishchuk, Günter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7:77–100, 2016. doi:10.20382/jocg.v7i2a5.

[10] Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, 1993. doi:10.1007/bf02189321.

[11] Boris Aronov, Marc van Kreveld, René van Oostrum, and Kasturirangan Varadarajan. Facility location on terrains. In *Algorithms and Computation*, pages 20–29. Springer Berlin Heidelberg, 1998. doi:10.1007/3-540-49381-6_4.

[12] Tetsuo Asano and Godfried Toussaint. Computing the geodesic center of a simple polygon. In *Discrete Algorithms and Complexity*, pages 65–79. Elsevier, 1987. doi:10.1016/b978-0-12-386870-1.50010-1.

[13] Franz Aurenhammer, Robert L Scot Drysdale, and Hannes Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006. doi:10.1016/j.ipl.2006.07.008.

[14] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing Company, 2013. doi:10.1142/8685.

[15] Luis Barba. Optimal algorithm for geodesic farthest-point Voronoi diagrams. In *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.12.

[16] Christoph Baur and Sándor P Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001. doi:10.1007/s00453-001-0022-x.

[17] Binay K Bhattacharya, Shreesh Jadhav, Asish Mukhopadhyay, and J-M Robert. Optimal algorithms for some intersection radius problems. *Computing*, 52(3):269–279, 1994. doi:10.1007/bf02246508.

[18] Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Dispersion for intervals: A geometric approach. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 37–44. SIAM, 2021. doi:10.1137/1.9781611976496.4.

[19] Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Distant Representatives for Rectangles in the Plane. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.17.

[20] Sergio Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007. doi:10.1016/j.jalgor.2004.06.009.

[21] Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. An improved analysis of local search for max-sum diversification. *Mathematics of Operations Research*, 44(4):1494–1509, 2019. doi:10.1287/moor.2018.0982.

[22] Erin Chambers, Alejandro Erickson, Sándor P. Fekete, Jonathan Lenchner, Jeff Sember, Venkatesh Srinivasan, Ulrike Stege, Svetlana Stolpner, Christophe Weibel, and Sue Whitesides. Connectivity graphs of uncertainty regions. *Algorithmica*, 78(3):990–1019, 2017. doi:10.1007/s00453-016-0191-2.

[23] Barun Chandra and Magnús M Halldórsson. Approximation algorithms for dispersion problems. *Journal of algorithms*, 38(2):438–465, 2001. doi:10.1006/jagm.2000.1145.

[24] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991. doi:10.1007/bf02574703.

[25] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993. doi:10.1007/bf02189314.

[26] Bernard Chazelle and Leonidas J Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4(6):551–581, 1989. doi:10.1007/bf02187747.

[27] Bernard Chazelle and Jiri Matousek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996. doi:10.1006/jagm.1996.0060.

[28] Otfried Cheong, Hazel Everett, Marc Glisse, Joachim Gudmundsson, Samuel Hornus, Sylvain Lazard, Mira Lee, and Hyeon-Suk Na. Farthest-polygon Voronoi diagrams. *Computational Geometry*, 44(4):234–247, 2011. doi:10.1007/978-3-540-75520-3_37.

[29] Francis Chin, Jack Snoeyink, and Cao An Wang. Finding the medial axis of a simple polygon in linear time. *Discrete & Computational Geometry*, 21(3):405–420, 1999. doi:10.1007/pl00009429.

[30] Wei-Pang Chin and Simeon Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991. doi:10.1007/bf02574671.

[31] Richard Cole and Micha Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 7(1):11–30, jan 1989. doi:10.1016/s0747-7171(89)80003-3.

[32] Hristo N Djidjev, Andrzej Lingas, and Jörg-Rüdiger Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8(2):131–152, 1992. doi:10.1007/bf02293040.

[33] Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum-and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, 56(1):220–250, 2015. doi:10.1007/s00224-014-9591-3.

[34] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S B Mitchell. Touring a sequence of polygons. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 473–482, 2003. doi:10.1145/780542.780612.

[35] R L Scot Drysdale and Asish Mukhopadhyay. An $O(n \log n)$ algorithm for the all-farthest-segments problem for a planar set of points. *Information Processing Letters*, 105(2):47–51, 2008. doi:10.1016/j.ipl.2007.08.004.

[36] Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory of Computing Systems*, 51(2):125–142, 2012. doi:10.1007/s00224-011-9331-x.

[37] Adrian Dumitrescu and Minghui Jiang. Systems of distant representatives in Euclidean space. *Journal of Combinatorial Theory, Series A*, 134:36–50, 2015. doi:10.1016/j.jcta.2015.03.006.

[38] Martin Dyer, Bernd Gärtner, Nimrod Megiddo, and Emo Welzl. Linear programming. In Csaba D. Tóth Jacob E. Goodman, Joseph O'Rourke, editor, *Handbook of Discrete and Computational Geometry*, pages 1291–1309. Chapman and Hall/CRC, 2017. doi:10.1201/9781420035315.pt6.

[39] Martin E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45, 1984. doi:10.1137/0213003.

[40] Martin E Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986. doi:10.1137/0215052.

[41] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990. doi:10.1145/77635.77639.

[42] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986. doi:10.1137/0215024.

[43] A. Efrat and M. Sharir. A near-linear algorithm for the planar segment-center problem. *Discrete & Computational Geometry*, 16(3):239–257, 1996. doi:10.1007/bf02711511.

[44] Sándor P. Fekete and Henk Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38(3):501–511, 2004. doi:10.1007/s00453-003-1074-x.

[45] Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005. doi:10.1016/j.dam.2004.02.018.

[46] Michael R. Garey, David S. Johnson, Barbara B. Simons, and Robert Endre Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981. doi:10.1137/0210018.

[47] Subir Kumar Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007. doi:10.1017/cbo9780511543340.

[48] Subir Kumar Ghosh and David M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991. doi:10.1137/0220055.

[49] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987. doi:10.1007/bf01840360.

[50] Leonidas J Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-x.

[51] Marshall Hall Jr. Distinct representatives of subsets. *Bulletin of the American Mathematical Society*, 54(10):922–926, 1948. URL: https://projecteuclid.org/euclid.bams/1183512379.

[52] Sariel Har-Peled. *Geometric Approximation Algorithms*. Number 173. American Mathematical Soc., 2011. doi:10.1090/surv/173.

[53] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, may 1984. doi:10.1137/0213024.

[54] David Haussler and Emo Welzl. ϵ-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987. doi:10.1007/bf02187876.

[55] John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.

[56] John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997. doi:10.1137/s0097539793253577.

[57] Ferran Hurtado, Maarten Löffler, Inês Matos, Vera Sacristán, Maria Saumell, Rodrigo I Silveira, and Frank Staals. Terrain visibility with multiple viewpoints. *International Journal of Computational Geometry & Applications*, 24(04):275–306, 2014. doi:10.1142/s0218195914600085.

[58] Chang Yong Han Hyeong In Choi. The medial axis transform. In Gerald Farin, Josef Hoschek, and M-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. Elsevier, 2002. doi:10.1016/b978-044451104-1/50020-4.

[59] Hiroshi Imai, D. T. Lee, and Chung-Do Yang. 1-segment center problems. *ORSA Journal on Computing*, 4(4):426–434, 1992. doi:10.1287/ijoc.4.4.426.

[60] Shreesh Jadhav, Asish Mukhopadhyay, and Binay Bhattacharya. An optimal algorithm for the intersection radius of a set of convex polygons. *Journal of Algorithms*, 20(2):244–267, 1996. doi:10.1006/jagm.1996.0013.

[61] Yan Ke and Joseph O'Rourke. Computing the kernel of a point set in a polygon. In *Workshop on Algorithms and Data Structures (WADS '89)*, pages 135–146. Springer, 1989. doi:10.1007/3-540-51542-9_12.

[62] Elena Khramtcova and Evanthia Papadopoulou. An expected linear-time algorithm for the farthest-segment Voronoi diagram. arXiv, 2014. doi:10.48550/arxiv.1411.2816.

[63] Clark Kimberling. Encyclopedia of Triangle Centers. https://faculty.evansville.edu/ck6/encyclopedia/ETC.html. Accessed: 2022-06-06.

[64] D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1(1-4):193–211, nov 1986. doi:10.1007/bf01840442.

[65] Der-Tsai Lee and Franco P Preparata. An optimal algorithm for finding the kernel of a polygon. *Journal of the ACM (JACM)*, 26(3):415–421, 1979. doi:10.1145/322139.322142.

[66] Der-Tsai Lee and Franco P Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984. doi:10.1002/net.3230140304.

[67] D.T Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22(2):207–221, may 1983. doi:10.1016/0734-189x(83)90065-8.

[68] Shimin Li and Haitao Wang. Dispersing points on intervals. *Discrete Applied Mathematics*, 239:106–118, 2018. doi:10.1016/j.dam.2017.12.028.

[69] Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235, 2010. doi:10.1007/s00453-008-9174-2.

[70] Maarten Löffler and Marc van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419–433, 2010. doi:10.1016/j.comgeo.2009.03.007.

[71] Anna Lubiw and Anurag Murty Naredla. The Visibility Center of a Simple Polygon. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 65:1–65:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.65.

[72] Anna Lubiw and Anurag Murty Naredla. The visibility center of a simple polygon. *arXiv preprint arXiv:2108.07366*, 2021. URL: https://arxiv.org/abs/2108.07366.

[73] Hanan Luss. On equitable resource allocation problems: A lexicographic minimax approach. *Operations Research*, 47(3):361–378, 1999. doi:10.1287/opre.47.3.361.

[74] Jiří Matoušek. Construction of epsilon nets. In *Proceedings of the Fifth Annual Symposium on Computational Geometry (SoCG '89)*, pages 1–10, 1989. doi:10.1145/73833.73834.

[75] Jiří Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6(3):385–406, 1991. doi:10.1007/bf02574697.

[76] Jiří Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50(2):203–208, 1995. doi:10.1006/jcss.1995.1018.

[77] Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer Verlag, 2002. doi:10.1007/978-1-4613-0039-7.

[78] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983. doi:10.1137/0212052.

[79] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

[80] Nimrod Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(6):605–610, 1989. doi:10.1007/bf02187750.

[81] Nimrod Megiddo and Kenneth J Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984. doi:10.1137/0213014.

[82] James G. Morris and John P. Norback. Linear facility location — solving extensions of the basic problem. *European Journal of Operational Research*, 12(1):90–94, jan 1983. doi:10.1016/0377-2217(83)90183-2.

[83] Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1994.

[84] Nabil H Mustafa. *Sampling in Combinatorial and Geometric Set Systems*, volume 265 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2022. doi:10.1090/surv/265.

[85] Włodzimierz Ogryczak. On the lexicographic minimax approach to location problems. *European Journal of Operational Research*, 100(3):566–585, 1997. doi:10.1016/S0377-2217(96)00154-3.

[86] Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, 63(2):418–454, 2020. doi:10.1007/s00454-019-00063-4.

[87] Eunjin Oh, Luis Barba, and Hee-Kap Ahn. The geodesic farthest-point Voronoi diagram in a simple polygon. *Algorithmica*, 82(5):1434–1473, 2020. doi:10.1007/s00453-019-00651-z.

[88] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 57. Oxford New York, NY, USA, 1987.

[89] Christos H Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM Journal on Computing*, 10(3):542–557, 1981. doi:10.1137/0210040.

[90] Evanthia Papadopoulou and Sandeep Kumar Dey. On the farthest line-segment Voronoi diagram. *International Journal of Computational Geometry & Applications*, 23(06):443–459, 2013. doi:10.1007/978-3-642-35261-4_22.

[91] Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989. doi:10.1007/bf02187751.

[92] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (FOCS 1975)*, pages 151–162. IEEE, 1975. doi:10.1109/sfcs.1975.8.

[93] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS '92)*, pages 567–579. Springer, 1992. doi:10.1007/3-540-55210-3_213.

[94] Robin Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1978. doi:10.1093/comjnl/21.3.243.

[95] Barbara Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science*, pages 246–252. IEEE, 1978. doi:10.1109/SFCS.1978.4.

[96] Barbara Simons and Michael Sipser. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research*, 32(1):80–88, 1984. doi:10.1287/opre.32.1.80.

[97] Subhash Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39:220–235, 1989. doi:10.1016/0022-0000(89)90045-7.

[98] James Joseph Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1(1):79–80, 1857. doi:10.1093/oso/9780198869030.003.0005.

[99] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman, editors. *Handbook of Discrete and Computational Geometry*. CRC press, 2017. doi:10.1201/9781315119601.

[100] Godfried T Toussaint. Computing geodesic properties inside a simple polygon. *Revue d'Intelligence Artificielle*, 3(2):9–42, 1989. URL: https://pascal-francis.inist.fr/vibad/index.php?action=getRecordDetail&idt=6661941.

[101] Haitao Wang. Quickest visibility queries in polygonal domains. *Discrete & Computational Geometry*, 62(2):374–432, 2019. doi:10.1007/s00454-019-00108-8.

[102] Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point Voronoi diagrams in simple polygons. *Discrete & Computational Geometry*, 2022. doi:10.1007/s00454-022-00424-6.