On the Hausdorff and Other Cluster Voronoi Diagrams

Doctoral Dissertation submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Elena Khramtcova

under the supervision of
Prof. Evanthia Papadopoulou

Dissertation Committee

Prof. Antonio Carzaniga	Università della Svizzera Italiana, Switzerland
Prof. Kai Hormann	Università della Svizzera Italiana, Switzerland
Prof. Franz Aurenhammer	Technische Universität, Graz, Austria
Prof. Stefan Langerman	Université Libre de Bruxelles, Belgium

Dissertation accepted on

Prof. Evanthia Papadopoulou
Research Advisor
Università della Svizzera Italiana, Switzerland

Prof. Walter Binder
PhD Program Director

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Elena Khramtcova
Lugano,

To Zhenja, Vitja, and to my mom

Abstract

The Voronoi diagram is a fundamental geometric structure that encodes proximity information. Given a set of geometric objects, called sites, their Voronoi diagram is a subdivision of the underlying space into maximal regions, such that all points within one region have the same nearest site. Problems in diverse application domains (such as VLSI CAD, robotics, facility location, etc.) demand various generalizations of this simple concept. While many generalized Voronoi diagrams have been well studied, many others still have unsettled questions. An example of the latter are cluster Voronoi diagrams, whose sites are sets (clusters) of objects rather than individual objects.

In this dissertation we study certain cluster Voronoi diagrams from the perspective of their construction algorithms and algorithmic applications. Our main focus is the *Hausdorff Voronoi diagram*; we also study the *farthest-segment* Voronoi diagram, as well as certain special cases of the *farthest-color* Voronoi diagram. We establish a connection between cluster Voronoi diagrams and the *stabbing circle* problem for segments in the plane. Our results are as follows.

(1) We investigate the randomized incremental construction of the Hausdorff Voronoi diagram. We consider separately the case of *non-crossing* clusters, when the combinatorial complexity of the diagram is $O(n)$ where n is the total number of points in all clusters. For this case, we present two construction algorithms that require $O(n\log^2 n)$ expected time. For the general case of arbitrary clusters, we present an algorithm that requires $O((m+n\log n)\log n)$ expected time and $O(m+n\log n)$ expected space, where m is a parameter reflecting the number of crossings between clusters' convex hulls.

(2) We present an $O(n)$ time algorithm to construct the farthest-segment Voronoi diagram of n segments, after the sequence of its faces at infinity is known. This augments the well-known linear-time framework for Voronoi diagram of points in convex position, with the ability to handle disconnected Voronoi regions.

(3) We establish a connection between the cluster Voronoi diagrams (the Hausdorff and the farthest-color Voronoi diagram) and the stabbing circle problem. This implies a new method to solve the latter problem. Our method results in a near-

optimal $O(n\log^2 n)$ time algorithm for a set of n parallel segments, and in an optimal $O(n\log n)$ time algorithm for a set of n segments satisfying some other special conditions.

(4) We study the farthest-color Voronoi diagram in special cases considered by the stabbing circle problem. We prove $O(n)$ bound for its combinatorial complexity and present an $O(n\log n)$ time algorithm to construct it.

Acknowledgements

Somebody said, that to increase the level of happiness around you, whenever you want to say "sorry", you should better transform your "sorry" to a "thank you". Here is my list of "thank you"s, both original and transformed, related to my PhD studies, and, what is hard to separate, the last quarter of my conscious life.

Of course, the central person is the supevisor, and I deeply thankful to Evanthia for every single discussion, every piece of advise, and for being always honest, patient, and supportive.

I want to thank the colleagues who contributed to some parts of this dissertation: Panagiotis Cheilaris, Merce Claverol, Stefan Langeman, Maria Saumell, and Carlos Seara.

I want to thank the members of my PhD committee, for having time to read through this dissertation, for being collaborative in the organizational matters, and for attending the defence presentation in person.

I want to thank Kira Vyatkina, who wisely suggested me to go to Lugano, as opposed to trying to do my PhD with her at SPbSU.

As the dissertation is only an outcome, but not the whole game, I have much more "thanks" to say. I thank:

- my mom and my kids Zhenja and Vitja for their infinite love, acceptance and care, without conditions.
- my psychoanalist/guardian angel Marina Volokhonskaya for showing me my ears. I would be a different person without her.
- Sasha Khramtcov for being a great dad for our children, and for living in Lugano, which must have been not easy. The members of Sasha's family for accepting and supporting my decisions, even though these decisions contradict their values and affect their lives.
- the members of our research group: Panagiotis Cheilaris, Sandeep Dey, Maksym Zavershinskiy, and Kolja Junginger. Sandeep, I'll try to never forget your lessons on detachment. Kolja, thanks for the many discussions, proofreading, and especially for the reading Fridays.

- the members of the Faculty of Informatics at USI, in particular, Antonio Carzaniga and Benedetto Lepori. Your lectures and discussions provided me with a point of view on the conemporary academic world as a social structure. This contributed to my decision to try to stay in academy despite all "buts" that life threw into me.
- Elisa Larghi, Nina Caggiano and others from Decanato, you are the best!
- the great research community I'm now happy to be a part of. In particular: Gill Barequet, Jit Bose, Jean Cardinal, Jean-Lou De Carufel, Travis Gagie, John Iacono, Irina Kostitsina, Stefan Langerman, Maarten Löffler, Maria Saumell, Mira Shalah, Carlos Seara, Sander Vendershot. I would like to thank the organizers of the Barbados and Mexico workshops in 2015 for inviting me, so that I got exposed to the great inspiring atmosphere and met many people. I thank ESF/EuroGIGA and SNF for supporting my research to the extent that often, when considering research visits/attending conferences or workshops, the issue was time, not money.
- the tango community of Ticino (Antonella and Martin from the "New Style Dance" school, Edmondo, Catherina and Andrea, and many others) for introducing me to the beauty, diversity, and depth of ways of communication. And to Stefan, for encouraging me to try it in the first place.
- the team of Excalibur training center (esp., Nazareno and Alberto), and the running section SAM Massagno (Soccorso and Antonella from there). You taught me that each marathon is nothing more and nothing less than a (finite) number of plainly simple steps in the right direction.
- the wonderful people of Lugano, who would meet me anywhere around, smile and say "Buongiorno!", no matter who I am.
- The final item, but extremely important for me: my friends who were "so close no matter how far", in different periods: Andrey Breslav, Stas Osipov, Nikita Alexeev, Masha Lebedeva, Polina Ivanova, Igor Evgenievich Kanunnikov, Kira Vyatkina, Tanja Shamaeva, Tasha Molchanova, Artiom Kovnatsky, Dima Anisimov, Parisa Marandi, Sveta Didenko, Sandeep Dey, Luba Smirnova, Masha Kogan, Ivan Egorov.

You all were, in many occasions, believing in me more than I was believing in myself. I think this is among a very few true miracles one can ever experience, and thus you all are magicians. Thank you!

Lena Khramtcova,
Lugano, summer 2016.

Contents

List of Figures

Chapter 1

Introduction

The research field of *computational geometry* is a branch in the area of *theory of algorithms* that deals with algorithmic problems in discrete geometry. In particular, the mission of computational geometry is to design efficient algorithms involving discrete geometric structures, and to analyze the performance (time and space complexity) of such algorithms. Computational geometry is often motivated by problems in application domains that require algorithmic tools to manipulate geometric objects. Such application domains include VLSI computer-aided design, motion planning in robotics, geographical information systems, and many other areas.

A typical research question in computational geometry is as follows. Given a finite set of simple geometric objects, provide an algorithm to efficiently construct a certain geometric structure that is induced by this set. One of the most celebrated geometric structures is the *Voronoi diagram*. It is a subdivision of a given space into maximal regions that reveal proximity information for an input set of simple geometric objects, called *sites*. Such proximity information is highly demanded in diverse areas, both theoretical and applied. Hence, Voronoi diagrams have numerous applications.

The classic Voronoi diagram is the *nearest-neighbor Voronoi diagram* of point sites, which is defined as follows. Given a set of points in some space, its *nearest-neighbor Voronoi diagram* is a partitioning of the underlying space into maximal regions such that all points in one region have the same nearest site, see Definition 1 and Figure 1.1. A symmetric concept, which has been well studied too, is the *farthest-site*, or simply the *farthest Voronoi diagram*. This diagram reveals the farthest site, as opposed to the nearest one, see Definition 3 and Figure 1.2. These two basic types of Voronoi diagram are of great help to solve a large number of geometric problems. However, the basic Voronoi diagrams may not reflect the real

situation sufficiently. This leads to various *generalized Voronoi diagrams*.

Generalized Voronoi diagrams have been studied by the computational geometry community for a long time already (almost for four decades) [75, 7, 9, 60, 10]. However, lots of questions regarding combinatorial properties and construction algorithms are still open. This is true even for very straightforward generalizations of the basic Voronoi diagrams. This dissertation investigates certain *cluster Voronoi diagrams*. A cluster Voronoi diagram reveals proximity information for a given input family of clusters (sets) of sites or for clusters of the input sites.

The cluster Voronoi diagrams considered in this dissertation are the *Hausdorff Voronoi diagram*, the *farthest-color Voronoi diagram*, and the *farthest-segment Voronoi diagram*. These diagrams are most natural generalizations of the basic nearest-neighbor and farthest Voronoi diagram of points. Each of them can be obtained from one of the two basic Voronoi diagrams of points simply by considering sites to be sets (*clusters*) of points, rather than single points, and by choosing the appropriate distance measure. Another result in this dissertation is establishing a connection of the Hausdorff and the farthest-color Voronoi diagrams with the problem of computing *stabbing circles* for sets of line segments.

For information about the Voronoi diagrams see the book of Aurenhammer et al. [10], and the book by Okabe et al. [60] that contains a comprehensive (to date) overview of the applications of the diagrams.

1.1 Basic concepts of Voronoi diagrams

Before discussing the generalized Voronoi diagrams that are the focus of this dissertation, we review basic concepts necessary to understand the material of this chapter.

Definition 1. The *nearest-neighbor Voronoi diagram* of a finite set of sites in a given space is a subdivision of this space into maximal regions such that all points within one region have the same nearest site.

Throughout this dissertation, unless explicitly stated otherwise, we consider the Euclidean plane as the underlying space (we refer to it as *the plane*, or $\mathbb{R}^2$), the Euclidean norm as a distance measure, and sets of sites, as well as finite clusters of points. We use the term "Voronoi diagram" in two ways: it either denotes the space subdivision (e.g., as in Definition 1), or it denotes the *graph structure* of this subdivision, i.e. the union of the boundaries of all Voronoi regions.

The most celebrated Voronoi diagram is the nearest-neighbor Voronoi diagram *of points in the plane*. Figure 1.1 shows the Voronoi diagram of a set of thirteen

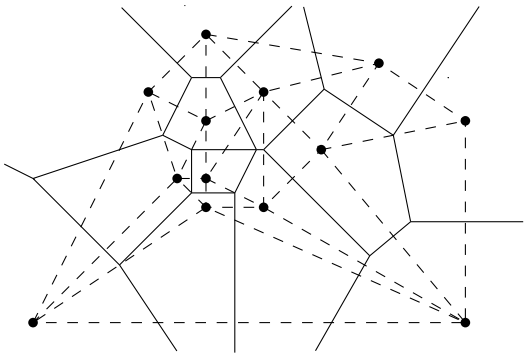


Figure 1.1. A set of points in $\mathbb{R}^2$; its nearest-neighbor Voronoi diagram (solid lines) and Delaunay triangulation (dashed lines)

points in solid lines. This diagram is a plane graph, with one convex face per site. Its combinatorial complexity (size) is linear in the number of sites. The diagram can be preprocessed and stored in a *point-location* data structure, which, for any query point in $\mathbb{R}^2$, returns its nearest site; the time for such query is logarithmic in the number of sites. This basic Voronoi diagram has been fully studied, and optimal construction algorithms have been derived [75, 42, 45].

A geometric structure, closely related to the Voronoi diagram, is the *Delaunay triangulation*.

Definition 2. The *Delaunay triangulation* of a set of point sites in the plane is a triangulation of this set such that for any of its triangles, no sites are enclosed in the interior of the circumcircle of this triangle.

The Delaunay triangulation of a set of points in general position (no four points are co-circular) is dual to the nearest-neighbor Voronoi diagram of that set: two sites are adjacent in the triangulation if and only if their Voronoi regions are neighbors in the diagram. Figure 1.1 shows the Delaunay triangulation of the set of point sites in dashed lines.

Symmetric to the nearest-neighbor Voronoi diagram is the *farthest-site Voronoi diagram* defined as follows.

Definition 3. The *farthest-site Voronoi diagram* of a set of sites is a subdivision of the underlying space into maximal regions, such that every point within one region has the same farthest site.

When the sites are points in the plane, the farthest-site Voronoi diagram is called the *farthest-point Voronoi diagram*. Each Voronoi region in this diagram is unbounded, and the graph structure of the diagram is a tree. Further, a point p

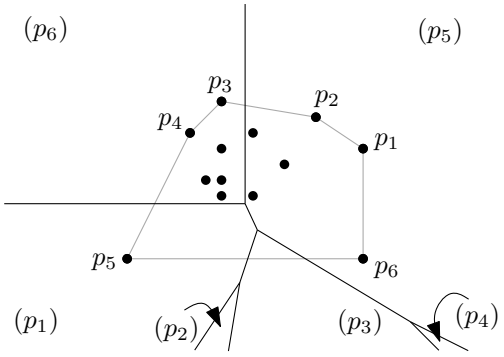


Figure 1.2. The set of points from Figure 1.1, its convex hull (gray lines), and its farthest-point Voronoi diagram (black lines)

has a non-empty Voronoi region if and only if p lies on the convex hull of the set of point sites. The ordering of sites along the convex hull corresponds to the ordering of their Voronoi regions at infinity. Figure 1.2 depicts the farthest-point Voronoi diagram of the same set of points as in Figure 1.1. The convex hull of the set is shown in gray lines, and the points on it are named $p_1, \ldots, p_6$. The Voronoi region of a point $p_i, 1 \leq i \leq 6$, is labeled as (p_i).

An "intermediate" structure between the nearest-neighbor and the farthest-site Voronoi diagrams is the *higher-order Voronoi diagram* [75, 53, 37]. Given a set of n sites, the regions of its *order-k Voronoi diagram* reveal information about k nearest sites, where k is an integer between 1 and $n-1$. The nearest-neighbor Voronoi diagram is the order-1 diagram, and the farthest Voronoi diagram is the order-$(n-1)$ Voronoi diagram.

Another important concept is the *abstract Voronoi diagram*, which reflects the common topological properties of many concrete Voronoi diagrams. Abstract Voronoi diagrams are defined using a system of bisecting curves that satisfy a number of simple combinatorial axioms [50]. As opposed to concrete Voronoi diagrams, no sites or distance functions are considered. The abstract Voronoi diagram machinery [50, 57, 52, 58] can be used to prove complexity bounds and to derive efficient construction algorithms for any concrete Voronoi diagrams that satisfy their axioms [10].

Lifting transformation [40, 37]. The *lifting transformation* is a powerful technique that connects Voronoi diagrams of points in $\mathbb{R}^d$ and arrangements of hyperplanes in $\mathbb{R}^{d+1}$. It lifts a point in $\mathbb{R}^d$ onto the unit paraboloid in $\mathbb{R}^{d+1}$, and returns the tangent hyperplane to the paraboloid at the lifted point.

As applied to points in the plane, the lifting transformation provides an equiv-

alence between the nearest-neighbor or the farthest Voronoi diagram of points and certain substructures of the arrangement of three-dimensional hyperplanes, which are respectively the *upper envelope* and the *lower envelope* of the hyperplanes. For details, see Chapter 2 (Section 2.2.2).

Exploiting this transformation, any algorithm to compute the upper and the lower envelope of hyperplanes may be used to compute the corresponding Voronoi diagram.

1.2 Generalized Voronoi diagrams in this dissertation

The focus of this dissertation is on generalized Voronoi diagrams, termed *cluster Voronoi diagrams*, whose sites are sets, or *clusters*, of points, rather than single points. These diagrams have a number of applications such as in VLSI computer-aided design [65, 55, 67] and in motion planning for robots with obstacles of different shapes, see e.g. a survey by Schwartz and Sharir [72]. They also help to solve problems in facility location: for example, to find a position that is as close as possible to each of k different types of facilities (such as schools, post offices, etc.), or to all the facilities of one type [2].

Cluster diagrams have many variants. Indeed, one can choose differently the form of clusters, the way to measure the distance from a point to a cluster, and restrictions on clusters' size; higher-order Voronoi diagrams are also instances of cluster Voronoi diagrams. Due to such freedom in choices while defining a cluster Voronoi diagram, the properties of different cluster Voronoi diagrams can vary a lot. Classic construction algorithms may not handle well specific properties of their instances. This dissertation considers such instances of cluster Voronoi diagrams.

1.2.1 The Hausdorff Voronoi diagram

Given a set S of points in the plane, we partition S into clusters, resulting in a family F of clusters of points, where no two clusters share a point. The Hausdorff Voronoi diagram of F is a subdivision of the plane according to the nearest cluster, where the closeness between a point and a cluster is measured by the maximum distance. The diagram is formally defined as follows.

Definition 4. The Hausdorff Voronoi region of a cluster $C \in F$ is:

$$\mathrm{hreg}_F(C) = \{p \mid \forall C' \in F \setminus \{C\} \colon \max_{x \in C} d(p,x) < \max_{x \in C'} d(p,x)\};$$

The Hausdorff Voronoi region of a point $c \in C$, $C \in S$ is:

$$\mathrm{hreg}_F(p) = \{p \mid p \in \mathrm{hreg}_F(C) \text{ and } \forall c' \in C \setminus \{c\} \colon d(p,c) > d(p,c')\}.$$

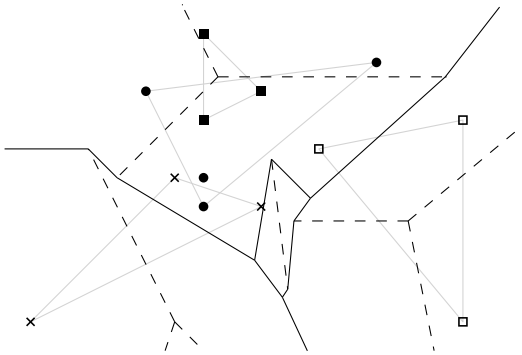


Figure 1.3. A family of four clusters and its Hausdorff Voronoi diagram

The *Hausdorff Voronoi diagram* of F is the subdivision of $\mathbb{R}^2$ derived by the Hausdorff Voronoi regions.

In Figure 1.3, solid lines indicate the boundaries between the Hausdorff Voronoi regions of clusters. Dashed lines indicate the subdivision of the Voronoi region of each cluster into finer regions as provided by the farthest Voronoi diagram of the cluster points.

The Hausdorff Voronoi diagram is a "min-max" type of diagram. It can be viewed as a generalization of the basic Voronoi diagrams. If each cluster in F is a single point, the Hausdorff diagram coincides with the nearest-neighbor Voronoi diagram of F. If F consists of one cluster, the diagram coincides with the farthest-point Voronoi diagram of that cluster. If F consists of all k-subsets of a set S of points, then the diagram coincides with the order-k Voronoi diagram of S.

Let n be the total number of points in all clusters in the input family F, or, equivalently, the number of points in the underlying set S. The Hausdorff Voronoi diagram was first considered by Edelsbrunner et al. [38] under the name *cluster Voronoi diagram*. For arbitrary clusters, the authors proved that the combinatorial complexity of the diagram is $O(n^2\alpha(n))$ and also provided an algorithm of the same time complexity for its construction, where $\alpha(n)$ is the inverse Ackermann function. These bounds were later improved to $O(n^2)$ [66]. When the convex hulls of the clusters are disjoint, the combinatorial complexity of the diagram is $O(n)$. In fact, it remains $O(n)$ for *non-crossing* clusters, a weaker condition than disjointness of convex hulls [66].

Two clusters P and Q are called *non-crossing*, if the convex hull of $P \cup Q$ admits at most two supporting segments with one endpoint in P and one endpoint in Q. If the convex hull of $P \cup Q$ admits more than two such supporting segments, then P and Q are called *crossing*. See Figure 1.4.

Papadopoulou [62] gave a more detailed analysis of the complexity of the Haus-

dorff Voronoi diagram. She showed that its complexity is $O(n+m)$, where m is the *number of crossings* between clusters, and this is tight. The number of crossings m is bounded from above by (half) the number of supporting segments between pairs of crossing clusters, $m = O(n^2)$. For a formal definition, see Definition 7. If all clusters are pairwise non-crossing ($m = 0$), the diagram has size $O(n)$.

Figure 1.4 shows two pairs of non-crossing clusters (Figures 1.4a,b), a pair of clusters with one crossing (Figure 1.4c), and a pair with two crossings (Figure 1.4d).

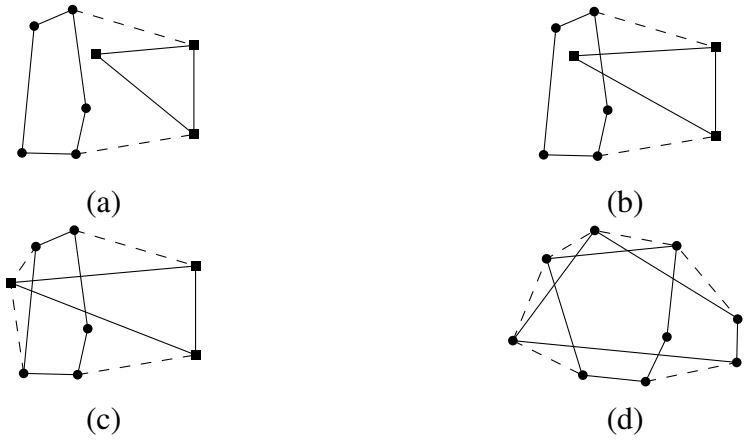


Figure 1.4. Non-crossing clusters with (a) disjoint and (b) non-disjoint convex hulls; crossing clusters with (c) one crossing and (d) two crossings

The $O(n^2)$ time algorithm of Edelsbrunner et al. [38] to construct the Hausdorff Voronoi diagram, although optimal in the worst case, remains quadratic in all cases, even for linear-complexity instances of the diagram.

A large part of this dissertation is dedicated to construction algorithms for the Hausdorff Voronoi diagram, see Chapter 3. We consider separately the case of non-crossing clusters, and for this case we give two randomized algorithms that are near optimal. For arbitrary clusters, we give another randomized algorithm whose time complexity improves the previous results for the families of clusters with small number of crossings.

Algorithms to construct the HVD prior to this dissertation

The first $O(n^2)$ time algorithm is given by Edelsbrunner et al. [38]. There are *plane sweep* and *divide and conquer* algorithms for constructing the Hausdorff Voronoi diagram of arbitrary clusters [62, 66]. Both algorithms have a $K \log n$ term in their time complexity, where K is a parameter reflecting the number of pairs of

clusters such that one is contained in a specially defined enclosing circle of the other, for example, the minimum enclosing circle [66]. However, K can be $\omega(n)$ (superlinear), even in the case of non-crossing clusters when the diagram has linear complexity.

For non-crossing clusters, the Hausdorff Voronoi diagram can be computed using the generic randomized incremental framework for abstract Voronoi diagrams of Klein et al. [52]. The expected time complexity of this algorithm is $O(bn\log n)$, where b is the time to compute the bisector between two clusters [1]. If there are clusters of linear size, then b can be $\Theta(n)$.

A parallel algorithm is given by Dehne et al. [33]. It constructs the Hausdorff Voronoi diagram of non-crossing clusters in $O(p^{-1}n\log^4 n)$ time with p processors, which implies a divide and conquer sequential algorithm of time complexity $O(n\log^4 n)$ and space complexity $O(n\log^2 n)$.

1.2.2 An industrial application of the Hausdorff Voronoi diagram

The Hausdorff Voronoi diagram finds direct application in VLSI computer-aided design [65, 18]. In fact, the intensive study of the diagram was motivated by this application. In particular, the use of the Hausdorff Voronoi diagram lowered considerably the time required to measure the *critical area* of a chip design [55], which is one of the key parameters in VLSI *yield prediction*.

Nowadays, a VLSI chip has all its elements packed within a so small area (a few nanometers), that even a tiny particle can easily cause a fault during the printing process. Since printing a chip is an expensive operation, a crucial problem for the semiconductor industry is predicting the yield (the percentage of successfully printed chips) prior to the printing phase, that is, only given a *design* of the chip. One of the important and widely used quality measures of a chip design is *critical area*, that represents the sensitivity of a design to random defects. For the extensive overview of the yield analysis tasks and the use of generalized Voronoi diagrams to address them, see the book chapter by Gupta and Papadopoulou [46]. Here we consider only the part that is directly connected to the Hausdorff Voronoi diagram.

One of the standard types of faults is a *via block*, and it consists on the blockage of the connection (via) between conducting regions in two different layers of the VLSI chip. In particular, a defect entirely covers a certain contact region between two layers. Defects are modeled as circles, and the contact regions as polygons. Given a point p on a via layer (an intermediate layer between two conducting ones), the *critical radius* of p is the smallest defect centered at p that causes a via block. Designers typically use *redundant* vias to increase the reliability of their design against manufacturing defects. Thus a via block happens if a particle overlaps

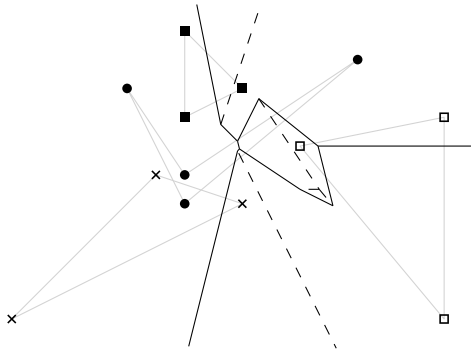


Figure 1.5. The farthest-color Voronoi diagram of the family of clusters from Figure 1.3

the entire cluster of the (redundant) vias. The Hausdorff Voronoi diagram of the clusters of redundant vias on the via layer provides the critical radius for every point on that layer. The critical area is an integral of the critical radius over the points on the via layer. Once the Hausdorff Voronoi diagram of the clusters of vias is available, the integral reduces to a summation over the Voronoi edges [61]. The critical area computation for via blocks can thus be reduced to construction of the Hausdorff Voronoi diagram.

The Hausdorff Voronoi diagram is useful for other problems in VLSI design, such as the *geometric min-cut* problem [68], motivated by redundant interconnects [63].

1.2.3 The farthest-color Voronoi diagram

A cluster Voronoi diagram of a "max-min" type is the *farthest-color Voronoi diagram* (see Figure 1.5) that can be regarded as the opposite to the Hausdorff Voronoi diagram. For a family F of clusters of points, its farthest-color Voronoi diagram is formally defined as follows.

Definition 5. The farthest-color Voronoi region of a cluster $C \in F$ is:

$$\mathrm{fcreg}_F(C) = \{p \mid \forall C' \in F \setminus \{C\} \colon \min_{x \in C} d(p,x) > \min_{x \in C'} d(p,x)\};$$

The farthest-color Voronoi region of a point $c \in C$, $C \in F$ is:

$$\mathrm{fcreg}_F(c) = \{p \mid p \in \mathrm{fcreg}_F(C) \text{ and } \forall c' \in C \setminus \{c\} \colon d(p,c) < d(p,c')\}.$$

The *farthest-color Voronoi diagram* of F, for brevity $\mathrm{FCVD}(F)$, or simply FCVD, is the subdivision of $\mathbb{R}^2$ derived by the farthest-color Voronoi regions.

In other words, the farthest-color Voronoi diagram of F subdivides $\mathbb{R}^2$ into maximal regions, such that every point within one region has the same farthest cluster, where the closeness between a point in $\mathbb{R}^2$ and a cluster is measured by the minimum distance from that point to a point in the cluster.

Figure 1.5 illustrates the farthest-color Voronoi diagram of four clusters. Solid lines indicate the edges that separate the farthest-color Voronoi regions of clusters. Dashed lines show the further subdivision of the region of an individual cluster by the nearest-neighbor Voronoi diagram of the point of this cluster.

The farthest-color Voronoi diagram was first considered by Huttenlocher et al. [47], who showed that its complexity is $O(nk\alpha(nk))$ and gave an $O(nk\log nk)$ time construction algorithm, where k is the number of clusters and n is the total number of points in all clusters. Independently, Abellanas et al. [2] considered this diagram, and gave a tighter $O(nk)$ bound on its complexity, and a matching $\Omega(nk)$ lower bound for $k \leq n/2$. They also provide an $O(n^2\alpha(k)\log k)$ time construction algorithm. An $O(n^2)$ time algorithm is implied by the result of Edelsbrunner et al. [38] (see also Property 2.2.3). Cheong et al. [23] considered the case when sites (clusters) form pairwise disjoint simple polygons. They call the FCVD of such sites the *farthest-polygon Voronoi diagram*, and show that it has $O(n)$ complexity and can be constructed in $O(n\log^3 n)$ time, where n is the total number of vertices in all polygons [23].

The farthest-color Voronoi diagram has been studied much less than the Hausdorff Voronoi diagram. Its structural properties are not yet well understood. In particular, so far there is no known analog of the non-crossing condition of the Hausdorff Voronoi diagram, that would guarantee the linear combinatorial complexity of the FCVD for a certain class of inputs.

In this dissertation, the farthest-color Voronoi diagram is involved through its application to the stabbing circle problem. We consider some special cases as they appear in our study of the stabbing circle problem. In particular, the clusters are pairs of points, and the line segments that connect them are either (i) parallel to each other, or (ii) are edges of the Delaunay triangulation of some set of points.

1.2.4 The farthest-segment Voronoi diagram

The *farthest-segment Voronoi diagram* is the farthest-site Voronoi diagram, where sites are line segments in $\mathbb{R}^2$, see Definition 3. Figure 1.6 illustrates a set of five line segments in black lines and its farthest-segment Voronoi diagram in red lines. Each face of the diagram is labeled by the name (in parentheses) of the line segment, farthest to it.

Despite being the most natural and most immediate generalization of the classic

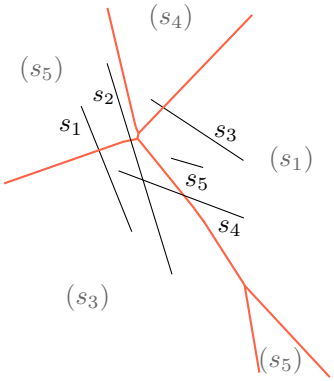


Figure 1.6. ([64]) Set $S = \{s_1, \ldots, s_5\}$ and its farthest-segment Voronoi diagram

farthest-point Voronoi diagram, this diagram has properties surprisingly different from the latter, as pointed out by Aurenhammer et al. [8]. For example, a single Voronoi region may have disconnected faces, see the region of the segment s_5 in Figure 1.6. In fact, the number of disconnected faces of one Voronoi region can be $\Theta(n)$, where n is the number of segments in the input set. However, the total combinatorial complexity of the farthest-segment Voronoi diagram is $O(n)$ [8].

The convex hull of the sites does not determine the non-empty Voronoi regions, neither does it determine the ordering of the faces at infinity. There is, however, another structure, called the *Gaussian map*, introduced by Papadopoulou and Dey [64], that determines the ordering of non-empty faces at infinity, and thus plays the role of convex hull for the farthest-segment Voronoi diagram.

The farthest-point Voronoi diagram can be constructed in linear time, if the convex hull of the sites is given [3]. In this dissertation we show that, despite the differences, the farthest-segment Voronoi diagram also can be constructed in linear time, given the ordering of its faces at infinity.

1.3 Algorithmic techniques to construct Voronoi diagrams

Several algorithmic techniques have been explored to construct Voronoi diagrams efficiently. These are typically *divide and conquer*, *plane sweep*, and *randomized incremental construction*.

Divide and conquer [75]. The *divide and conquer* technique to construct a Voronoi diagram, in its first ("divide") phase, subdivides the sites into two sufficiently large subsets. This is usually, but not necessarily, done with respect to a straight line. This breaks the problem into two subproblems, which are then solved recursively. That is, the Voronoi diagrams of the two subsets are recursively computed. In the second ("conquer") phase, the two Voronoi diagrams are *merged*. To merge two diagrams, their portions that must appear in the final Voronoi diagram are stitched together along the so-called *merge curves*.

The key conditions for an $O(n \log n)$ time divide-and-conquer algorithm to construct a Voronoi diagram are: (1) the size of both subproblems must be at least a constant fraction of the size of the whole problem, with the two corresponding constants summing up to 1; (2) while determining the merge curve of the two diagrams, one visits only those parts of the diagrams that do not appear (or appear only partially) in the resulting merged diagram. The latter condition implies a linear-time merging procedure. Given this property and condition (1), *the Master theorem* [31] ensures an $O(n \log n)$ time complexity for the whole algorithm.

Plane sweep [42]. The *plane sweep* technique handles the two dimensions of the problem separately. In particular, it considers a vertical line, called a *sweepline*, that sweeps the plane from left to right. While this process is advancing, the algorithm maintains the Voronoi diagram of all the sites lying to the left of the sweepline, together with the sweepline as an additional site. The diagram is updated only when the sweepline hits an *event point*, which is either a new site, or an intersection of two Voronoi edges. The "dynamic" part of the diagram, which needs to be kept track of, is the boundary of the Voronoi region of the sweepline. This boundary is called the *wavefront*. Storing the wavefront and the pending event points in appropriate dynamic data structures yields an efficient algorithm.

Randomized incremental construction (RIC) [26, 45]. A randomized incremental construction algorithm inserts sites one by one in random order, and updates the diagram after each insertion. Such algorithm has a good expected time complexity, while still enjoying the simplicity of an incremental algorithm. The randomized incremental construction (RIC) framework was introduced by Clarkson and Shor [26]. It provides a powerful method to construct a geometric structure, should the construction problem be formulated in the generic terms of the framework and satisfy certain *update conditions*. The classic RIC framework has two variations, depending on the auxiliary data structure it uses to perform efficient updates at each incremental step. In particular, it uses either a *conflict graph*,

or a *history graph*. Applying this framework, many Voronoi diagrams have been computed in expected $O(n \log n)$ time [44, 45, 52, 58].

There is another variation of RIC [35, 48], which is based on point location. This technique is specific to the proximity-based geometric structures, such as the Delaunay triangulation and Voronoi diagrams. To perform updates at each incremental step efficiently, it uses a dynamic hierarchical data structure called *Voronoi hierarchy* [48]. This technique is practical and simple, and it has been implemented in the CGAL library [19], despite its $O(n \log^2 n)$ time complexity for Voronoi diagrams, which is worse than the classic RIC.

A linear-time construction technique for points in convex position [3]. It is well known that the task to construct the nearest-neighbor Voronoi diagram of a set of points is at least as hard as to construct a convex hull, thus it requires $\Omega(n \log n)$ time [70]. However, if the points are in convex position, and their ordering along the convex hull is known in advance, then their Voronoi diagram can be computed in $O(n)$ time [3].

The technique to achieve this is a variation of the divide-and-conquer approach. The difference with the standard divide-and-conquer strategy is as follows. At each step during the "divide" phase, the algorithm takes out of consideration a constant fraction of sites, called *crimson sites*. The algorithm continues execution for the remaining sites only. During the "conquer" phase, in addition to the standard merging of the recursively computed Voronoi diagrams, the crimson sites are inserted one by one. The key property of the crimson sites is that their Voronoi regions are pairwise non-adjacent. The existence of such a set of crimson sites was shown by Aggarwal et al. in their *combinatorial lemma* [3], which also provides the linear-time procedure to choose a crimson set.

The Aggarwal et al. technique is so powerful that it was applied to solve many problems of similar nature. First, it immediately applies to a list of problems, resulting in a linear-time algorithm for each of them [3]. This list includes: constructing the farthest-point Voronoi diagram, when the convex hull of the sites is known; updating the nearest-neighbor Voronoi diagram of points after deletion of one site; computing the order-$(k+1)$ subdivision of a face of the order-k Voronoi diagram of points. The last result implies a speedup by a logarithmic factor in the incremental construction of higher-order Voronoi diagram of points. In particular, the time complexity of a construction by Lee [53] can be improved from $O(k^2 n \log n)$ to $O(k^2 n)$. Further, the technique has been exploited by Chin et al. to construct the medial axis of a simple polygon in linear time [24].

The technique has been extended to handle certain types of abstract Voronoi

diagrams. Klein and Lingas [51] show how to use it to compute the *Hamiltonian abstract Voronoi diagrams*. The Hamiltonian abstract Voronoi diagrams are abstract Voronoi diagrams with the following additional property: an unbounded curve is known, which intersects each Voronoi region exactly once, and may intersect each bisector at most once. Recently, Bohler et al. further generalized the algorithm to treat the *forest-like abstract Voronoi diagrams* [14]. The forest-like abstract Voronoi diagrams are abstract Voronoi diagrams, such that every site has a unique face, and the diagram of the input set is a tree, but the diagram of a subset may be a forest.

In Chapter 4 of this dissertation, we extend this technique to construct the farthest-segment Voronoi diagram after the sequence of its faces at infinity is known. The farthest-segment Voronoi diagram may have disconnected Voronoi regions, and we augment the Aggarwal et al. technique with the ability to handle this issue.

This dissertation. The algorithms in this dissertation exploit the techniques reviewed above. Chapter 3 investigates application of the RIC framework, both the one based on point location and the classic one, to construct the Hausdorff Voronoi diagram. We give a more detailed review of the RIC framework in Chapter 2 (Section 2.1). Chapter 4 applies the Aggarwal et al. linear-time technique to construct the farthest-segment Voronoi diagram.

1.4 The stabbing circle problem: an application of cluster Voronoi diagrams

The stabbing circle problem poses questions about the classification of a family of pairs of points according to their containment in a circle. For our purposes, it is convenient to represent a pair of points by a line segment connecting them, and we stick to such representation.

Let S be a set of n line segments in the plane.

Definition 6. A circle c is a stabbing circle for set S if exactly one endpoint of each segment in S is contained in the exterior of the closed disk induced by c; see Figure 1.7.

A stabbing circle intersects all the segments in S and classifies their endpoints into two classes, depending on whether or not the endpoint lies in the exterior of the corresponding disk. Two stabbing circles are *combinatorially different* if they classify the endpoints of S differently.

Figure 1.7. Left: Segment set with a stabbing circle; Right: Segment set with no stabbing circle

The *stabbing circle problem* asks the following questions: (1) Report a representation of all the combinatorially different stabbing circles for S; and (2) Find the stabbing circles of minimum and maximum radius.

The stabbing circle problem is a generalization of the famous *stabbing line problem* [36]. The stabbing line can be solved in optimal $O(n \log n)$ time [36, 37]. There is a matching $\Omega(n \log n)$ lower bound, in the fixed order algebraic decision tree model, to decide whether a stabbing line exists [11]. If all segments are parallel, the stabbing line problem can be solved in $O(n)$ time by linear programming.

It is natural to ask about types of *stabbers* other than lines. Variations of the problem include stabbing wedges [28], as well as isothetic stabbing strips, quadrants and 3-sided rectangles [29].

The *stabbing circle problem* was posed by Claverol in her PhD dissertation [27]. She gives an algorithm that in $O\left(n^2\sqrt{n}\log n\right)$ time computes the stabbing circle of minimum radius (if one exists) for a set of arbitrary segments, and an $O(n^2)$ time algorithm for the case when all segments are parallel. She also provides an example of a segment set that admits $\Omega(n^2)$ combinatorially different stabbing circles.

Chapter 5 of this dissertation continues exploring this problem. We show that the stabbing circle problem is tightly connected to *cluster Voronoi diagrams*, specifically, the *Hausdorff* and the *farthest-color Voronoi diagram*, where clusters are the endpoints of segments in S. In particular, any point $p \in \mathbb{R}^2$ is the center of a stabbing circle for S if and only if p is closer to its farthest than to its nearest cluster. Here by the farthest and the nearest cluster of a point we mean respectively the owner of the region of this point in the farthest-color or the Hausdorff Voronoi diagram.

The above connection is not only interesting on its own right, but it also yields a method to solve the stabbing circle problem. For sets of line segments, satisfying certain conditions, we show that our method leads to efficient algorithms, improving the previously known results. In particular, it results in $O(n \log^2 n)$ time algorithm for the parallel segments. Further, if all segments are edges of a Delau-

nay triangulation of some set of points, and either all segments are parallel or all have same length, the algorithm has optimal $O(n \log n)$ time complexity.

1.5 Dissertation contributions

The main focus of this dissertation is on cluster Voronoi diagrams. The goal is to improve the state of the art for these diagrams. In particular, we study their properties, and design new construction algorithms of better complexity. We also give an algorithmic application, involving the stabbing circle problem. The rest of this section gives a detailed overview of the contributions.

Randomized incremental construction (RIC) of the Hausdorff Voronoi diagram (Chapter 3)

We investigate the RIC framework as applied to the Hausdorff Voronoi diagram of a family of k clusters, having n points in total. We present two different randomized incremental approaches to construct the diagram, that improve all previous results for this problem.

1. A RIC for the Hausdorff Voronoi diagram of non-crossing clusters, based on point location (Section 3.2).

    - We show that the Hausdorff Voronoi diagram can be constructed in expected $O(n \log n \log k)$ time and expected $O(n)$ space, following the variant of the RIC approach that is based on point location.
    - We augment the Voronoi hierarchy with the ability to handle the features of the Hausdorff Voronoi diagram of non-crossing clusters that were not handled originally by the Voronoi hierarchy. These features include:
        - sites of non-constant complexity;
        - sites not enclosed in their Voronoi regions;
        - empty Voronoi regions.

      The proposed ways of handling those features may be of help in similar situations with other generalized Voronoi diagrams.
    - We also extend the Voronoi hierarchy to perform *parametric point location queries* in expected $O(\log n \log k)$ time.

- We show how to preprocess the farthest Voronoi diagram of the points of an individual cluster, building the *centroid decomposition* data structure, that can perform the point location and the *segment* queries in $O(\log n)$ time.

2. We apply the (classic) Clarkson-Shor RIC framework [26, 25] to the Hausdorff Voronoi diagram (Sections 3.3 and 3.4). We consider both non-crossing and arbitrary clusters.

   - For non-crossing clusters, we propose a simplified definition of a conflict, that is based on the properties of the Hausdorff Voronoi diagram. This yields an algorithm to construct the diagram in expected $O(n\log n + k\log n\log k)$ time and deterministic $O(n)$ space. The complexity of our algorithm is considerably better than one that would follow from a straightforward application of the RIC framework.
   - We apply the classic RIC framework to the Hausdorff Voronoi diagram of arbitrary clusters that may cross. We address the problem of constructing a Voronoi diagram with disconnected regions and disconnected bisectors, via the randomized incremental construction framework. Such setting has not been considered before. The resulting algorithm runs in expected $O((m + n\log k)\log n)$ time and expected $O(m + n\log k)$ space, where m is the total *number of crossings* between clusters.

Linear-time construction of the farthest-segment Voronoi diagram (Chapter 4)

We show that, given the sequence of Voronoi faces at infinity, the farthest-segment Voronoi diagram of n line segments can be computed in $O(n)$ time. Our algorithm follows the linear-time framework to compute the Voronoi diagram of points in convex position [3], and augments this framework with the ability to handle disconnected Voronoi regions.

Application of the cluster Voronoi diagrams to the stabbing circle problem (Chapter 5)

- We point out that the stabbing circle problem for segments in the plane can be reduced to the problem of computing stabbing planes for segments in $\mathbb{R}^3$. The latter problem can be solved in $O(n^2)$ time [38], and the reduction can be

performed in $O(n)$ time. Moreover, there is a matching $\Omega(n^2)$ lower bound on the number of combinatorially different stabbing circles, i.e., the size of the output for the stabbing circle problem [27]. Thus, based on previous results, we establish the tight worst-case time complexity for this problem.

- We point out the connection between the stabbing circle problem and the cluster Voronoi diagrams. In particular, the existence of a stabbing circle centered at a given point p in the plane is fully determined by the Hausdorff Voronoi region and the farthest-color Voronoi region that contain p.
- Building upon the connection to cluster Voronoi diagrams, we present an approach to solve the stabbing circle problem. This approach complements the $O(n^2)$ time solution. The latter is worst-case optimal, but in certain cases our alternative approach performs better, as it is sensitive to properties of the cluster Voronoi diagrams.
- We prove that for parallel segments our alternative approach works in $O(n\log^2 n)$ time. This improves considerably upon the best previously known $O(n^2)$ time complexity bound for this setting.
- For segments that are disjoint edges of the Delaunay triangulation of some set of points (we say that such segments *satisfy the Delaunay property*), and in addition that either have equal length, or are parallel to each other, we show that the stabbing circle problem can be solved in $O(n\log n)$ time.

Some results on the farthest-color Voronoi diagram (Chapter 5)

As part of our study of the stabbing circle problem, we prove the following results on the farthest-color Voronoi diagram (assuming S to be a set of n line segments).

- If all segments in S are parallel to each other, the farthest-color Voronoi diagram of pairs of their endpoints has $O(n)$ combinatorial complexity. Moreover, it can be constructed in $O(n\log n)$ time.
- If the segments in S are disjoint edges of the Delaunay triangulation of some set of points (i.e., the segments satisfy the above Delaunay property), then the farthest-color Voronoi diagram is an instance of the *farthest abstract Voronoi diagram* [58]. This fact implies that its graph structure is a tree, and that its combinatorial complexity is $O(n)$.
- For segments satisfying the Delaunay property we independently give an algorithm to construct the farthest-color Voronoi diagram in deterministic $O(n\log n)$ time. (Note that being an instance of farthest abstract Voronoi diagrams leads only to an expected $O(n\log n)$ time algorithm.)

Dissertation overview

This dissertation is organized as follows. Chapter 2 reviews various important concepts that are used in this dissertation. Chapter 3 presents our results on the randomized incremental construction of the Hausdorff Voronoi diagram. Chapter 4 presents a linear-time construction algorithm for the farthest-segment Voronoi diagram, after the sequence of faces at infinity is known. Chapter 5 addresses the stabbing circle problem. It provides an algorithmic application of cluster Voronoi diagrams to the stabbing circle problem, as well as the results on the farthest-color Voronoi diagram.

1.6 List of publications

Chapter 3 is based on the following papers:

- *P. Cheilaris, E. Khramtcova, S. Langerman, and E. Papadopoulou. A randomized incremental algorithm for the Hausdorff Voronoi diagram of non-crossing clusters.* Algorithmica*, 2016. DOI 10.1007/s00453-016-0118-y.*

  Preliminary version as a conference paper:

  P. Cheilaris, E. Khramtcova, S. Langerman, and E. Papadopoulou. A randomized incremental approach for the Hausdorff Voronoi diagram of non-crossing clusters. In Alberto Pardo and Alfredo Viola, editors, LATIN 2014: Theoretical Informatics - 11th Latin American Symposium*, volume 8392 of* LNCS*, pages 96–107. Springer, 2014.*

  Preliminary short versions:

  P. Cheilaris, E. Khramtcova, and E. Papadopoulou. Randomized incremental construction of the Hausdorff Voronoi diagram of non-crossing clusters. In Abstracts of the 29th European Workshop on Computational Geometry (EuroCG'13)*, pages 159–162, 2013.*

- *E. Khramtcova, and E. Papadopoulou Randomized Incremental Construction for the Hausdorff Voronoi Diagram of Point Clusters. Submitted to a conference.*

  Preliminary short versions:

  E. Khramtcova and E. Papadopoulou. A simple RIC for the Hausdorff Voronoi diagram of non-crossing clusters. In Abstracts of the 31th European Workshop on Computational Geometry (EuroCG'14).

E. Khramtcova and E. Papadopoulou. Randomized incremental construction for the Hausdorff Voronoi diagram. In Young Researchers Forum of Computational Geometry Week (CG YRF'15)

Chapter 4 is based on the following paper:

- *E. Khramtcova and E. Papadopoulou. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In* Proc. Algorithms and Computation - 26th International Symposium, ISAAC*, pages 404–414, 2015.* Journal version in preparation.

  Preliminary version:

  E. Khramtcova and E. Papadopoulou. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In Abstracts of the 31th European Workshop on Computational Geometry (EuroCG'15)*, pages 252–255, 2015.*

Chapter 5 is based on the following papers:

- *M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for sets of segments in the plane. In* LATIN 2016: Theoretical Informatics - 12th Latin American Symposium*, pages 290–305, 2016.* Invited to the special issue of Algorithmica on LATIN 2016 (20% of conference papers are invited).

  Preliminary version:

  M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for sets of segments in the plane. In Abstracts of 16th Spanish Meeting on Computational Geometry (EGC'15)*, pages 112–115, 2015.*
- *M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for some sets of Delaunay segments. In* Abstracts of the 32th European Workshop on Computational Geometry (EuroCG'16)*, pages 139–142, 2016.*

Chapter 2

Background

In this chapter we give a review on the concepts and techniques that are the basis of the results in this dissertation. We first discuss the randomized incremental construction framework. Then we give an overview of the geometric transformation, including the point-line duality, the lifting transformation, and the duality for the fartest-segment Voronoi diagram.

2.1 The randomized incremental construction (RIC) framework

Clarkson and Shor [26] introduced a generic framework, which yields simple and efficient construction algorithms for various geometric structures. Several extensions have been proposed on both the algorithmic [25, 15, 45, 34] and the analytical [45, 73, 76] part. The framework is discussed in detail in the books of Boissonnat and Yvinec [17] and of Mulmuley [59]. We term it "classic RIC framework" in order to distinguish it from another variant of randomized incremental construction, which is based on point location. Both variations of the RIC are reviewed in the next two sections. Our presentation of the classic RIC (Section 2.1.1) follows the presentation of the book of Boissonnat and Yvinec [17].

2.1.1 Classic RIC framework

Suppose we are given a set of objects (sites), and we aim to construct a certain geometric structure (target structure) induced by this given set of sites. The framework operates with *objects*, *ranges* and *conflicts*, whose definitions are specific to the concrete problem. A randomized incremental algorithm to construct the target

structure, or RIC for brevity, inserts objects one by one in random order, updating the target structure at each step. *Ranges*[1] are simple subsets of the space, *defined by* a constant number of objects. The target structure at each step is seen as a collection of ranges which are *defined* and *without conflicts* over the set of objects inserted so far. While the target structure at two consecutive steps may differ a lot, the expected total number of ranges created during the whole execution of the algorithm is bounded by a function of the complexity (size) of the target structure, see Theorems 2.1.1 and 2.1.2. In particular, the expected number of created ranges is linear for the target structures whose size is linear in the number of sites. To update the target structure efficiently, an auxiliary data structure is maintained, which is either a *conflict graph*, or a *history graph*.

Conflict graph. The *conflict graph* is a bipartite graph, defined as follows. At each step of the algorithm, one of the two groups of nodes of the conflict graph corresponds to ranges comprising the current target structure, and the other group of nodes corresponds to objects that are not yet inserted. Two nodes are connected by an edge if the corresponding range and object are in conflict. An algorithm using a conflict graph satisfies a sufficient *update condition* if the time to update the conflict graph at each step of the algorithm is proportional to the number of edges of the conflict graph, deleted or created during this step. This condition yields good expected time and space complexity:

Theorem 2.1.1 (Thm 5.2.3 [17])**.** *Let S be a set of n objects, and consider a randomized incremental algorithm that uses a conflict graph to process S. Let $f_0(r,S)$ be the expected number of ranges in the target structure for a random sample of r objects from S.*

1. *The expected total number of ranges created by the algorithm is*
$$O\left(\sum_{r=1}^{n} \frac{f_0(r,S)}{r}\right).$$
2. *The expected total number of arcs added to the conflict graph by the algorithm is*
$$O\left(n\sum_{r=1}^{n} \frac{f_0(r,S)}{r^2}\right).$$

[1] We use the original [26] term *range*, instead of a more usual term *region* [17], to avoid the confusion with the Voronoi regions latter.

3. *If the algorithm satisfies the update condition, then its expected complexity (both in time and space) is*

$$O\left(n\sum_{r=1}^{n}\frac{f_0(r,S)}{r^2}\right).$$

The above framework requires that the whole input set of sites is available from the beginning. That is, the framework yields *off-line* randomized incremental construction algorithms. In order to drop this requirement and thus make the algorithms *on-line*, a variation of the framework was designed, that, instead of a conflict graph, uses a *history graph*.

History graph. The *history graph* [25, 15], also called *influence graph*, is a directed acyclic graph defined as follows. At each step of the algorithm, the nodes of the history graph correspond to all ranges created by the algorithm so far. The nodes with zero out-degree, termed *leaf nodes*, correspond to the ranges present in the current target structure. The nodes that have outgoing edges, termed *intermediate nodes*, correspond to the ranges that have been already deleted from the target structure. Intermediate nodes are connected, by their outgoing edges, to the nodes whose insertion caused the deletion of their ranges. The latter nodes are referred to as the *children* of the former nodes. The *update condition for a history graph* is the following: (1) The out-degree of each node is bounded by a constant; and (2) the existence of a conflict between a given range and a given object can be tested in constant time. This update condition yields similar bounds on the complexity of a randomized incremental algorithm, as in in the case of conflict graph:

Theorem 2.1.2 (Thm 5.3.4 [17])**.** *Consider a RIC algorithm that uses a history graph to process a set S of n objects. Let $f_0(r,S)$ be the expected number of ranges in the target structure for a random sample of r objects from S. If the algorithm satisfies the update condition, then:*

1. *The expected storage used by the algorithm to process the n objects is*

$$O\left(\sum_{r=1}^{n}\frac{f_0(r,S)}{r}\right).$$

2. *The expected time complexity of the algorithm is*

$$O\left(n\sum_{r=1}^{n}\frac{f_0(r,S)}{r^2}\right).$$

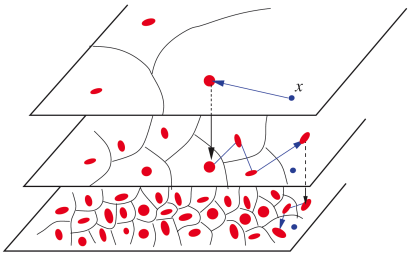


Figure 2.1. ([16]) The Voronoi hierarchy for the Voronoi diagram of convex objects, consisting of three levels; walks for a query point x at each level (solid lines)

For some generalized Voronoi diagrams it is shown how to choose objects, ranges and conflicts so that the above update conditions hold. Klein et al. [52] presented the application of the RIC framework to abstract Voronoi diagrams. This was followed by a similar result on the farthest abstract Voronoi diagram [58]. These results imply that any Voronoi diagram which is an instance of the nearest of farthest abstract Voronoi diagrams can be computed in expected $O(n \log n)$ time[2] via the RIC framework. Among the generalized Voronoi diagrams in the plane that are not instances of these abstract diagrams, the RIC framework has been applied, for example, to the Voronoi diagram of *curved objects*, where bisectors may be closed curves [4].

2.1.2 RIC based on point location

For Voronoi diagrams and the Delaunay triangulation, there is an alternative on-line randomized incremental technique, which is based on point location. Algorithms following this technique, are simple and practical, despite often being inferior to the classic RIC framework in terms of time complexity. Such algorithms have been implemented in the *CGAL* library [19].

At each incremental step, when a new object is inserted, the target structure is updated in the following way. First, the range that contains a *representative point* is located, where a representative point is guaranteed to belong to one of the new ranges after the update. Then, starting from this point, the structural changes in the target structure are traced, and the target structure is updated.

To perform the necessary location queries efficiently, a semi-dynamic data structure is maintained. Originally the *Delaunay hierarchy* was proposed by Dev-

[2]More precisely, these algorithms perform $O(n \log n)$ *basic operations*, which yields $O(n \log n)$ time, assuming that a basic operation requires $O(1)$ time.

illers [35] as such data structure; its use resulted in an algorithm to construct the Delaunay triangulation of n points in expected optimal $O(n \log n)$ time [35]. Later, the Delaunay hierarchy inspired the *Voronoi hierarchy* for the nearest-neighbor Voronoi diagram of convex objects, presented by Karavelas and Yvinec [48]. The Voronoi hierarchy is also presented in a book chapter by Boissonat et al. [16]. Using this data structure leads to an $O(n \log^2 n)$ time construction algorithm for the Voronoi diagram of convex objects.

The Voronoi hierarchy. The Voronoi hierarchy for a set of n sites consists of an expected $O(\log n)$ levels: The zero level corresponds to all the sites, and stores their Voronoi diagram. Each consecutive level corresponds to a random sample of sites of the previous level following a Bernoulli distribution with a constant parameter. Point location in this structure is done by performing *walks* level by level, starting from the topmost level. Each walk goes from the site found at the previous level to the closest site on the current level. Each step of the walk proceeds from a site to one of its neighbors, such that the distance to the query point is reduced. See Figure 2.1. The Voronoi hierarchy (as well as the Delaunay hierarchy) can be viewed as a 2D analog of the *skip lists* [71].

2.2 Geometric transformations

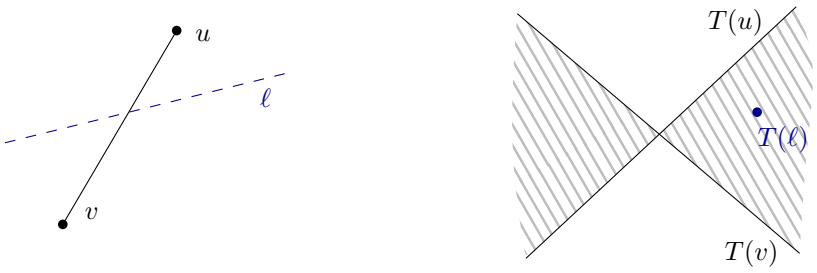
Geometric transformations are a powerful tool in computational geometry that help to gain insights in studying a geometric structure through its connection with other seemingly unrelated structures.

2.2.1 Duality and related transformations for line segments in the plane

Point-line duality is a well-known transformation that connects geometric structures involving points to certain substructures of arrangements of lines. The transformation can also handle line segments, which is useful in this dissertation. In its general form, the duality transformation maps points in $\mathbb{R}^d$ to hyperplanes in $\mathbb{R}^d$ and vice versa. Here we consider the two-dimensional case.

The transformation T maps a point $p = (a,b)$ in the primal plane to the line $T(p) : y = ax - b$ in the dual plane. It also maps a line $\ell : y = ax + b$ in the primal plane to the point $T(\ell) = (a,-b)$ in the dual plane.

Transformation T has an important property that it reverses the *above/below* relationship between the geometric objects. In particular, a point p is above a line



(a) (b)

Figure 2.2. Point-line duality transformation: (a) a line segment uv in the primal plane, and a line ℓ that intersects it; (b) the left-right double wedge formed by lines $T(u)$ and $T(v)$, and the point $T(\ell)$ inside it

ℓ if and only if the dual point $T(\ell)$ is above the dual line $T(p)$. Similarly, the transformation T preserves the incidence relationship.

The above duality transformation proved useful for many problems involving point sets, such as the testing if a point set contains three collinear points, computing the *visibility graph*, and computing *ham-sandwich cuts* [41, 37, 54, 21, 32]. In the dual setting, these problems involve arrangements of lines that are intuitively easier to address.

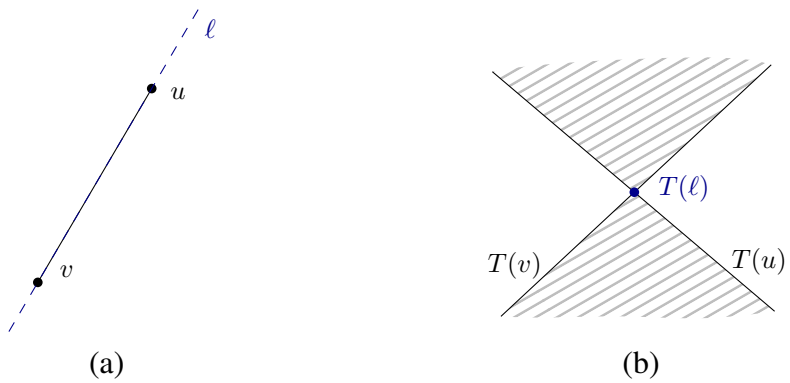
Point-line duality and the stabbing line problem

An example of the power of the duality transformation is its application to the *stabbing line problem* [36, 37].

Let S be a set of segments in the plane. A line ℓ is a stabbing line for S if ℓ intersects all segments in S. The *stabbing line problem* is to construct (a representation of) all the stabbing lines for a given segment set S.

The duality transformation turns this problem into computing the intersection of *double wedges* in the plane as follows. Consider a line segment uv. Any (non-vertical) line ℓ that intersects uv is below one of its endpoints and above the other; see Figure 2.2. The point $T(\ell)$ dual to line ℓ lies in the *left-right double wedge* formed by the lines $T(v)$ and $T(u)$. The left-right double wedge is the union of the two (out of the four possible) areas bounded by these lines, that do not contain the vertical line, shown shaded in Figure 2.2b.

A non-vertical line ℓ is a stabbing line for S if and only if point $T(\ell)$ lies in the intersection of the double wedges of all the line segments in S. This intersection region is not necessarily connected, but it consists of at most $n+1$ potentially



(a) (b)

Figure 2.3. Point-line duality transformation: (a) a line segment uv, and its supporting line ℓ; (b) the upper and lower wedges formed by lines $T(u)$ and $T(v)$, and the point $T(\ell)$

unbounded convex polygons. The combinatorial complexity of the boundary of this intersection region is $O(n)$. It can be computed in $O(n \log n)$ time by a divide and conquer algorithm [36].

Duality for the farthest-segment Voronoi diagram

Consider the farthest-segment Voronoi diagram of a set S of n line segments. Recall that the farthest-segment Voronoi diagram of S, denoted $\mathrm{FVD}(S)$, is a subdivision of the plane induced by the farthest Voronoi regions of segments, where the region of a segment $s_i \in S$ is $freg(s_i) = \{x \in \mathbb{R}^2 \mid d(x,s_i) > d(x,s_j), 1 \leq j \leq n, j \neq i\}$.

Aurenhammer et al. [8] pointed out that the point-line duality transformation T offers a correspondence between the faces of $\mathrm{FVD}(S)$ and envelopes of double wedges, which are derived from the input segments.

Let uv be a segment in S. We map uv to a double wedge that is the union of a *lower wedge* and an *upper wedge*, defined respectively as the area below and the area above both lines $T(u)$ and $T(v)$. Figures 2.3a,b show respectively a segment in the primal plane and its double wedge in the dual plane (shaded).

Since the transformation T reverses the above/below relationship, a line passing above the segment uv in the primal plane corresponds to a point in the lower wedge formed by $T(u)$ and $T(v)$ in the dual plane; and vice versa.

Let E be the boundary of the union of all the lower wedges of the segments in S (see Figure 2.5a). Let E' be defined symmetrically for the upper wedges. Then, the faces of $\mathrm{FVD}(S)$, which are unbounded in directions from 0 to π in cyclic order, correspond exactly to the edges of E in the order of increasing x-coordinate [8]. Symmetrically for the edges of E' and the Voronoi faces unbounded in directions

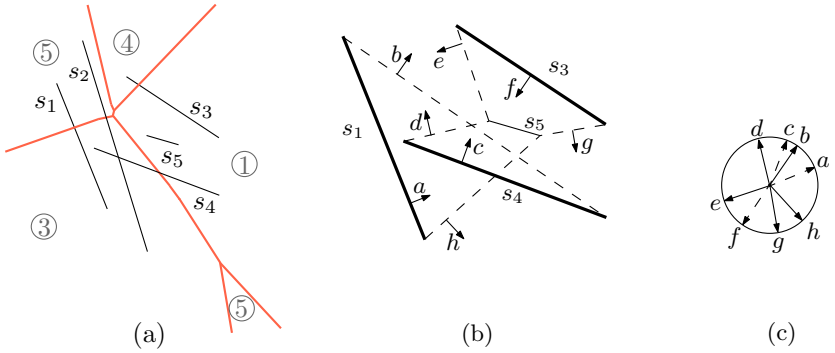


Figure 2.4. ([64]) (a) FVD(S), $S = \{s_1, \ldots, s_5\}$; (b) its farthest hull; (c) Gmap(S)

from π to 2π. We often refer to this dual setting in Chapter 4.

In addition, Aurenhammer et al. [8] observed the following.

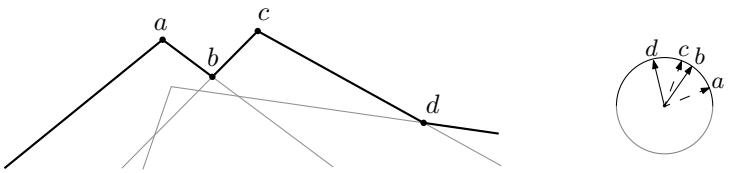
Observation 2.2.1 ([8])**.** *A farthest Voronoi region* $\mathrm{freg}(s_i)$ *of a segment* $s_i \in S$ *is non-empty and unbounded in some direction* ϕ *if and only if there exists an open halfplane, normal to* ϕ*, which intersects all segments in* S *but* s_i*.*

Another correspondence for the faces of the farthest-segment Voronoi diagram is provided by the *farthest hull* and its *Gaussian map*, introduced by Papadopoulou and Dey [64].

To define the farthest hull and the Gaussian map, we need to give the following notions. The line ℓ, normal to ϕ, bounding the halfplane of Observation 2.2.1, is called a *supporting line*. The direction ϕ (normal to ℓ) is referred to as the *hull direction* of ℓ and it is denoted by $\nu(\ell)$. An unbounded Voronoi edge towards infinity, separating $\mathrm{freg}(s_i)$ and $\mathrm{freg}(s_j)$, is a portion of $b(p,q)$, where p,q are endpoints of s_i and s_j, such that the line through $\overline{pq}$ induces an open halfplane that intersects all segments in S, except s_i, s_j (and possibly except additional segments incident to p,q). Segment $\overline{pq}$ is called a *supporting segment*; the direction normal to it and pointing inside this halfplane is denoted by $\nu(\overline{pq})$ and is called the *hull direction* of $\overline{pq}$. A segment $s_i \in S$ such that the line ℓ through s_i is supporting, is called a *hull segment*; its *hull direction* is $\nu(s_i) = \nu(\ell)$, normal to ℓ.

The *farthest hull* is the closed polygonal line obtained by following the supporting and hull segments in the angular order of their hull directions.

Figures 2.4a,b illustrate a farthest-segment Voronoi diagram and its farthest hull respectively. In Figure 2.4b, supporting segments are shown in dashed lines, and hull segments are shown in bold. Arrows indicate the hull directions of all supporting and hull segments.



(a) (b)

Figure 2.5. (a) The dual arrangement of lower wedges and the boundary of its union (black) (b) The upper Gmap of S from Figure 2.4

The *Gaussian map* of $\mathrm{FVD}(S)$, denoted $\mathrm{Gmap}(S)$ (see Figure 2.4c), provides a correspondence between the faces of $\mathrm{FVD}(S)$ and a *circle of directions* K [64]. K can be assumed to be a unit circle, where each point x on K corresponds to a direction as indicated by the radius of K at x. Each Voronoi face is mapped to an arc on K, which represents the set of directions along which the face is unbounded. An arc is delimited by consecutive hull directions of supporting segments. The $\mathrm{Gmap}(S)$ can be viewed as a cyclic sequence of consecutive arcs on K (in counterclockwise order), where each arc corresponds to one face of $\mathrm{FVD}(S)$. Two neighboring arcs α, γ are separated by the hull direction of a supporting segment $\overline{pq}$, denoted by $\nu(\alpha,\gamma) = \nu(\overline{pq})$. Direction $\nu(\alpha,\gamma)$ is the relevant direction towards infinity of bisector $b(p,q)$.

The arc of a hull segment s is called a *segment arc* and consists of two sub-arcs, one for each endpoint of s, separated by the hull direction of the segment, $\nu(s)$. An arc that corresponds to a single endpoint of a segment is called a *single-vertex* arc.

Figure 2.4c shows the $\mathrm{Gmap}(S)$ for the segment set S from Figure 2.4a; hull directions of the hull segments and of the supporting segments are shown as dashed and solid arrows respectively. Arc *bc* is a single-vertex arc, whereas arc *ce* is a segment arc.

Let the *upper* and *lower Gmap* denote the portion of $\mathrm{Gmap}(S)$ above and below the horizontal diameter of K respectively, i.e., in the upper and lower half-circle of K. There is a clear correspondence between E (resp., E') and the upper (resp., lower) Gmap: the vertices of E are exactly the hull directions of supporting segments on the upper Gmap and the apexes of wedges in E are exactly the hull directions of hull segments [64]; see Figure 2.5a,b.

Note that for both E and E', the (left-to-right) order of increasing x-coordinate corresponds to the counterclockwise order of increasing slope of the directions in the Gmap (i.e., in the upper and the lower Gmap respectively).

The $\mathrm{Gmap}(S)$ and the farthest hull of S can be computed in $O(n \log n)$ time (or in output-sensitive $O(n \log h)$ time, where h is the number of faces of $\mathrm{FVD}(S)$) by

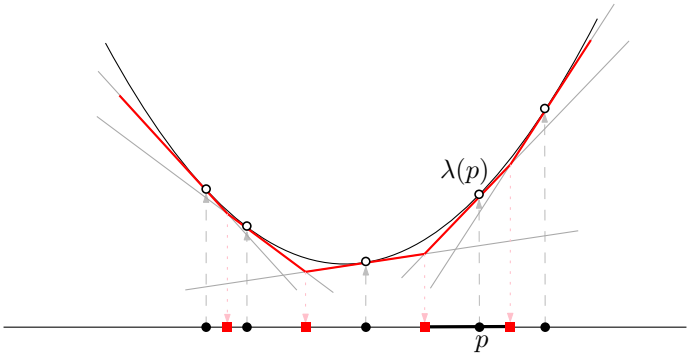


Figure 2.6. One-dimensional nearest-neighbor Voronoi diagram as a projection of the upper envelope of lines tangent to the unit parabola

adapting most of the algorithms for the convex hull of points in the plane (except for the *Graham scan* algorithm) [64].

2.2.2 Lifting transformation

The *lifting transformation* was introduced by Edelsbrunner and Seidel [40]. It connects Voronoi diagrams of points in $\mathbb{R}^d$ and arrangements of hyperplanes in $\mathbb{R}^{d+1}$. The transformation is discussed in detail in the book by Edelsbrunner [37].

For points in the plane, the lifting transformation is defined as follows. For a point p, let $\lambda(p)$ be a point on the unit paraboloid in $\mathbb{R}^3$ vertically above p. The lifting transformation, as applied to point p, returns the hyperplane tangent to the unit paraboloid at point $\lambda(p)$.

We need the following definitions. Given a set of hyperplanes, the *upper envelope* of their arrangement is the locus of all points q such that q lies on at least one of the hyperplanes, and a vertical ray drawn from q upwards does not intersect any hyperplane of the set. The *lower envelope* of an arrangement of hyperplanes is defined symmetrically for the rays pointing downwards.

Now we are ready to formulate the connection between the envelopes of the arrangement of hyperplanes and the Voronoi diagrams.

Property 2.2.1 ([40])**.** *Let S be a set of points in $\mathbb{R}^2$, and let the set $T(S)$ of hyperplanes be its image after the lifting transformation. The vertical projection of the upper envelope of $T(S)$ to the horizontal plane coincides with the nearest-neighbor Voronoi diagram of S. Respectively for the lower envelope of $T(S)$ and the farthest Voronoi diagram of S.*

The above property suggests that any algorithm to compute an envelope of hyperplanes may be used to compute the corresponding Voronoi diagram.

Figure 2.6 illustrates Property 2.2.1 for $\mathbb{R}^1$ (instead of $\mathbb{R}^2$, for simplicity). In particular, the sites are points on a horizontal line, indicated by the filled black disks. Unfilled disks indicate their counterparts on the unit parabola (see, e.g., points p and $\lambda(p)$); the tangent lines are shown in gray, and the upper envelop of their arrangement is shown in red lines. The vertices of the upper envelope, as projected back onto the horizontal line, are the vertices of the (one-dimensional) nearest-neighbor Voronoi diagram of the set of initial sites; the vertices are shown as red filled squares. The Voronoi region of point p is shown bold.

Connection to cluster Voronoi diagrams. Property 2.2.1 can be used to connect the Hausdorff Voronoi diagram of a family F of clusters of points in the plane with the arrangement of hyperplanes in three dimensions. For each cluster in F, take the lower envelope of hyperplanes corresponding to it, and then consider the upper envelope of these lower envelopes. The projection of the resulting upper envelope to the horizontal plane coincides with the $\mathrm{HVD}(F)$. Formally, the property is as follows.

Property 2.2.2 ([38])**.** *The Hausdorff Voronoi diagram of F is equivalent to the upper envelope of a family of lower envelopes of planes in $\mathbb{R}^3$, where each envelope corresponds to a cluster.*

A symmetric property holds for the farthest-color Voronoi diagram of F:

Property 2.2.3. *The farthest-color Voronoi diagram of F is equivalent to the lower envelope of a family of upper envelopes of planes in $\mathbb{R}^3$, where each envelope corresponds to a cluster.*

The above properties, for both the HVD and the FCVD, reduce the construction of these diagrams to the construction of the corresponding three-dimensional surfaces. Using a technique by Edelsbrunner et al. [38], the latter task can be accomplished in $O(n^2)$ time. For both diagrams, this implies an $O(n^2)$ time construction algorithm, which is worst-case optimal.

Summary. The material reviewed in this chapter is used in the following parts of this dissertation. The randomized incremental construction framework provides the base of our algorithms in Chapter 3. In particular, Section 3.2 builds upon the RIC variation that is based on point location, which is described in Section 2.1.2; whereas Sections 3.3 and 3.4 rely on the classic RIC framework described in Section 2.1.1. The point-line duality and the related transformations (Section 2.2.1) are used in Chapter 4. Chapter 5 exploits the lifting transformation (Section 2.2.2).

In fact, the stabbing circle problem in Chapter 5 is a direct generalization of the stabbing line problem that was discussed in Section 2.2.1.

Chapter 3

Randomized incremental construction of the Hausdorff Voronoi diagram

This chapter presents randomized incremental algorithms to compute the Hausdorff Voronoi diagram.

Section 3.2 is based on:

P. Cheilaris, E. Khramtcova, S. Langerman, and E. Papadopoulou. A randomized incremental algorithm for the Hausdorff Voronoi diagram of non-crossing clusters. Algorithmica*, 2016. DOI 10.1007/s00453-016-0118-y.*

Sections 3.3 and 3.4 are based on:

E. Khramtcova, and E. Papadopoulou Randomized Incremental Construction for the Hausdorff Voronoi Diagram of Point Clusters. In preparation. Preliminary version is submitted to a conference.

We investigate two variants of the randomized incremental framework: the classic RIC, and the RIC based on point location, as applied to the Hausdorff Voronoi diagram. We consider separately the cases of non-crossing and arbitrary input clusters. After giving definitions and properties of the diagram used in all the algorithms (Section 3.1), we proceed with our results. In Section 3.2 we present a RIC based on point location for non-crossing clusters. Then we apply the classic RIC framework to the Hausdorff Voronoi diagram, which results in two algorithms. Section 3.3 presents an algorithm for non-crossing clusters, and Section 3.4 presents one for arbitrary clusters that are allowed to cross.

3.1 Preliminaries and Definitions

Let F be a family of k clusters of points in the plane such that no two clusters have a common point, and let $n = |\cup F|$. Unless stated otherwise, we use C to denote a cluster in F, and c to denote a point within C. Let $\operatorname{conv} C$ denote the convex hull of C. Let $d(\cdot,\cdot)$ denote the Euclidean distance between two points in $\mathbb{R}^2$.

For simplicity of presentation, we follow a general position assumption that no four points lie on the same circle. This assumption can be removed similarly to an ordinary Voronoi diagram of points, e.g., following techniques of symbolic perturbation [74].

The *farthest Voronoi diagram* of C, in brief $\mathrm{FVD}(C)$, is the farthest-site Voronoi diagram of its points (see Definition 3). It partitions the plane into regions such that the *farthest Voronoi region* of a point $c \in C$ is:

$$\mathrm{freg}_C(c) = \{t \mid \forall c' \in C \setminus \{c\} \colon d(t,c) > d(t,c')\}.$$

Let $\mathscr{T}(C) = \mathbb{R}^2 \setminus \bigcup_{c \in C} \mathrm{freg}_C(c)$, if $|C| > 1$; and let $\mathscr{T}(C) = c$, if $C = \{c\}$. For $|C| > 1$, $\mathscr{T}(C)$ is the graph structure of $\mathrm{FVD}(C)$, which is well known to be a tree. It is also well known that $\mathrm{freg}_C(c) \neq \emptyset$ if and only if c is a vertex of $\operatorname{conv} C$. We assume that the tree $\mathscr{T}(C)$ is rooted at a point at infinity along an unbounded Voronoi edge.

For a point $t \in \mathbb{R}$ and a cluster $P \in F$, let the *farthest distance* between t and P be defined as

$$\mathrm{d_f}(t,P) = \max_{p \in P} d(t,p).$$

Using the notion of the farthest distance, the Hausdorff Voronoi diagram of F is defined as follows (see also Definition 4 in Section 1.2.1).

The *Hausdorff Voronoi region* of a cluster $C \in F$ is defined as:

$$\mathrm{hreg}_F(C) = \{p \mid \forall C' \in F \setminus \{C\} \colon \mathrm{d_f}(p,C) < \mathrm{d_f}(p,C')\}.$$

The Hausdorff Voronoi region of a point $c \in C$ is defined as:

$$\mathrm{hreg}_F(c) = \mathrm{hreg}_F(C) \cap \mathrm{freg}_C(c).$$

The partitioning of the plane into Hausdorff Voronoi regions is called the *Hausdorff Voronoi diagram* of F, for brevity $\mathrm{HVD}(F)$, or simply HVD.

By the definition of a Hausdorff Voronoi region, $\mathrm{hreg}_F(c) = \emptyset$ for any point $c \in C$ that is not a vertex of $\operatorname{conv} C$. Since such points are not relevant to the Hausdorff diagram, we assume throughout this chapter that all points in C are also vertices of $\operatorname{conv} C$.

For two clusters $C, P \in F$, their Hausdorff bisector is:

$$\mathrm{b_h}(C,P) = \{y \mid \mathrm{d_f}(y,C) = \mathrm{d_f}(y,P)\}.$$

The Hausdorff bisector $\mathrm{b_h}(C,P)$ is a subgraph of $\mathscr{T}(C \cup P)$. It consists of one (if P, Q are non-crossing) or more (if P, Q are crossing) unbounded polygonal chains. Each vertex of $\mathrm{b_h}(C,P)$ is the center of a circle passing through two points of one cluster and one point of another that entirely encloses P and Q. These vertices are called *mixed*.

Definition 7. A vertex on the bisector $\mathrm{b_h}(C,P)$, induced by two points $c_i, c_j \in C$ and a point $p_l \in P$ is called *crossing*, if there is a diagonal $p_l p_r$ of P that crosses the diagonal $c_i c_j$ of C, and all points c_i, c_j, p_l, p_r are on $\mathrm{conv}\, C \cup P$. The total number of crossing vertices along the bisectors of all pairs of clusters is the *number of crossings* and is denoted by m.

The structure of a face of a Hausdorff Voronoi region is illustrated in Figure 3.1. For a point $c \in C$, the boundary of a face of $\mathrm{hreg}_F(c)$ consists always of two chains: (1) the *farthest boundary*, which is internal to $\mathrm{hreg}_F(C)$ (i.e., $\partial\mathrm{hreg}_F(c) \cap \partial\mathrm{freg}_C(c) \subseteq \mathscr{T}(C)$); and (2) the *Hausdorff boundary* (i.e., $\partial\mathrm{hreg}_F(c) \cap \partial\mathrm{hreg}_F(C)$). Neither chain can be empty, if $|C| > 1$ [66].

There are three types of vertices in a Hausdorff Voronoi diagram (see Figure 3.1) [62]:

(1) *Pure* Voronoi vertices, which are equidistant to three clusters and appear on Hausdorff boundaries;

(2) *Mixed* Voronoi vertices, which are equidistant to three points of two clusters;

(3) *Farthest* Voronoi vertices of $\mathscr{T}(C)$, which appear only on farthest boundaries.

Mixed Voronoi vertices are the points incident to a Hausdorff and a farthest boundary. Mixed vertices that are induced by two points in cluster C and one point in another cluster are called *C-mixed* vertices. The farthest boundary of $\mathrm{hreg}_F(c)$ meets its Hausdorff boundary at a *C-mixed* vertex, or it extends to infinity, if $\mathrm{hreg}_F(c)$ is unbounded.

Hausdorff Voronoi edges are polygonal lines that connect pure Voronoi vertices and separate the Voronoi regions of different clusters, see the solid lines in Figure 1.3. Mixed Voronoi vertices correspond to the breakpoints of these polygonal lines.

A line segment connecting two points in C is called a *chord* of C. The closure of a Voronoi region is denoted as $\overline{reg}(\cdot)$.

The following definition is illustrated in Figure 3.2.

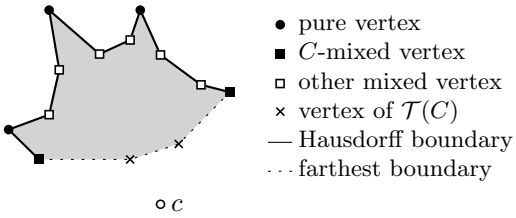


Figure 3.1. Features of $\partial \mathrm{hreg}_F(c)$

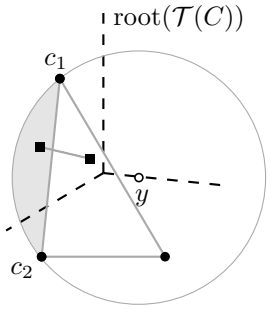


Figure 3.2. $\mathscr{D}_y$ is partitioned by $\overline{c_1c_2}$ in $\mathscr{D}_y^r$ (shaded) and $\mathscr{D}_y^f$. The 2-point cluster P, illustrated in squares, is rear limiting w.r.t. the 3-point cluster C.

Definition 8 (Rear/forward limiting cluster [66])**.**

- Let y be a point on an edge e of $\mathscr{T}(C)$ induced by $c_1, c_2 \in C$, i.e., $\mathrm{d_f}(y,C) = d(y,c_1) = d(y,c_2)$. Point y partitions $\mathscr{T}(C)$ into two parts: $\mathscr{T}_y^r$ and $\mathscr{T}_y^f$, where $\mathscr{T}_y^r$ is the subtree rooted at y as we traverse $\mathscr{T}(C)$ starting at its root; $\mathscr{T}_y^f$ is the complement of $\mathscr{T}_y^r$.
- Let $\mathscr{D}_y$ be the disk centered at y of radius $\mathrm{d_f}(y,C)$. Chord $\overline{c_1c_2}$ partitions $\mathscr{D}_y$ in two parts: $\mathscr{D}_y^r$ and $\mathscr{D}_y^f$, where $\mathscr{D}_y^r$ (shown shaded in Figure 3.2) is the portion that contains the points of C that induce $\mathscr{T}_y^r$.
- A cluster P enclosed in $\mathscr{D}_y^r \cup \mathrm{conv}\, C$ (resp., in $\mathscr{D}_y^f \cup \mathrm{conv}\, C$) is called *rear* (resp., *forward*) limiting for C with respect to y.

Below we list some useful properties of the Hausdorff Voronoi diagram.
For the next three properties, assume all clusters in F to be non-crossing.

Property 3.1.1 ([66])**.** *If cluster C has a rear (resp., forward) limiting cluster P with respect to y in $\mathscr{T}(C)$, then the entire $\mathscr{T}_y^r$ (resp., $\mathscr{T}_y^f$) is closer to P than to C.*

Property 3.1.1 implies also Properties 3.1.2 and 3.1.3.

Property 3.1.2. *Let $C, P \in F$.*

(a) Region $\mathrm{hreg}_F(C)$ contains exactly one connected component of $\mathscr{T}(C)$, unless $\mathrm{hreg}_F(C) = \emptyset$.

(b) Let v be a vertex in $\mathscr{T}(C)$. If $\mathrm{d_f}(v,P) < \mathrm{d_f}(v,C)$, then only one of the subtrees incident to v may intersect $\mathrm{hreg}_F(C)$.

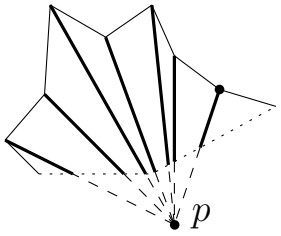


Figure 3.3. Visibility-based decomposition of a face of $\mathrm{hreg}_F(p)$

(c) Let $e = uv$ be an edge in $\mathscr{T}(C)$. If both u and v are closer to P than to C, then $\mathrm{hreg}_F(C) \cap e = \emptyset$.

Property 3.1.3 ([66])**.** *Region* $\mathrm{hreg}_F(C) = \emptyset$ *if and only if one of the following conditions holds: (1) there is a cluster in F entirely contained in* $\mathrm{conv}\,C$*; (2) there is a pair of clusters in F such that one is forward limiting and the other is rear limiting with respect to the same point $y \in \mathscr{T}(C)$.*

The cluster or pair of clusters of Property 3.1.3 is called a *killer* or a *killing pair* of C respectively.

Property 3.1.2a can be generalized for the case of arbitrary clusters, that are allowed to cross:

Property 3.1.4 ([62])**.** *Each face of a (non-empty) region* $\mathrm{hreg}_F(C)$ *intersects $\mathscr{T}(C)$ in one non-empty connected component. This component is delimited by C-mixed vertices.*

Property 3.1.5 ([66])**.** *For any point $x \in \mathrm{hreg}_F(c)$ the line segment $cx \cap \mathrm{freg}_C(c)$ lies entirely in* $\mathrm{hreg}_F(c)$.

Property 3.1.5 implies that the Hausdorff Voronoi diagram can be refined to simpler regions by the *visibility-based decomposition*, without increasing combinatorial complexity of the diagram. The decomposition is defined as follows.

Visibility-based decomposition of the Hausdorff Voronoi diagram [66]. Consider each region $\mathrm{hreg}_F(p)$ of $\mathrm{HVD}(F)$. For each vertex v on its boundary, draw the line segment pv, as restricted within $\mathrm{hreg}_F(p)$, see Figure 3.3.

Each face f of the visibility-based decomposition of $\mathrm{hreg}_F(p)$ is convex, and its boundary consists of three parts:(1) a chain that is portion of $\mathscr{T}(P)$, shown in dotted lines in Figure 3.3; (2) a segment of $\mathrm{b_h}(P,Q), Q \in F$, shown solid in Figure 3.3; (3) at most two edges of the visibility-based decomposition, shown bold in Figure 3.3.

Observation 3.1.1. *A face f of the visibility-based decomposition of* $\mathrm{hreg}_F(p)$*, $p \in P$, borders the Hausdorff Voronoi regions of one, two or three clusters in $F \setminus \{P\}$.*

3.2 A RIC for the Hausdorff Voronoi diagram of non-crossing clusters based on point location

Throughout this section we assume that all clusters in F are pairwise non-crossing.

Let $C_1,\ldots,C_k$ be a random permutation of the input family F of clusters. Let $F_i = \{C_1,\ldots,C_i\}$, $1 \le i \le k$, be the set of the first i clusters in the given permutation. Our algorithm incrementally computes $\mathrm{HVD}(F_i)$, for $1 < i \le k$, starting with $\mathrm{HVD}(F_1) = \mathrm{FVD}(C_1)$. At step i, we insert cluster C_i in $\mathrm{HVD}(F_{i-1})$ and derive $\mathrm{HVD}(F_i)$. To this goal, we identify a *representative* point $t \in \mathrm{hreg}_{F_i}(C_i)$ or we determine that no such point exists. If t exists, we trace the boundary of $\mathrm{hreg}_{F_i}(C_i)$, and update the diagram to $\mathrm{HVD}(F_i)$. Else, we conclude that $\mathrm{hreg}_{F_i}(C_i) = \emptyset$ and $\mathrm{HVD}(F_i) = \mathrm{HVD}(F_{i-1})$.

The main challenge of our algorithm is to efficiently identify a representative point t or conclude that no such point exists. Then, the tracing of $\mathrm{hreg}_{F_i}(C_i)$ can be done similarly to [66], in time proportional to the complexity of the new region, plus the complexity of the deleted portion of the diagram, times $O(\log n)$, see Section 3.2.3. In the remaining of this section we focus on identifying a representative point t. We skip the subscript "F_i" and let $\mathrm{hreg}(C_i)$ stand for $\mathrm{hreg}_{F_i}(C_i)$.

Property 3.1.2a implies three possibilities for $\mathrm{hreg}(C_i)$: (1) $\mathrm{hreg}(C_i) \cap \mathscr{T}(C_i)$ contains a vertex of $\mathscr{T}(C_i)$; (2) $\mathrm{hreg}(C_i)$ intersects exactly one edge of $\mathscr{T}(C_i)$; and (3) $\mathrm{hreg}(C_i)$ is empty. The edge of case 2 is called a *candidate edge*, see Definition 9. In Figure 1.3, the bounded region in the middle illustrates case 2, while the three unbounded regions illustrate case 1.

To identify case (1), it is enough to perform point location of the vertices in $\mathscr{T}(C_i)$ in $\mathrm{HVD}(F_{i-1})$. If any vertex v is found to be closer to C_i than to its owner in $\mathrm{HVD}(F_{i-1})$, then v can serve as a representative point, i.e., $t = v$. Suppose that no vertex of $\mathscr{T}(C_i)$ satisfies case (1). Then we look for a *candidate edge* that may satisfy case (2).

Definition 9 (candidate edge)**.** Let uv be an edge of $\mathscr{T}(C_i)$ and let Q^u, Q^v be the clusters in F_{i-1} closest to u and v respectively. Edge uv is called a *candidate edge* if $Q^u \neq Q^v$ and uv satisfies the following predicate: $\mathrm{cand}(uv) = (\mathrm{d_f}(u,Q^u) < \mathrm{d_f}(u,C_i) < \mathrm{d_f}(u,Q^v)) \wedge (\mathrm{d_f}(v,Q^v) < \mathrm{d_f}(v,C_i) < \mathrm{d_f}(v,Q^u))$.

Lemma 3.2.1. *If there is an edge uv of $\mathscr{T}(C_i)$ that is a candidate edge, then either $\mathrm{hreg}(C_i)$ intersects uv or $\mathrm{hreg}(C_i) = \emptyset$.*

Proof. Consider the clusters Q^u and Q^v of Definition 9. By Property 3.1.2b, only one of the subtrees of u can be closer to C_i than to Q^u. Because uv is a candidate edge satisfying Definition 9, v lies in this subtree. That is, only the subtree of

u that contains uv may intersect $\mathrm{hreg}(C_i)$. Symmetrically, only the subtree of v that contains uv may intersect $\mathrm{hreg}(C_i)$. The intersection of these two subtrees is exactly the edge uv. Thus, $\mathscr{T}(C_i) \cap \mathrm{hreg}(C_i) \subset uv$, or $\mathscr{T}(C_i) \cap \mathrm{hreg}(C_i) = \emptyset$. In the latter case, by Property 3.1.2a, $\mathrm{hreg}(C_i) = \emptyset$. □

Lemma 3.2.1 implies that, given a candidate edge uv, it suffices to search on uv to identify a representative point. Furthermore, if such a point cannot be found on uv, then $\mathrm{hreg}(C_i) = \emptyset$. We can perform the search for a representative point as follows: Traverse $\mathscr{T}(C_i)$, starting at its root, checking vertices and possibly pruning appropriate subtrees according to Properties 3.1.2b and 3.1.2c. During the traversal, either determine t as a vertex of $\mathscr{T}(C_i)$, or determine a candidate edge uv, or conclude that $\mathrm{hreg}(C_i) = \emptyset$.

When a candidate edge uv is determined, we still need to identify a representative point t on uv or determine that $\mathrm{hreg}(C_i) = \emptyset$. This is achieved by performing a *parametric point location query* in $\mathrm{HVD}(F_{i-1})$ for edge uv as given in the following definition.

Definition 10 (Parametric point location query)**.** Given a family of clusters F, $\mathrm{HVD}(F)$, a cluster $C \notin F$, and a candidate edge $uv \subset \mathscr{T}(C)$, determine a cluster $P \in F$ and a point $t \in uv$ such that $t \in \mathrm{hreg}_F(P)$ and $\mathrm{d_f}(t,P) = \mathrm{d_f}(t,C)$. If such a point does not exist, return *nil*.

The performance of the parametric point location query sets the time complexity of our algorithm. To perform parametric as well as ordinary point location efficiently, we store $\mathrm{HVD}(F)$ in a hierarchical data structure, called the Voronoi hierarchy. This data structure is described in Section 3.2.2 and the parametric point location query is detailed in Section 3.2.2. The parametric point location requires to also answer an additional non-standard location query on the static farthest Voronoi diagram of a given cluster, called a *segment query*. A data structure to efficiently answer segment queries is given in the Section 3.2.1.

3.2.1 Centroid Decomposition

This section describes a data structure, called the *centroid decomposition*, that can efficiently answer queries related to point location on a planar subdivision induced by a tree structure. The centroid decomposition was introduced by Megiddo et al. [56], and we use it in this chapter to efficiently perform *segment queries* on the farthest Voronoi diagram of a cluster. Segment queries are used during parametric point location in the Hausdorff Voronoi diagram. The query is defined as follows.

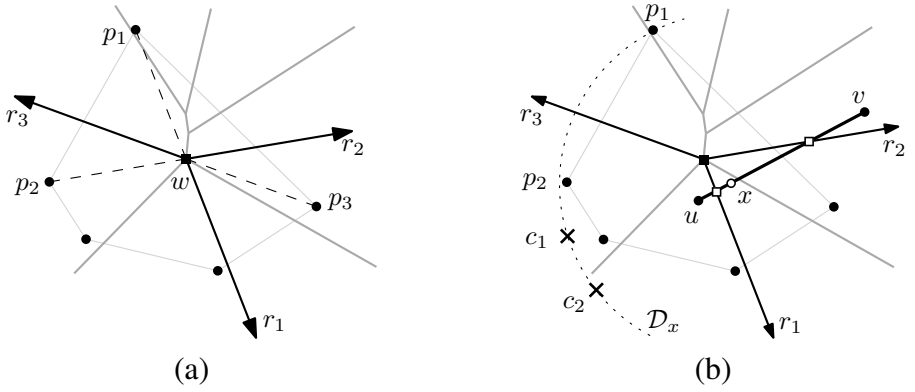


Figure 3.4. A cluster P (disks), $\mathrm{FVD}(P)$ (thick grey lines) and (a) the rays r_i corresponding to vertex w; (b) the segment query for uv outputs point x

Definition 11 (Segment query)**.** Consider two clusters C, P. Given a segment $uv \subset \mathscr{T}(C)$ such that $\mathrm{d_f}(u,C) < \mathrm{d_f}(u,P)$ and $\mathrm{d_f}(v,C) > \mathrm{d_f}(v,P)$, find the point $x \in uv$ that is equidistant from C and P, i.e., $\mathrm{d_f}(x,C) = \mathrm{d_f}(x,P)$. See Figure 3.4b.

Let P be a cluster of points and let $\mathrm{CD}(P)$ denote the centroid decomposition for $\mathrm{FVD}(P)$. We first define $\mathrm{CD}(P)$, and then we describe how to use it to perform an ordinary point location query on $\mathrm{FVD}(P)$. Finally, we adapt the point location query to perform a segment query on $\mathrm{FVD}(P)$.

Definition of $\mathrm{CD}(P)$. Any tree with h vertices has a vertex v, called the *centroid*, whose removal decomposes the tree into subtrees of at most $h/2$ vertices each [56]. Exploiting this fact, we build $\mathrm{CD}(P)$ as a balanced tree whose nodes correspond to Voronoi vertices of $\mathrm{FVD}(P)$ as follows:

- Find the centroid w of $\mathscr{T}(P)$. Create a node for w and assign it as the root of $\mathrm{CD}(P)$.
- Remove w from $\mathscr{T}(P)$. Recursively build the centroid decomposition trees for the connected components of $\mathscr{T}(P)$ incident to w and link them as subtrees of w.

Point location on $\mathrm{FVD}(P)$ using $\mathrm{CD}(P)$. Given a query point q, we perform point location as follows. Starting from the root, we traverse $\mathrm{CD}(P)$, testing q against the current node of $\mathrm{CD}(P)$ (we explain the procedure of such test in the next paragraph). Every time, we choose one of the node's subtrees to continue, until a leaf node is reached. Among the three points in P inducing the Voronoi

vertex of the leaf node, we choose the one farthest from q. Then point q belongs in the farthest Voronoi region of the chosen point.

Testing q against a node α is performed following Aronov et al. [6]. In particular, let w be the Voronoi vertex of $\mathrm{FVD}(P)$, corresponding to node α of $\mathrm{CD}(P)$. Let $p_1, p_2, p_3 \in P$ have farthest Voronoi regions incident to w. Consider the rays $r_i, i = 1,2,3$, originating at w and having direction $\overrightarrow{p_i w}$ respectively; see Figure 3.4a. The rays r_i subdivide the plane into three sectors. Among the subtrees of $\mathrm{CD}(P)$ incident to α, pick the one that corresponds to the sector containing q. The correctness of this procedure is implied by the construction of $\mathrm{CD}(P)$ and the following lemma.

Lemma 3.2.2. *Rays $r_i, i = 1,2,3$, subdivide the plane into three sectors, where each sector contains exactly one connected component of $\mathscr{T}(P) \setminus \{w\}$.*

Proof. It is well known that for any point $t \in \mathrm{freg}_P(p)$ ($p \in P$), the ray originating at t and having direction $\overrightarrow{pt}$ is entirely contained in $\mathrm{freg}_P(p)$. Thus, no ray r_i can intersect the edges of $\mathscr{T}(P)$. Since the rays r_i lie in three distinct regions of $\mathrm{FVD}(P)$, there is a component of $\mathscr{T}(P) \setminus \{w\}$ in each of the three sectors formed by these three rays. The claim follows. □

Segment query on $\mathrm{FVD}(P)$ using $\mathrm{CD}(P)$. The segment query can be performed similarly to a point location query within the same time complexity. The difference is in the testing of segment uv against a node α of $\mathrm{CD}(P)$. Let w be the Voronoi vertex of $\mathrm{FVD}(P)$ that corresponds to α. Let rays $r_i, i = 1,2,3$, emanate from w as defined above; see Figure 3.4b.

Consider the (at most two) intersection points of the rays with uv. If any of these points is equidistant to C and P, return it. Otherwise, these intersection points break uv into (at most three) subsegments, each lying in one of the three sectors formed by the rays r_i. Among these subsegments, pick subsegment $u'v'$ such that $\mathrm{d_f}(u', C) < \mathrm{d_f}(u', P)$ and $\mathrm{d_f}(v', C) > \mathrm{d_f}(v', P)$, and continue with the child of α in $\mathrm{CD}(P)$ whose Voronoi vertex lies in the same sector as $u'v'$.

If α is a leaf of $\mathrm{CD}(P)$, let e be the edge of $\mathscr{T}(P)$ incident to the vertex w that lies in the same sector as $u'v'$. Let p_1, p_2 be the points in P that induce the edge e, and let c_1, c_2 be the points in C that induce uv. Since $\mathrm{d_f}(v, P) < \mathrm{d_f}(v, C) = \mathrm{d_f}(v, c_1) = \mathrm{d_f}(v, c_2)$, the closed disk centered at v and passing through c_1, c_2 must contain both p_1 and p_2. Since C and P are non-crossing, both p_1 and p_2 lie to the same side of the chord $\overline{c_1 c_2}$. Thus, one of the two closed disks defined by points p_1, c_1, c_2 or by points p_1, c_1, c_2 must contain both p_1 and p_2. Return as an answer to the segment query the center of this disk. In Figure 3.4b, such a disk $\mathscr{D}_x$ has a dotted arc on its boundary and its center x is shown as an unfilled circle.

Lemma 3.2.3. *The centroid decomposition* $\mathrm{CD}(P)$ *of a cluster* P *can be built in* $O(n_p \log n_p)$ *time, where* n_p *is the number of vertices of* $\mathrm{FVD}(P)$. *Both the point location and the segment query in* $\mathrm{CD}(P)$ *require* $O(\log n_p)$ *time.*

Proof. Given a subtree of $\mathscr{T}(P)$, its centroid can be computed in $O(h)$ time [56], where h is the number of vertices in this subtree. Building $\mathrm{CD}(P)$ requires to recursively compute the centroids of its subtrees, each of size at most half the size of P. This implies an $O(n_p \log n_p)$ total time to build $\mathrm{CD}(P)$.

The point location query consists of $O(\log n_p)$ tests of a query point against a node of $\mathrm{CD}(P)$. Each test involves a constant number of points and rays, thus, it can be performed in constant time. The same argument works for the test of a segment against a node of $\mathrm{CD}(P)$ during a segment query, which implies the same $O(\log n_p)$ time bound. □

3.2.2 The Voronoi Hierarchy for the Hausdorff Voronoi Diagram

We describe a randomized semi-dynamic data structure to store the Hausdorff Voronoi diagram that supports insertion of a cluster, and point location queries (ordinary and parametric). It augments the Voronoi hierarchy [16, 48] with the ability to handle the generalized Voronoi features present in the Hausdorff diagram. These are sites of non-constant complexity, sites that are not entirely contained in their regions, and empty Voronoi regions. We refer to our adaptation as the *Hausdorff Voronoi hierarchy*. For details about the Voronoi hierarchy, refer to Section 2.1.2. It is formally defined as follows.

Definition 12 ([48, 35])**.** The *Voronoi hierarchy* of a set of sites S is a sequence of Voronoi diagrams $\mathrm{VD}(S^{(\ell)}), \ell = 0, \ldots, h$, where the sets $S^{(\ell)}$ form a hierarchy of subsets of S, built as follows. $S^{(0)} = S$, and for $\ell = 1, \ldots, h$, $S^{(\ell)}$ is a random sample of $S^{(\ell-1)}$ following a Bernoulli distribution with a fixed constant parameter $\beta \in (0,1)$. We refer to ℓ as "level of the Voronoi hierarchy".

To perform *point location* for a query point q in the Voronoi hierarchy, we start at the last level h, and for each level ℓ, we determine the site $s^\ell \in S^{(\ell)}$ that is closest to q by performing a *walk*. Each step of the walk moves from the current site to one of its neighbors such that the distance to q is reduced. To determine an appropriate neighbor, binary search may be used [48]. A walk at level ℓ starts at $s^{\ell+1}$. The answer to the query is s^0.

For the Hausdorff Voronoi diagram, a first difference to consider is that clusters are not of constant complexity and n can be $\omega(k)$. Recall that k is the number

of sites (clusters) and n is their total complexity ($k \leq n$). Nevertheless, the complexity of the Hausdorff Voronoi hierarchy is expected $O(n)$ as for the original hierarchy [35, 48].

Lemma 3.2.4. *Consider the Voronoi diagram of a family of k sites of total complexity n, where the size of the diagram is also $O(n)$. Then the Voronoi hierarchy for such diagram has expected size $O(n)$ and expected number of levels $O(\log k)$.*

Proof. Let $\|S^{(\ell)}\|$ denote the total complexity of the sites at a level $S^{(\ell)}$. For any site $s \in S$, the probability that s appears in $S^{(\ell)}$ is β^ℓ. Then the expectation of $\|S^{(\ell)}\|$ is $\mathbb{E}[\|S^{(\ell)}\|] = \beta^\ell \|S\| = \beta^\ell n$, and the expected size of the Voronoi diagram at level ℓ is $O(\beta^\ell n)$. The expected size of the hierarchy is

$$\sum_{\ell=0}^{\infty} O(\mathbb{E}[\|S^{(\ell)}\|]) = \sum_{\ell=0}^{\infty} O(\beta^\ell n) = \frac{1}{1-\beta} O(n) = O(n).$$

The bound on the expected number of levels follows immediately from properties of the Bernoulli distribution [35, 48]. □

To adapt the Voronoi hierarchy for the Hausdorff Voronoi diagram, several difficulties have to be addressed. When performing a walk at a level ℓ of the hierarchy, at each step we need to reduce the distance between the current cluster C and the query point q. However, the farthest distance $\mathrm{d_f}(q, C)$ may be realized by a point $c \in C$ that has an empty Voronoi region at level ℓ. Thus, instead of $\mathrm{d_f}(\cdot,\cdot)$, we base the walk on a slightly different distance function, which reflects the diagram better and which equals $\mathrm{d_f}(\cdot,\cdot)$ at the end of the walk. In addition, the neighbors of a Hausdorff Voronoi region do not have a natural ordering, and thus, it is not easy to use binary search when performing one step in the walk. To address these problems, we first redesign one step of the walk. Then, point location can be performed as in the ordinary hierarchy. Further, we describe the parametric point location query that is needed for our algorithm. Empty Voronoi regions in the Hausdorff diagram pose another major difficulty when updating the hierarchy because they complicate the transition between the hierarchy levels. Finally, we show how to maintain the hierarchy and deal with regions that become empty.

In our modified hierarchy that handles the above difficulties, a walk at level ℓ does not necessarily start from the same cluster where it stopped at level $\ell + 1$, but possibly from another cluster that is closer to q. The following lemma shows that the expected length of the walk on one level of the Hausdorff Voronoi hierarchy is constant. It is a simple modification of [48, Lemma 9].

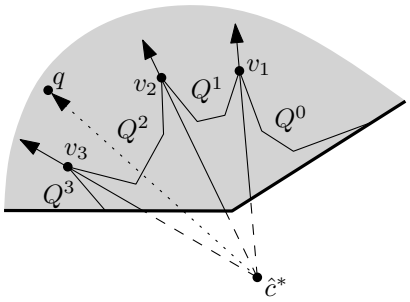


Figure 3.5. One step of a walk for a query point q and a starting cluster C

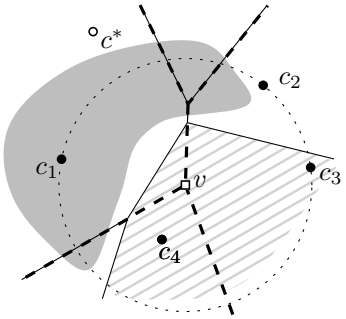


Figure 3.6. Illustration of the proof of Lemma 3.2.6

Lemma 3.2.5. *Let $s_0^\ell, \ldots, s_r^\ell = s^\ell$ be the sequence of sites visited at level ℓ during point location for a query point q. Assuming that $\mathrm{d_f}(q, s_i^\ell) < \mathrm{d_f}(q, s_{i-1}^\ell)$, $i = 1, \ldots, r$, and either $s^{\ell+1} = s_0^\ell$, or $\mathrm{d_f}(q, s_0^\ell) < \mathrm{d_f}(q, s^{\ell+1})$, the expectation of the length r of the walk at level ℓ is constant.*

Proof. Each site visited by the walk at level ℓ is closer to q than $s^{\ell+1}$, and thus does not belong to level $\ell+1$. The probability that there are t such sites is $\beta(1-\beta)^{t-1}$. Thus, the expected number of sites visited at level ℓ is at most

$$\sum_{t=1}^{|S^{(\ell)}|} t(1-\beta)^{t-1}\beta < \beta \sum_{t=1}^{\infty} t(1-\beta)^{t-1} = \frac{1}{\beta}.$$

□

One step of the walk

Let $\ell \in \{0, \ldots, h\}$ be a level in the Hausdorff Voronoi hierarchy of F. Let $\mathrm{hreg}_F^{(\ell)}(\cdot)$ denote $\mathrm{hreg}_{F^{(\ell)}}(\cdot)$ and let $\overline{\mathrm{hreg}}_{F^{(\ell)}}(\cdot)$ denote the closure of this region.

A point $c \in C$ is called *active* if $\mathrm{hreg}_F^{(\ell)}(c) \neq \emptyset$. Let $\hat{C}$ denote the set of all active points in a cluster C. The walk to locate a query point q uses the farthest distance to the active points of a cluster C as opposed to the farthest distance to all points of C. One step of the walk is defined as follows.

Definition 13 (a step of the walk at level ℓ)**.** Given a query point q and a cluster $C \in F^{(\ell)}$ such that $q \notin \overline{\mathrm{hreg}}_{F^{(\ell)}}(C)$, determine $Q \in F^{(\ell)}$ such that $\mathrm{hreg}_F^{(\ell)}(Q)$ is adjacent to $\mathrm{hreg}_F^{(\ell)}(C)$ and $\mathrm{d_f}(q, \hat{Q}) < \mathrm{d_f}(q, \hat{C})$. If $q \in \overline{\mathrm{hreg}}_{F^{(\ell)}}(C)$ then $Q = C$.

To perform one step of the walk we use the set of active points $\hat{C}$. We store $\hat{C}$ as a circular list of its points in the order of its convex hull. Each point $c \in \hat{C}$ has a link

Algorithm *Step-ℓ*
1. Determine c^* by locating q in $\mathrm{FVD}(C)$;
2. **if** $c^* \in \hat{C}$ **then**
3. Let $\hat{c}^* = c^*$;
4. **else**
5. Let $\hat{c}^*$ be the point in $\{c_1, c_2\}$ that is the farthest from q (See Lemma 3.2.6);
6. Let $Q = Q^i$ such that ray $\overrightarrow{\hat{c}^* q}$ follows $\overrightarrow{\hat{c}^* v_i}$ and/or precedes $\overrightarrow{\hat{c}^* v_{i+1}}$ (See Figure 3.5);
7. **if** $\mathrm{d_f}(q, \hat{Q}) < \mathrm{d_f}(q, \hat{C})$ **then**
8. **return** Q
9. **else return** C

Figure 3.7. An algorithm to perform a step of the walk at level ℓ for a query point q, starting from cluster C

to the ordered list of pure Voronoi vertices $v_1, \ldots, v_j$ on the boundary of $\mathrm{hreg}_F^{(\ell)}(c)$. (Recall from Section 3.1 that pure Voronoi vertices are equidistant to three different clusters.) Let $Q^0, \ldots, Q^{j+1}$ be the corresponding list of clusters whose Voronoi regions are adjacent to $\mathrm{hreg}_F^{(\ell)}(c)$, see Figure 3.5. We determine cluster Q by binary search in these lists. The detailed algorithm is given in Algorithm *Step-ℓ*, see Figure 3.7.

For the rest of this section, let c^* (resp., $\hat{c}^*$) be the point in C (resp., in $\hat{C}$) that is farthest from q, i.e., $\mathrm{d_f}(q, c^*) = \max_{c \in C} \mathrm{d_f}(q, c)$ and $\mathrm{d_f}(q, \hat{c}^*) = \max_{c \in \hat{C}} \mathrm{d_f}(q, c)$. Let $c_1, c_2 \in \hat{C}$ be the active points immediately following and preceding c^* respectively on the boundary of $\mathrm{conv}\, C$. In the following, we use c_1 and c_2 to determine point $\hat{c}^*$.

To establish the correctness of Algorithm *Step-ℓ* (Figure 3.7) we need to prove correctness for Lines 5 and 6–9. The following lemma shows that $\hat{c}^* \in \{c_1, c_2\}$ (if c^* is not active), and thus, it establishes the correctness of Line 5.

Lemma 3.2.6. *Assuming $c^* \notin \hat{C}$, $\mathrm{freg}_C(c^*) \subset \mathrm{freg}_{\hat{C}}(c_1) \cup \mathrm{freg}_{\hat{C}}(c_2)$.*

Proof. Let $C' = \hat{C} \cup \{c^*\}$. Since $C' \subseteq C$, $\mathrm{freg}_C(c^*) \subseteq \mathrm{freg}_{C'}(c^*)$. Thus it is enough to prove that $\mathrm{freg}_{C'}(c^*) \subset \mathrm{freg}_{\hat{C}}(c_1) \cup \mathrm{freg}_{\hat{C}}(c_2)$. Suppose for the sake of contradiction that there is $c_3 \in \hat{C} \setminus \{c_1, c_2\}$ such that $\mathrm{freg}_{C'}(c^*) \cap \mathrm{freg}_{\hat{C}}(c_3) \neq \emptyset$; see Figure 3.6. Then $\mathrm{freg}_{C'}(c^*)$ (shown striped in Figure 3.6) has at least three neighbors in $\mathrm{FVD}(C')$, which implies that $\mathrm{freg}_{C'}(c^*)$ contains at least one vertex v of $\mathscr{T}(\hat{C})$. Figure 3.6 shows $\mathrm{hreg}_F^{(\ell)}(C)$ as a grey area, $\mathscr{T}(\hat{C})$ in thick dashed lines,

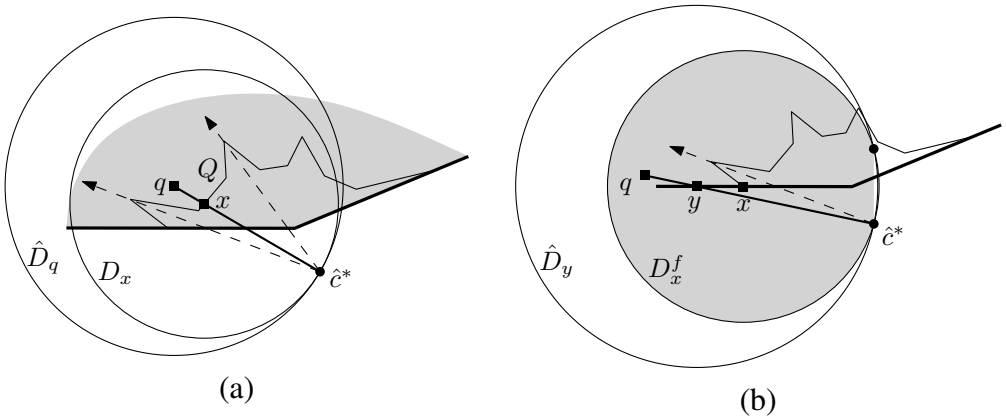


Figure 3.8. Two cases of the proof of Lemma 3.2.7: (a) $\hat{c}^*q$ intersects the Hausdorff boundary of $\hat{c}^*$; (b) $\hat{c}^*q$ does not intersect it

and $\mathscr{T}(C')$ in thin solid lines. Since all points in $\hat{C}$ have non-empty Voronoi regions in $\mathrm{HVD}_F^{(\ell)}(C)$, all vertices of $\mathscr{T}(\hat{C})$ must be contained in $\mathrm{hreg}_F^{(\ell)}(C)$. Thus, $v \in \mathrm{hreg}_F^{(\ell)}(C)$, which implies that c^* is active; a contradiction. □

Lemma 3.2.7. *Let Q be the cluster determined at Line 6 of Algorithm* Step-ℓ *(Figure 3.7). Then $\mathrm{d_f}(q,\hat{Q}) < \mathrm{d_f}(q,\hat{C})$ if and only if $q \notin \overline{\mathrm{hreg}}_{F^{(\ell)}}(C)$.*

Proof. Suppose $q \in \overline{\mathrm{hreg}}_{F^{(\ell)}}(C)$. Let $\hat{F} = \{\hat{P}, P \in F^{(\ell)}\}$ be the family of sets of active points of all clusters in $F^{(\ell)}$. Clearly, $\mathrm{HVD}(\hat{F})$ is identical to $\mathrm{HVD}(F^{(\ell)})$. Since $\mathrm{hreg}_F^{(\ell)}(C) = \mathrm{hreg}_{\hat{F}}(\hat{C})$ it follows that $q \in \overline{\mathrm{hreg}}_{\hat{F}}(\hat{C})$. Therefore $\mathrm{d_f}(q,\hat{C}) \leq \mathrm{d_f}(q,\hat{Q})$.

Suppose $q \notin \overline{\mathrm{hreg}}_{F^{(\ell)}}(C)$. Let $\hat{D}_q$ be the closed disk centered at q with radius $|\hat{c}^*q|$. We prove that $\hat{Q}$ is enclosed in $\hat{D}_q$, which is equivalent to $\mathrm{d_f}(q,\hat{Q}) < \mathrm{d_f}(q,\hat{C})$. There are two cases:

1. Suppose that segment $\hat{c}^*q$ intersects the Hausdorff boundary of $\mathrm{hreg}_F^{(\ell)}(\hat{c}^*)$ at a point x. (By Property 3.1.5, $\hat{c}^*q$ may intersect this boundary at most once). Let $\mathscr{D}_x$ be the closed disk centered at x with radius $\mathrm{d_f}(x,C)$. Since x lies on the boundary between $\mathrm{hreg}_F^{(\ell)}(C)$ and $\mathrm{hreg}_F^{(\ell)}(Q)$, $\mathrm{d_f}(x,C) = \mathrm{d_f}(x,Q)$. Equivalently, $Q \subset \mathscr{D}_x$. Since $q \notin \mathrm{hreg}_F^{(\ell)}(\hat{c}^*)$, then $\mathrm{d_f}(q,\hat{Q}) < \mathrm{d_f}(q,\hat{C}) = |q\hat{c}^*|$. Thus $\hat{Q}$ is enclosed in $\hat{D}_q$. See Figure 3.8a.
2. Suppose that $\hat{c}^*q$ does not intersect the Hausdorff boundary of $\mathrm{hreg}_F^{(\ell)}(\hat{c}^*)$. Since $q \notin \mathrm{hreg}_F^{(\ell)}(C)$, the situation is as in Figure 3.8b. Let y be a point

where $\hat{c}^*q$ intersects $\partial\mathrm{freg}_{\hat{C}}(\hat{c}^*)$; let e be the edge of $\partial\mathrm{freg}_{\hat{C}}(\hat{c}^*)$ that contains y, and let x be the C-mixed Voronoi vertex encountered first as we traverse $\partial\mathrm{freg}_{\hat{C}}(\hat{c}^*)$ from y towards $\mathrm{hreg}_F^{(\ell)}(C)$. Note that Q is the cluster whose Voronoi region is incident to x. Cluster $\hat{Q}$ is either rear or forward limiting with respect to $\hat{C}$, see Definition 8. Without loss of generality, let $\hat{Q}$ be forward limiting, that is, $\hat{Q} \subset D_x^{\mathrm{f}} \cup \mathrm{conv}\hat{C}$. Since $y \notin \mathrm{hreg}_F^{(\ell)}(C)$ and $y \in \overline{\mathrm{freg}_{\hat{C}}}(\hat{c}^*)$, then $D_x^{\mathrm{f}} \cup \mathrm{conv}\hat{C} \subset \hat{D}_y \subset \hat{D}_q$, where $\hat{D}_y$ the closed disk centered at y with radius $\mathrm{d_f}(y,\hat{C})$. Thus, $\hat{Q}$ is enclosed in $\hat{D}_q$.

□

Lemma 3.2.8. *One step of the walk is performed in $O(\log n)$ time.*

Proof. We analyze the time complexity of the algorithm in Figure 3.7. In Line 1, point c^* is determined by locating q in $\mathrm{FVD}(C)$ in $O(\log n)$ time. In Line 5, points c_1, c_2 are determined in time $O(\log n)$ by drawing the tangents from c^* to $\hat{C}$, which is a convex chain. In Line 6, cluster Q is found by binary search in the list $v_1, \ldots, v_j$. Thus, all steps are performed within time $O(\log n)$. Correctness is established by Lemmas 3.2.6 and 3.2.7. □

Parametric point location in the Voronoi hierarchy

In this section we show how to perform parametric point location on a candidate edge uv of $\mathscr{T}(C)$. Recall the definition of a candidate edge (Definition 9) and of a parametric point location query (Definition 10).

We follow the same top-down traversal of the hierarchy, as for the ordinary point location. Starting at the last level h of the hierarchy, at each level ℓ, we search for a cluster $Q^\ell \in F^{(\ell)}$ and a point $u^\ell \in uv$ such that $u^\ell \in \mathrm{hreg}_F^{(\ell)}(Q^\ell)$ and $\mathrm{d_f}(u^\ell, C) = \mathrm{d_f}(u^\ell, Q^\ell)$. The answer to the query is the cluster C^0 and the point u^0 of level 0. If at any level ℓ we find out that the desired cluster Q^ℓ or point u^ℓ do not exist, the answer to the parametric point location query is *nil*. At level ℓ, we determine a sequence $(a_j)_{j=0}^r$ of points on the line segment uv, such that $a_0 = u^{\ell+1}$ and $a_r = u^\ell$. Let Q^{a_j}, $j = 1, \ldots, r$, denote the cluster in $F^{(\ell)}$ such that $a_j \in \mathrm{hreg}_F^{(\ell)}(Q^{a_j})$. For each $j = 0, \ldots, r$ point a_{j+1} is derived from a_j so that it is equidistant to Q^{a_j} and C ($\mathrm{d_f}(a_{j+1}, C) = \mathrm{d_f}(a_{j+1}, Q^{a_j})$).

The algorithm to perform parametric point location is given in Algorithm *Parametric-point-location* in Figure 3.9. At level ℓ, we determine a_{j+1} from a_j by performing a walk starting at Q^{a_j}, and by performing a segment query (see Definition 11) for a subsegment of uv on $\mathrm{FVD}(Q^{a_j})$.

Algorithm *Parametric-point-location*
1. Find Q^h and u^h (by brute force);
2. **for** $\ell = (h-1)$ **downto** 0
3. **do** Set $a_0 = u^{\ell+1}$, and $j = 0$;
4. Find Q^{a_0} by performing a walk at level ℓ starting at $Q^{\ell+1}$;
5. **while** $\mathrm{d_f}(a_j, C) > \mathrm{d_f}(a_j, Q^{a_j})$ **do**
6. **if** $\mathrm{d_f}(v, Q^{a_j}) > \mathrm{d_f}(v, C)$
7. **then** Find $a_{j+1} \in a_j v$ by performing a segment query for $a_j v$ on $\mathrm{FVD}(Q^{a_j})$;
8. Find $Q^{a_{j+1}} \in F^{(\ell)}$ by a walk at level ℓ starting at Q^{a_j};
9. Set $j = j+1$;
10. **else** Exit and **return** nil.
11. Set $Q^\ell = Q^{a_j}$ and $u^\ell = a_j$;
12. Exit and **return** Q^0, u^0.

Figure 3.9. Parametric point location on candidate edge *uv*

Lemma 3.2.9. *The expected length of the sequence* $(a_j)_{j=0}^r$ *at one level of the hierarchy is* $O(1)$.

Proof. Consider the sequence (a_j) at level ℓ. Let $a = a_0$, and let P be the cluster at level $\ell+1$ that is nearest to a. We first prove that for each $j = 0, \ldots, r-1$, a is closer to Q^{a_j} than to P. By the construction of the sequence, $\mathrm{d_f}(a_{j+1}, C) = \mathrm{d_f}(a_{j+1}, Q^{a_j})$ and $\mathrm{d_f}(v, Q^{a_j}) > \mathrm{d_f}(v, C)$. Since clusters C and Q^{a_j} are non-crossing, cluster Q^{a_j} is either forward or rear limiting for C with respect to point a_{j+1}; see Definition 8. By Property 3.1.1, a is closer to Q^{a_j} than to C. Since $\mathrm{d_f}(a, C) = \mathrm{d_f}(a, P)$, we have $\mathrm{d_f}(a, Q^{a_j}) < \mathrm{d_f}(a, P)$. Similarly to the proof of Lemma 3.2.5, we can derive that the expected number of clusters in $F^{(\ell)}$ that are closer to a than to P is constant. In addition, clusters Q^{a_j} for each $j = 0, \ldots, r-1$, are distinct. Thus, r is expected $O(1)$, which proves the claim. □

Lemma 3.2.10. *Parametric point location in the Hausdorff Voronoi hierarchy can be performed in expected* $O(\log n \log k)$ *time.*

Proof. The expected number of clusters at level h of the Voronoi hierarchy is O(1) [48] and computing the distance from a point to a cluster requires $O(\log n)$ time, thus Line 1 of Algorithm *Parametric-point-location* in Figure 3.9 requires expected $O(\log n)$ time. At a level ℓ, $\ell = 0, \ldots, h-1$, it identifies points of the sequence (a_j) one by one, each time performing a walk and a segment query. The expected number of such walks and segment queries is $O(1)$ (see Lemma 3.2.9),

each walk performs expected $O(1)$ steps (see Lemma 3.2.5), and each step of the walk requires $O(\log n)$ time (see Lemma 3.2.8). Each segment query can be performed in time $O(\log n)$ (see Lemma 3.2.3). Since the expected number of levels in the Voronoi hierarchy is $O(\log k)$ (see Lemma 3.2.4), the claim follows. □

Updating the Voronoi hierarchy

To insert a new cluster C in the Hausdorff Voronoi hierarchy, we traverse the hierarchy starting at level 0 until a randomly computed maximum level for C, denoted as $\ell(C)$, is found. Inserting C at a level ℓ may make the region of a cluster P at this level empty.

Definition 14. A cluster $P \in F$ is called *critical at level* ℓ with respect to $C \notin F$, if $\mathrm{hreg}_F^{(\ell-1)}(P) \neq \emptyset$, $\mathrm{hreg}_{F\cup\{C\}}^{(\ell-1)}(P) = \emptyset$, and $\mathrm{hreg}_{F\cup\{C\}}^{(\ell)}(P) \neq \emptyset$.

Such a critical cluster P becomes an obstacle to correct point location. Indeed, if a query point lies in $\mathrm{hreg}_{F\cup\{C\}}^{(\ell)}(P)$, we do not know where to continue the point location at level $\ell-1$. To fix the problem, P could be deleted from all levels of the hierarchy, however, this is computationally expensive. Instead of deleting P, we link P to the cluster or to the pair of clusters responsible for the empty region of P. One of these responsible clusters is C. There are the following cases:

1. Cluster C is a killer for P, and $\ell(C) = \ell - 1$.
2. There is a cluster $K \in F^{(\ell-1)}$ such that $\{C,K\}$ is a killing pair for P, and one of the following holds:
   (a) $\ell(K) \geq \ell$, and $\ell(C) = \ell - 1$;
   (b) $\ell(C) \geq \ell$, and $\ell(K) = \ell - 1$;
   (c) $\ell(C) = \ell(K) = \ell - 1$.

In cases 1 and 2a, we link cluster P to cluster C only, see Lemma 3.2.11. In case 2b we link P to cluster K, and in case 2c to both clusters C and K. In the latter two cases we also need to identify cluster K. The linking process is detailed in Figure 3.10.

Lemma 3.2.11. *Cases 1 or 2a occur if and only if all the P-mixed vertices on the boundary of $\mathrm{hreg}_F^{(\ell)}(P)$ are closer to C than to P.*

Proof. Suppose we have cases 1 or 2a, i.e., $C \in F^{(\ell-1)}$, $C \notin F^{(\ell)}$, and either C is a killer for P (case 1) or there is $K \in F^{(\ell)}$ such that C,K is a killing pair for P

Algorithm *Linking*

1. Let $V_{\ell-1}$ be the list of the P-mixed vertices on $\partial\mathrm{hreg}_F^{(\ell-1)}(P)$;
2. Let V_ℓ be the list of the P-mixed vertices on $\partial\mathrm{hreg}_F^{(\ell)}(P)$;
3. **if** all vertices in V_ℓ are closer to C than to P
4. **then** Link P to C and **return** .
5. **else** Let $v \in V_\ell$ be closer to P than to C;
6. Let $c \in C$ be such that $\mathrm{d_f}(v,C) = d(v,c)$;
7. **for** $u \in V_{\ell-1}$
8. let $p_1, p_2 \in P$ and $q \in Q$ be such that v borders $\mathrm{hreg}_F^{(\ell)}(p_1)$, $\mathrm{hreg}_F^{(\ell)}(p_2)$ and $\mathrm{hreg}_F^{(\ell)}(q)$;
9. **if** c and q lie on different sides of chord $\overline{p_1p_2}$
10. **then** set $K = Q$; **if** $\ell(C) \geq \ell$
11. **then** Link P to K and **return** .
12. **else** Link P to $\{C,K\}$ and **return** .

Figure 3.10. Linking cluster P that is critical at level w.r.t. cluster C

(case 2a). In any of these two cases the addition of C to $\mathrm{HVD}(F^{(\ell)})$ would make the region of P empty ($\mathrm{hreg}_{F^{(\ell)}\cup C}(P) = \emptyset$). Thus, each point in $\overline{\mathrm{hreg}}_{F^{(\ell)}}(P)$, including all the P-mixed vertices, is closer to C than to P.

Now suppose that all P-mixed vertices on the boundary of $\mathrm{hreg}_F^{(\ell)}(P)$ are closer to C than to P. Clearly none of these P-mixed vertices is contained in $\overline{\mathrm{hreg}}_{F^{(\ell)}\cup C}(P)$. We need to prove that either C is a killer for P (case 1) or C,K is the killing pair for P for some cluster $K \in F^{(\ell)}$. Suppose on the contrary, that neither of these two cases holds. Then by Property 3.1.3, $\mathrm{hreg}_{F^{(\ell)}\cup C}(P)$ is not empty, and by Property 3.1.2a, $\mathrm{hreg}_{F^{(\ell)}\cup C}(P)$ has at least two P-mixed vertices on its boundary. These vertices are equidistant to P and C; let v be any of them. Since P and C are non-crossing, we have that C is forward or rear limiting for P with respect to v, see Definition 8. By Property 3.1.1, there is a subtree of $\mathscr{T}(P)$ incident to v ($\mathscr{T}_v^r$ or $\mathscr{T}_v^f$) such that all its points are closer to P than to C. This subtree includes at least one P-mixed vertex on the boundary of the region of P in $\mathrm{HVD}(F^{(\ell)} \cup C)$; a contradiction. □

Lemma 3.2.12. *Algorithm* Linking *in Figure 3.10 performs the linking correctly. That is, for any point $x \in \mathrm{hreg}_{F\cup C}^{(\ell)}(P)$, $\mathrm{d_f}(x,Q) < \mathrm{d_f}(x,P)$, where Q is the cluster (or one of the two clusters) linked to P.*

Proof. We need to prove that Algorithm *Linking* (Figure 3.10) always identifies a

cluster (or a pair of clusters) such that any point in $\mathrm{hreg}_F^{(\ell)}(P)$ is closer to these cluster(s) than to P.

Suppose that Line 4 of the procedure is executed, i.e., linking is done to cluster C. Then by Lemma 3.2.11, cases 1 or 2a occur, and Property 3.1.3 guarantees that any point in $\mathrm{hreg}_F^{(\ell)}(P)$ is closer to C than to P. Thus, the linking to C is correctly done.

Suppose that Algorithm *Linking* (Figure 3.10) does not terminate at Line 4. Let vertex v and point c be as determined in Lines 6 and 7 respectively. Since $\mathrm{d_f}(v,C) > \mathrm{d_f}(v,P)$ and $\mathrm{d_f}(v,C) = d(v,c)$, we have $c \notin \mathrm{conv}\, P$. Since $\mathrm{hreg}_{F\cup\{C\}}^{(\ell-1)}(P) = \emptyset$ and v is closer to P than to C, we have $v \notin \mathrm{hreg}_F^{(\ell-1)}(P)$. Cluster C is (forward or rear) limiting for P with respect to any P-mixed vertex w on the boundary of $\mathrm{hreg}_F^{(\ell-1)}(P)$. Suppose, without loss of generality, that C is forward limiting, i.e., $C \subset D_w^{\mathrm{f}} \cup \mathrm{conv}\, P$; then $c \in D_w^{\mathrm{f}} \setminus \mathrm{conv}\, P$. Let u be the first P-mixed vertex encountered as we traverse $\mathscr{T}(P)$ from v to its portion enclosed in $\mathrm{hreg}_F^{(\ell-1)}(P)$. Let Q be the cluster inducing u, and q be the point in Q such that $d(u,q) = \mathrm{d_f}(u,Q)$. The pair $\{Q,C\}$ is by definition a killing pair for P, and q, c lie at opposite sides of the chord of P inducing u. All other P-mixed vertices $v_i \neq u$ on $\partial\mathrm{hreg}_F^{(\ell-1)}(P)$ must be induced by forward limiting clusters, see Property 3.1.1. Thus, any point q_i, $q_i \neq q$, inducing a P-mixed vertex v_i must lie on the same side of the corresponding chord as c. Thus, Line 12 correctly sets $K = Q$.

The check in Line 13 distinguishes between cases 2b (Line 14) and 2c (Line 16). Property 3.1.3 again guarantees correctness. □

We summarize in the following theorem.

Theorem 3.2.1. *The Voronoi hierarchy for the Hausdorff Voronoi diagram of a family of k clusters of total complexity n has expected size $O(n)$. Both the point location query and the parametric point location query can be performed in expected time $O(\log n \log k)$. Insertion of a cluster takes $O((N/k)\log n)$ amortized time, where N is the total number of update operations in all levels during the insertion of all k clusters.*

Proof. The expected space of the Voronoi hierarchy is analyzed in Lemma 3.2.4. Lemmas 3.2.4 to 3.2.7 imply that point location in the Voronoi hierarchy can be done in expected $O(\log n \log k)$ time. By Lemma 3.2.10, parametric point location is performed in expected $O(\log n \log k)$ time. During the insertion of a cluster, two procedures are performed: updating the diagram at all necessary levels, and the linking of regions that disappear. Since updating each (constant-sized) element of

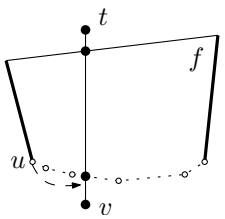


Figure 3.11. Tracing the $\mathscr{T}$-chain of face f (dotted lines), starting from the endpoint u

a diagram requires $O(1)$ time, the total time required for all update operations to insert all k clusters is $O(N)$.

Consider the linking of a cluster P that is critical at level ℓ with respect to a cluster C. We visit the P-mixed vertices of $\mathrm{HVD}(F^{(\ell-1)})$, but these vertices get deleted during the same step. We also visit the P-mixed vertices of $\mathrm{HVD}((F \cup \{C\})^{(\ell)})$. The latter vertices are visited at most twice: when P is critical at level $\ell+1$ and when P is critical at level ℓ. The time complexity of each of these visits is $O(\log n)$. Thus, the claimed complexity follows. □

3.2.3 Tracing a new Voronoi region

In this section we give details on how to compute the boundary of a new region $\mathrm{hreg}_{F_i}(C_i)$ within $\mathrm{HVD}(F_{i-1})$, given a representative point t in $\mathrm{hreg}_{F_i}(C_i)$. The first task is to determine a point w on the boundary of $\mathrm{hreg}_{F_i}(C_i)$. Then tracing can be performed as described in [66].

Let $\mathrm{HVD}^*(F)$ denote the visibility-based decomposition of $\mathrm{HVD}(F)$. The main algorithm (see the beginning of Section 3.2) has identified a segment tv, along an edge of $\mathscr{T}(C_i)$, where t is the representative point in $\mathrm{hreg}_{F_i}(C_i)$, and v is the parent of t in $\mathscr{T}(C_i)$ such that $v \notin \mathrm{hreg}_{F_i}(C_i)$. We determine a C_i-mixed vertex w along tv. To this goal, we trace segment tv through $\mathrm{HVD}(F_{i-1})$, starting at t, until we determine w.

In more detail, let f be a face of $\mathrm{HVD}^*(F_{i-1})$ intersected by tv. Let us call $\mathscr{T}$-chain of f the portion of $\mathscr{T}(P)$ on ∂f, where P is such that $f \subset \mathrm{hreg}_{F_{i-1}}(P)$. Initially, f is the face of $\mathrm{HVD}^*(F_{i-1})$ containing t. In constant time, we check whether w lies in the interior of f, and if so we identify w. If it does not, we move to the face g of $\mathrm{HVD}^*(F_{i-1})$ that is adjacent to f and is intersected by tv. To identify g, we may need to trace a portion of the $\mathscr{T}$-chain of f. This is performed as follows: Among the two endpoints of the $\mathscr{T}$-chain, at least one must be in $\mathrm{hreg}_{F_i}(C_i)$, by Property 3.1.2a. We first identify such an endpoint u, and then we trace the $\mathscr{T}$-chain, starting at u, until we meet its intersection with tv, see Fig-

ure 3.11b. At this time, we have determined g and we can continue our search for w with $f = g$. Tracing the $\mathscr{T}$-chain of f, starting at u, has no effect on the overall time complexity because all traced edges of the $\mathscr{T}$-chain intersect $\mathrm{hreg}_{F_i}(C_i)$, and thus, they are guaranteed to be deleted from $\mathrm{HVD}(F_{i-1})$ by the main algorithm during the insertion of C_i at step i. To identify u, we consider both endpoints of the $\mathscr{T}$-chain, and compare their distances to C_i and to their closest cluster in F_{i-1}. The latter distance is readily available from $\mathrm{HVD}(F_{i-1})$. To derive the former distance, we perform point location in $\mathrm{FVD}(C_i)$. Thus, in the worst case, we perform two point locations in $O(\log n)$ time, and in addition, we trace a number of edges of $\mathrm{HVD}(F_{i-1})$, none of which will appear in $\mathrm{HVD}(F_i)$, spending $O(1)$ time per edge.

After w is identified, the tracing of the boundary of $\mathrm{hreg}_{F_i}(C_i)$ is performed in time proportional to the total number of edges that are inserted or deleted from the Hausdorff diagram during step i, plus $|C_i|$. Note that to identify the new Voronoi vertices we simply walk sequentially along edges of $\mathrm{HVD}(F_{i-1})$ and $\mathrm{FVD}(C_i)$, which are deleted, using the visibility-based decomposition. To identify w, we also perform point location, thus, an $O(\log n)$ factor is multiplied to the above quantity.

We conclude that the time complexity for tracing $\mathrm{hreg}_{F_i}(C_i)$ is proportional to the number of updates (insertions and deletions) in the Hausdorff diagram as a result of inserting cluster C_i, multiplied by $O(\log n)$. Combining with the overall time complexity analysis, given in the following section, the total expected time devoted to the tracing of new regions throughout the algorithm is $O(n \log n)$.

3.2.4 Complexity analysis

The running time of our algorithm depends on the number of update operations (insertions and deletions) during the construction of the diagram. Based on the Clarkson-Shor technique [26], we prove that the expectation of this number is linear, when clusters are inserted in random order. In the Hausdorff Voronoi diagram, sites (clusters) do not have constant size, as it is typically assumed in the literature. Thus, we need to adapt the standard probabilistic arguments in this environment.

Theorem 3.2.2. *Given a family F of non-crossing clusters of points, the expected total number of update operations during the randomized incremental construction of* $\mathrm{HVD}(F)$ *is $O(n)$, where n is the total complexity of the clusters in F.*

Theorem 3.2.2 can be extended to all levels of the Voronoi hierarchy as stated in the following corollary. We defer its proof to Section 3.2.4, after the proof of Theorem 3.2.2. The proof of Theorem 3.2.2 is contributed by Panagiotis Cheilaris.

Corollary 3.2.1. *The expected number of update operations made on all the levels of the Hausdorff Voronoi hierarchy of F during the incremental construction is $O(n)$.*

We conclude with the following theorem.

Theorem 3.2.3. *The Hausdorff Voronoi diagram of a family F of non-crossing clusters can be constructed in $O(n \log n \log k)$ expected time and $O(n)$ expected space, where k is the number of clusters in F and n is number of points in all clusters.*

Proof. As a preprocessing, we build the centroid decomposition for each cluster in F, in total $O(n \log n)$ time (see Lemma 3.2.3). The algorithm to insert a cluster $C \in F$ does the following: (1) searches for a representative point in the new Hausdorff Voronoi region; (2) traces the boundary of the new region (see Section 3.2.3); and (3) inserts C in the Voronoi hierarchy (see Section 3.2.2). By the discussion in Section 3.2.3, and by Theorem 3.2.2 and Corollary 3.2.1, the total time to perform (2) and (3), for all clusters, is expected $O(n \log n)$. Searching for a representative point in the Hausdorff Voronoi region of C (part (1)) performs $O(|C|)$ point location queries and at most one parametric point location query in the Voronoi hierarchy. Combining with Lemma 3.2.3 and Theorem 3.2.1, we derive that the total expected time to determine a representative point for all clusters is $O(n \log n \log k)$; the claim follows. □

Remark 3.2.1. *Deterministic $O(n)$ space complexity can be achieved by using a dynamic point location data structure for a planar subdivision [5, 12]. On this data structure, parametric point location can be performed as described in Cheong et al. [23]. The time complexity of such a query is t_q^2, where t_q is the time complexity of point location in the chosen data structure. In particular, the data structure by Baumgarten et al. [12] has $t_q \in O(\log n \log \log n)$, which leads to the construction of the Hausdorff Voronoi diagram with expected running time $O(n \log^2 n (\log \log n)^2)$ and deterministic space $O(n)$.*

Proof of Theorem 3.2.2

In order to count the number of update operations of the algorithm (i.e., insertions and deletions of *features* (vertices, edges, faces) of the incrementally constructed diagram), we will associate these operations with such features. Each feature of the diagram that appears during the incremental algorithm has been inserted by an operation. If a feature is deleted, then it cannot be inserted again in the future. As a result, the number of deletion operations is bounded by the number of insertion

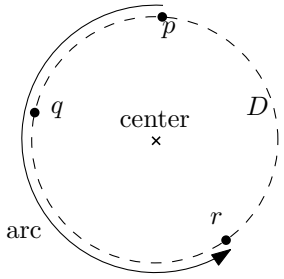


Figure 3.12. A configuration (p,q,r)

operations. So, we intend to prove that the expected number of features that appear during the construction of the diagram is $O(n)$. To that end, we can ignore features associated only with the farthest Voronoi diagrams of each cluster, because their total worst case combinatorial complexity is $O(n)$.

Configurations. We give some definitions related to features of the diagram.

Definition 15. A *configuration* is a triple of points (p,q,r) such that p, q, r lie on the boundary of a disk D and q is contained in the interior of the counterclockwise arc from p to r. We call D the *disk of the configuration*, its center the *center* of the configuration, and the counterclockwise arc pr the *arc of the configuration*. See Figure 3.12.

A configuration is *pure* if its three points belong to *three* different clusters of F and all other points of these three clusters are contained in the interior of the disk of the configuration.

A configuration is *mixed* if its three points belong to *two* different clusters of F and all other points of these two clusters are contained in the interior of the disk of the configuration.

From now on, configurations of our interest will be either pure or mixed. Therefore, each configuration is either associated with three (a pure one) or two (a mixed one) clusters.

Definition 16. A cluster C is in *conflict with a configuration* if (a) C does not contain any of the points in the configuration, and (b) C is contained in the union of the interior of the disk of the configuration and the arc of the configuration.

The *weight* of a configuration is the number of clusters in conflict with it.

Definition 16 is general and it does not follow the general position assumption stated in Section 3.1. Under this assumption (b) can simplify to: "(b) C is contained in the union of the interior of the disk of the configuration".

Lemma 3.2.13. *The number of zero-weight configurations of F is of the same order as the combinatorial complexity of the Hausdorff Voronoi diagram of F.*

Proof. Each zero-weight configuration is associated with a vertex of the Hausdorff Voronoi diagram. Indeed, the center of this configuration is at the vertex and the disk of the configuration that contains the clusters associated with the configuration. Consider a vertex v of the Hausdorff Voronoi diagram. The degree of v in the diagram equals the number of configurations with center v plus the number of some features that are associated just with farthest Voronoi diagrams (that we have claimed before that we can ignore). As a result, zero-weight configurations estimate well the combinatorial complexity of the Hausdorff Voronoi diagram. □

Configurations of weight at most k. Let $K_0^{\text{pure}}(F)$, $K_k^{\text{pure}}(F)$, $K_{\leq k}^{\text{pure}}(F)$ denote the sets of pure configurations of zero weight, weight equal to k, and weight at most k, of a family F of non-crossing clusters, respectively. Let $N_0^{\text{pure}}(F)$, $N_k^{\text{pure}}(F)$, $N_{\leq k}^{\text{pure}}(F)$ denote the cardinality of the aforementioned sets, respectively. Define analogously the sets of mixed configurations $K_0^{\text{mix}}(F)$, $K_k^{\text{mix}}(F)$, $K_{\leq k}^{\text{mix}}(F)$ and their cardinalities $N_0^{\text{mix}}(F)$, $N_k^{\text{mix}}(F)$, $N_{\leq k}^{\text{mix}}(F)$, respectively. Both $N_0^{\text{pure}}(F)$ and $N_0^{\text{mix}}(F)$ are $O\big(\sum_{C\in F}|C|\big) = O(n)$ [62]. Then, using the Clarkson-Shor technique [26], and in particular [76, Theorem 1.2], with a random sample of the clusters in F, we obtain:

$$N_{\leq k}^{\text{pure}}(F) \leq c^{\text{pure}} \cdot nk^2 \text{ and } N_{\leq k}^{\text{mix}}(F) \leq c^{\text{mix}} \cdot nk,$$

for $k > 0$ and some constants c^{pure} and c^{mix}. The details to obtain these bounds are quite standard and we refer the interested reader to [26].

Appearance of a feature. Consider a configuration c of weight k in family F with m clusters.[1] Assume the Hausdorff Voronoi diagram of F is constructed with the incremental algorithm and the clusters are inserted according to permutation π. The feature corresponding to c appears at some stage of the incremental algorithm if and only if the clusters associated with c occur in π *before* the k clusters that conflict with configuration c. This event happens with probability

$$\Pr[\text{pure } c \text{ feature appears}] = \frac{3!k!}{(k+3)!} = \frac{6}{(k+1)(k+2)(k+3)}$$

[1] Note the difference in notation from previous sections: here k denotes the weight of a configuration and m denotes the number of clusters in F. We do this change in order to be consistent with the notation of Sharir [76].

for pure configurations, and with probability

$$\Pr[\text{mixed } c \text{ feature appears}] = \frac{2!k!}{(k+2)!} = \frac{2}{(k+1)(k+2)}$$

for mixed configurations.

The expected number of appearances of features corresponding to a pure configuration is therefore:

$$\begin{aligned}
&\sum_{k=0}^{m-3} \sum_{c \in K_k^{\text{pure}}(F)} \Pr[\text{pure } c \text{ feature appears}] = \sum_{k=0}^{m-3} \sum_{c \in K_k^{\text{pure}}(F)} \frac{6}{(k+1)(k+2)(k+3)} \\
&= 6 \sum_{k=0}^{m-3} \frac{N_k^{\text{pure}}(F)}{(k+1)(k+2)(k+3)} = N_0^{\text{pure}}(F) + 6 \sum_{k=1}^{m-3} \frac{N_{\le k}^{\text{pure}}(F) - N_{\le k-1}^{\text{pure}}(F)}{(k+1)(k+2)(k+3)} \\
&= \frac{3}{4} N_0^{\text{pure}}(F) + 18 \sum_{k=1}^{m-4} \frac{N_{\le k}^{\text{pure}}(F)}{(k+1)(k+2)(k+3)(k+4)} + \frac{N_{\le m-3}^{\text{pure}}(F)}{(m-2)(m-1)m} \\
&\le \frac{3}{4} N_0^{\text{pure}}(F) + 18 \sum_{k=1}^{m-4} \frac{c^{\text{pure}} \cdot nk^2}{(k+1)(k+2)(k+3)(k+4)} + \frac{c^{\text{pure}} \cdot n(m-3)^2}{(m-2)(m-1)m} \\
&\le \frac{3}{4} N_0^{\text{pure}}(F) + 18 \cdot c^{\text{pure}} \cdot n \sum_{k=1}^{m-4} \frac{1}{k^2} + \frac{c^{\text{pure}} \cdot n}{m} = O(n)
\end{aligned}$$

Similarly, the expected number of appearances of features corresponding to a mixed configuration is:

$$\begin{aligned}
&\sum_{k=0}^{m-2} \sum_{c \in K_k^{\text{mix}}(F)} \Pr[\text{mixed } c \text{ feature appears}] = \sum_{k=0}^{m-2} \sum_{c \in K_k^{\text{mix}}(F)} \frac{2}{(k+1)(k+2)} \\
&= 2 \sum_{k=0}^{m-2} \frac{N_k^{\text{mix}}(F)}{(k+1)(k+2)} = N_0^{\text{mix}}(F) + 2 \sum_{k=1}^{m-2} \frac{N_{\le k}^{\text{mix}}(F) - N_{\le k-1}^{\text{mix}}(F)}{(k+1)(k+2)} \\
&= \frac{1}{2} N_0^{\text{mix}}(F) + 4 \sum_{k=1}^{m-3} \frac{N_{\le k}^{\text{mix}}(F)}{(k+1)(k+2)(k+3)} + \frac{c^{\text{mix}} \cdot n(m-2)}{(m-1)m} \\
&\le \frac{1}{2} N_0^{\text{mix}}(F) + 4 \cdot c^{\text{mix}} \cdot n \sum_{k=1}^{m-3} \frac{1}{k^2} + \frac{c^{\text{mix}} \cdot n}{m} = O(n)
\end{aligned}$$

Therefore, we have proved the following, which implies Theorem 3.2.2.

Lemma 3.2.14. *The expected number of features that appear during the incremental construction is $O(n)$.*

Proof of Corollary 3.2.1

By the discussion in the proof of Theorem 3.2.2, the expected number of structural changes during the incremental construction is proportional to the expected number of appearing features (i.e., pure and mixed vertices). For a fixed level ℓ, the expected total number of points in $F^{(\ell)}$ is $\beta^\ell n$. By Lemma 3.2.14, the expected number of features that appear during the incremental construction of $\mathrm{HVD}(F^{(\ell)})$ at level ℓ is $O(\beta^\ell n)$. Therefore, the expected total number of features that appear at all the levels is at most $\sum_{\ell=0}^{\infty} O(\beta^\ell n) = O(n)$.

This concludes our presentation of the RIC for the Hausdorff Voronoi diagram of non-crossing clusters, based on point location. The rest of this chapter presents a classic RIC for the Hausdorff Voronoi diagram.

3.3 A classic randomized incremental construction of the Hausdorff Voronoi diagram of non-crossing clusters

In this section we give a construction algorithm for the Hausdorff Voronoi diagram for a family F of non-crossing clusters of points in $\mathbb{R}^2$. Recall that, since clusters in F are non-crossing, the Hausdorff Voronoi regions are connected and the combinatorial complexity of the diagram is $O(n)$.

Despite the $O(n)$ complexity of the diagram, a direct application of the RIC framework does not result in an efficient algorithm. This is because input clusters, and thus the edges of the Voronoi diagram, are of non-constant complexity. If we assigned *objects* to be clusters, and *ranges* so that one Voronoi edge corresponds to one or two ranges, then both objects and ranges would have non-constant complexity. This would imply that the work to update the conflict/history graph is not proportional to the number of conflicts created or deleted, but rather to the total complexity of the objects/ranges involved in these conflicts, contradicting the update condition, see Section 2.1.1. Still the framework can be applied at the cost of a multiplicative factor that matches the time required to test whether a given object is in conflict with a given range [17]. However, such test may require $\Omega(n)$ time, resulting in a $O(n^2 \log k)$ time algorithm. To reduce the multiplicative factor we choose ranges to be simpler than Hausdorff Voronoi edges. This choice, however, yields another problem: The number of new ranges that appear in the place of one deleted range R, may be $\Omega(n)$. To overcome these problems, we introduce a new

definition of a conflict, and give a procedure to find new conflicts efficiently.

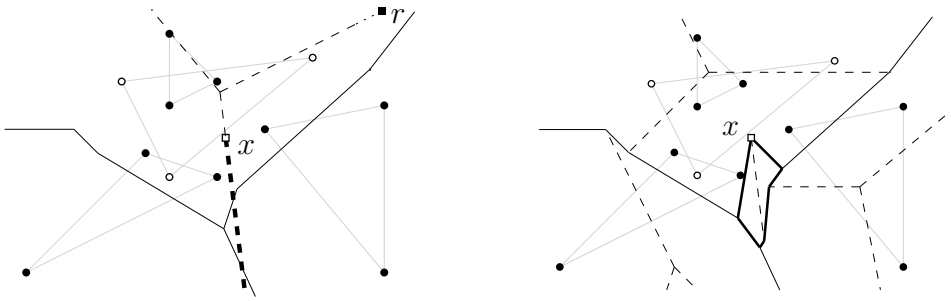


Figure 3.13. Insertion of a cluster C (filled disks): (a) $\mathscr{T}(C)$ (dashed) rooted at r, its active subtree (bold); (b) the HVD after the insertion: x is the rear C-mixed vertex

Defining conflicts and ranges

Let $S \subset F$, and $C \in F \setminus S$. Recall from Section 3.1, $\mathscr{T}(C)$ is rooted at a point at infinity along one of its unbounded edges.

Let $\mathscr{T}_a(C,S)$ denote the *active subtree* of $\mathscr{T}(C)$ with respect to S, which is the portion of $\mathscr{T}(C)$ relevant to the insertion of C in $\mathrm{HVD}(S)$. In particular, let x be the first point on the boundary of $\mathrm{hreg}_{S\cup\{C\}}(C)$ that is encountered as we traverse $\mathscr{T}(C)$ starting at its root. Then $\mathscr{T}_a(C,S)$ is the subtree of $\mathscr{T}(C)$ following x, and x is the root of $\mathscr{T}_a(C,S)$. Point x is a mixed vertex (unless the root of $\mathscr{T}(C)$ is contained in $\mathrm{hreg}_{S\cup\{C\}}(C)$, in which case $\mathscr{T}_a(C,S)$ and $\mathscr{T}(C)$ coincide).

Figures 3.13a and 3.13b illustrate $\mathrm{HVD}(S)$ and $\mathrm{HVD}(S\cup\{C\})$ respectively. The points of C are shown as unfilled circles. Figure 3.13a illustrates also $\mathscr{T}(C)$ in dashed lines, superimposed on $\mathrm{HVD}(S)$; $\mathscr{T}(C)$ is rooted at r. Point x is the root of $\mathscr{T}_a(C,S)$ which is shown in bold. Figure 3.13b shows the boundary of $\mathrm{hreg}_{S\cup\{C\}}(C)$ in bold lines. The following property is implied by Property 3.1.4, given that clusters in F are non-crossing, and thus, Voronoi regions are connected.

Property 3.3.1. *A cluster C has a non-empty Voronoi region in* $\mathrm{HVD}(S\cup\{C\})$*, $S \subseteq F$, if and only if $\mathscr{T}_a(C,S)$ is non-empty. Moreover, the root of $\mathscr{T}_a(C,S)$ is a C-mixed vertex of* $\mathrm{HVD}(S\cup\{C\})$ *(possibly a vertex at infinity).*

Property 3.3.1 suggests that once the root of the active subtree $\mathscr{T}_a(C,S)$ is determined, cluster C can easily be inserted into $\mathrm{HVD}(S)$. Thus, the crux of our problem becomes to efficiently maintain the up-to-date root of $\mathscr{T}_a(C,S)$ for every C in $F \setminus S$.

We now formulate the problem in terms of *objects*, *ranges* and *conflicts*. Objects are the clusters in F. Ranges are the faces of $\mathrm{HVD}(S)$ as refined by the visibility-based decomposition (see Section 3.1). A range corresponding to face f, $f \subset \mathrm{hreg}_S(P)$, is said to be defined by the clusters whose Voronoi regions border f (including P). By Observation 3.1.1 these clusters are at most four.

Definition 17 (Conflict for non-crossing clusters)**.** A range f is *in conflict* with a cluster $C \in F \setminus S$, if $\mathscr{T}_a(C,S)$ is not empty and its root x lies in f. The triple (f,x,C) is called a *conflict*. The *list of conflicts* of a range f is denoted as $\mathscr{L}(f)$.

Each range f stores a pointer to its owner p and a pointer to the list of its conflicts $\mathscr{L}(f)$. The same notation is used for both a range and its corresponding face. The following is a direct implication of the above definition and Property 3.3.1.

Corollary 3.3.1. *Each cluster $C \in F \setminus S$ has at most one conflict. If a cluster C has no conflicts, then $\mathrm{hreg}_{S\cup\{C\}}(C) = \emptyset$, and thus, $\mathrm{hreg}_F(C) = \emptyset$.*

Our definition of a conflict can be seen as a generalization of the idea used in the RIC for the trapezoidal map of line segments [17, Ch. 5.3], where conflicts are defined in terms of the (static) left endpoint of each line segment. Here, the corresponding point is dynamic, being a point on $\mathscr{T}(C)$, which serves as a *skeleton* representing a cluster. This dynamic definition of conflict could be of use to other non-standard applications of the RIC framework.

Insertion of a cluster, variant with the conflict graph

Suppose that $\mathrm{HVD}(S), S \subset F$ has been constructed together with its conflict graph. The algorithm to insert a cluster $C \in F \setminus S$ in $\mathrm{HVD}(S)$ is given as pseudocode in Figure 3.14.

To update the Hausdorff Voronoi diagram after the insertion of C, the boundary of $\mathrm{hreg}_{S\cup\{C\}}(C)$ is traced, starting at the root of $\mathscr{T}_a(C,S)$. This tracing is detailed in [66]. The update of the conflict graph is detailed in Lines 4–16 of Figure 3.14. In particular, for each deleted range f we delete its conflicts, and create new conflicts in place of the deleted ones. Let (f,y,Q) be a conflict of f. If $\mathscr{T}_a(Q,S\cup\{C\})$ remains the same as $\mathscr{T}_a(Q,S)$, then we find a new range that contains the root y of $\mathscr{T}_a(Q,S\cup\{C\})$ (Line 9), by performing a binary search on the sides of the ranges comprising $\mathrm{hreg}_{S\cup\{C\}}(p)$. Otherwise, we first find the root z of $\mathscr{T}_a(Q,S\cup\{C\})$ and then find the range of $\mathrm{HVD}(S)$ that contains z. Using the (updated) root of the active subtree and the range containing it, we create the new conflict of Q. To locate a range in Line 15 we do the following: Find the owner $c \in C$ of the range h, such that $z \in h$, by performing a point location for z in $\mathrm{FVD}(C)$; The actual range h can be found by binary search as in Line 9.

Algorithm *Insert-NonCrossing*
(∗ Inserts a cluster C in $\mathsf{HVD}(S)$; updates the conflict graph ∗)
1. Let (g, x, C) be the conflict of C.
2. Trace the region $\mathsf{hreg}_{S\cup\{C\}}(C)$ starting from x.
3. Update $\mathsf{HVD}(S)$ to $\mathsf{HVD}(S \cup \{C\})$.
4. **for** $f \in \mathsf{HVD}(S) \setminus \mathsf{HVD}(S \cup \{C\})$ **do**
5. Let p be the owner of f.
6. **for** each conflict $(f, y, Q) \in \mathcal{L}(f)$ **do**
7. Discard the conflict (f, y, Q).
8. **if** $\mathsf{d_f}(y, p) < \mathsf{d_f}(y, C)$ **then**
9. Locate the range $f' \subset \mathsf{hreg}_{S\cup\{C\}}(p)$ that contains y.
10. Create conflict (f', y, Q).
11. **else**
12. Search for an edge uv in $\mathcal{T}_a(Q, S)$ such that $\mathsf{d_f}(u, Q) > \mathsf{d_f}(u, C)$ and $\mathsf{d_f}(v, Q) < \mathsf{d_f}(v, C)$.
13. **if** uv is found **then**
14. Perform a segment query for uv in $\mathsf{FVD}(C)$ to find root z of $\mathcal{T}_a(Q, S \cup \{C\})$.
15. Locate the range $h \subset \mathsf{hreg}_{S\cup\{C\}}(C)$ that contains z.
16. Create conflict (h, z, Q)

Figure 3.14. Algorithm to insert cluster C; case of non-crossing clusters

Lemma 3.3.1. *The algorithm* Insert-NonCrossing *(Figure 3.14) is correct.*

Proof. We only consider important points that are not easy to see. In Line 8, it is enough to compare distance from y to only p and C as no other cluster may become the closest to y as a result of inserting C. Suppose that $\mathrm{d_f}(y, C) < \mathrm{d_f}(y, Q) = \mathrm{d_f}(y, p)$, and that $\mathcal{T}_a(Q, S\cup\{C\}) \neq \emptyset$ (Lines 11—16). In this case y is no longer a part of the (updated) diagram $\mathrm{HVD}(S\cup\{C\})$. Clearly, $\mathcal{T}_a(Q, S\cup\{C\})$ is a subtree of $\mathcal{T}_a(Q, S)$, thus, there is exactly one edge uv of $\mathcal{T}_a(Q, S)$ such that u is not in $\mathcal{T}_a(Q, S\cup\{C\})$ and v is in $\mathcal{T}_a(Q, S\cup\{C\})$. Edge uv satisfies the condition of Line 12 by definition of an active subtree. The root of active subtree never coincides with a vertex of $\mathcal{T}(C)$ due to the general position assumption. □

Complexity analysis for the conflict graph

We now analyze the time and space complexity of our algorithm. Consider a random permutation $\{C_1, \ldots, C_k\}$ of the input family F, and the sequence $\{F_0, \ldots, F_k\}$, where $F_0 = \emptyset$, $F_i = F_{i-1} \cup \{C_i\}$ and $F_k = F$. At step i we insert cluster C_i, following the procedure described in Section 3.3.

Corollary 3.3.1 directly implies the following.

Lemma 3.3.2. *The number of edges in the conflict graph at any step is $O(k)$.*

Lemma 3.3.3. *Updating the conflict graph at step i of the algorithm requires $O((N_i + R_i)\log n)$ time, where N_i is the total number of edges dropped out of the active subtrees of clusters in $F \setminus F_i$ at this step; R_i is the total number of conflicts deleted at this step.*

Proof. Updating the conflict graph corresponds to two nested for loops in Lines 4–16 of the algorithm in Figure 3.14. Clearly the inner loop (Lines 6–15) is performed $O(R_i)$ times. Inside this loop, the breath-first search is performed that spends $O(\log n)$ time per visited edge. By Corollary 3.3.1, one active subtree is considered at most once. All the visited edges, except the last one, are dropped out of the respective active subtree. It remains to show, that, except for the breath-first search, the rest of the work in any execution of the inner loop requires $O(\log n)$ time. Indeed, it is a point location in Line 8, a segment query in $\mathrm{FVD}(C_i)$ in Line 13, and a binary search in Line 9 or Line 15. □

Theorem 3.3.1. *The randomized incremental construction algorithm to compute the Hausdorff Voronoi diagram of k non-crossing clusters of total complexity n, requires $O(n)$ space, and expected $O(n\log n + k\log n\log k)$ time.*

Proof. The expected total number of ranges created and deleted during the algorithm is $O(n)$, see Theorem 3.2.2. Updating the Hausdorff Voronoi diagram can be done in time proportional to this number using the tracing described in [66]. Now we will analyze only the work to update the conflict graph. By Lemma 3.3.3, the total time required for this task is $\mathscr{T} = \sum_{i=1}^{k} O((N_i + R_i)\log n)$. An edge of $\mathscr{T}(C)$ of any cluster $C \in F$ is dropped out from the active subtree at most once. The total number of edges in the farthest Voronoi diagrams of all clusters in F is $O(n)$. Thus $\mathscr{T} = O(\log n)(n + \sum_{i=1}^{k} R_i)$.

Applying backwards analysis, it is easy to see that the expected number of the conflicts created at step i is $O(k/i)$. Each conflict deleted at step i either induces one new conflict, or the cluster in $F \setminus F_i$ corresponding to that conflict stop having a conflict at step i.

The expectation of $\mathscr{T}$ is $O(n + \sum_{i=1}^{k} (k/i + D_i \log n))$, where D_i is the number of clusters in $F \setminus F_i$ that stop having a conflict at step i. Note that $\sum_{i=1}^{k} D_i \leq k$, since the active subtree of a cluster can become empty at most once. The claimed time complexity follows.

The space requirement at any step is proportional to the combinatorial complexity of the Hausdorff Voronoi diagram, which is $O(n)$, plus the total number of edges of the conflict graph at this step, which is at most k by Lemma 3.3.2. Hence the claimed $O(n)$ bound holds. □

Adapting the algorithm for a history graph

Suppose that the algorithm maintains a *history graph*. Let $\mathscr{H}(F_i)$ denote the history graph that has been computed at step i. $\mathscr{H}(F_0)$ is a single node corresponding to the whole $\mathbb{R}^2$. For $i \in \{1,\ldots,k\}$, $\mathscr{H}(F_i)$ consists of all nodes and edges of $\mathscr{H}(F_{i-1})$ and in addition it contains the following: (i) A node for each new range in $\mathrm{HVD}(F_i) \setminus \mathrm{HVD}(F_{i-1})$; these nodes are called the *nodes of level i*. (ii) An edge connecting a deleted range $f \in \mathrm{HVD}(F_{i-1}) \setminus \mathrm{HVD}(F_i)$ to every new range $f' \in \mathrm{HVD}(F_i) \setminus \mathrm{HVD}(F_{i-1})$ such that f' intersects f.

Suppose that $\mathrm{HVD}(F_{i-1})$ and $\mathscr{H}(F_{i-1})$ are already computed. To insert the next cluster C_i, we traverse $\mathscr{H}(F_{i-1})$ from root to a leaf. Simultaneously, we move in $\mathscr{T}(C_i)$, keeping track of the root x of the active subtree $\mathscr{T}_a(C_i,F_j)$ at the current level j of $\mathscr{H}(F_{i-1})$. When we reach a leaf of $\mathscr{H}(F_{i-1})$, we trace the boundary of the Voronoi region $\mathrm{hreg}_{F_i}(C_i)$, starting at the root x of $\mathscr{T}_a(C_i,F_i)$, and update $\mathscr{H}(F_{i-1})$ to become $\mathscr{H}(F_i)$.

In more detail, the procedure for level j is as follows: Let f be the face in $\mathrm{HVD}(F_j)$ that contains x. Suppose that f is deleted at step ℓ. If $\mathrm{d_f}(x,C_i) < \mathrm{d_f}(x,C_\ell)$, we search for the child f' of f, that has the same owner as f, and contains x; we move to the level ℓ, keeping x intact, and updating its face to be f'. Else we search for the root z of the (new) active subtree $\mathscr{T}_a(C_i,F_\ell \cup \{C_i\})$ (the procedure to do this is the same as for the conflict graph, see Section 3.3). If z is found, we move to level ℓ, replace x by z and the face f by $f' \subset \mathrm{hreg}_{F_\ell}(C_\ell)$ that contains z. If z is not found, the active subtree is empty, hence $\mathrm{hreg}_{F_i}(C_i) = \emptyset$.

Updating the history graph during step i takes time $O(\log n(N_i + K_i))$, where N_i is the number of edges of $\mathscr{T}(C_i)$ that do not belong to the active subtree $\mathscr{T}_a(C_i,F_i)$, and thus, they are eliminated by the breath-first search. K_i is the number of clusters in the sequence $\{C_1,\ldots,C_{i-1}\}$ that change the root of the active subtree as we move in the history graph level by level. The expectation of K_i is $O(\log i)$. Summing over all k steps gives us $O(k\log k)$. The total expected running time of the algorithm using the history graph is thus, $O(n\log n + k\log n\log k)$. The space complexity of the algorithm is expected $O(n)$.

3.4 A classic randomized incremental construction of the Hausdorff Voronoi diagram of arbitrary clusters

In this section we drop the assumption that clusters are pairwise non-crossing, and consider a classic variant of the randomized incremental construction framework, as applied to a family F of arbitrary clusters that may cross.

Allowing clusters to cross raises a major difficulty that Voronoi regions may be disconnected and the simple definition of a conflict of Section 3.3 is no longer adequate. We adapt a more standard definition for a conflict (see Definition 18). Ranges remain the same as in Section 3.3.

Let $S \subset F$, and let f be a face of $\mathrm{HVD}(S)$ such that f is in $\mathrm{hreg}_S(p)$, $p \in P$. Let C be a cluster in $F \setminus S$. Consider the Hausdorff bisector $\mathrm{b_h}(P,C)$ as truncated within f, and let $\mathscr{L}_v(f,C)$ denote the list of vertices of $\mathrm{b_h}(P,C) \cap \overline{f}$ in clockwise order as seen from p. Figure 3.15a illustrates $\mathrm{b_h}(P,C) \cap f$ as a blue bold polygonal line, however, $\mathrm{b_h}(P,C) \cap f$ may consist of several connected components.

Definition 18 (Conflict for arbitrary clusters)**.** A range f of $\mathrm{HVD}(S)$ is *in conflict* with a cluster $C \in F \setminus S$, if f intersects $\mathrm{hreg}_{F \cup \{C\}}(C)$. A *conflict* is a triple $(f, \mathscr{L}_v(f,C), C)$. List $\mathscr{L}_v(f,C)$ is called the *vertex list* of the conflict and it consists of the vertices of $\mathrm{b_h}(P,C)$ in $\overline{f}$.

Using this new definition of conflict we need to efficiently update the conflict graph after the insertion of C.

Before describing our algorithm to update the conflict graph, we need to introduce some notation regarding visibility-based decomposition of the Hausdorff Voronoi regions. Recall from Section 3.1 the definition of the visibility-based decomposition, and that the boundary of each face f of the visibility-based decomposition of consists of three parts:

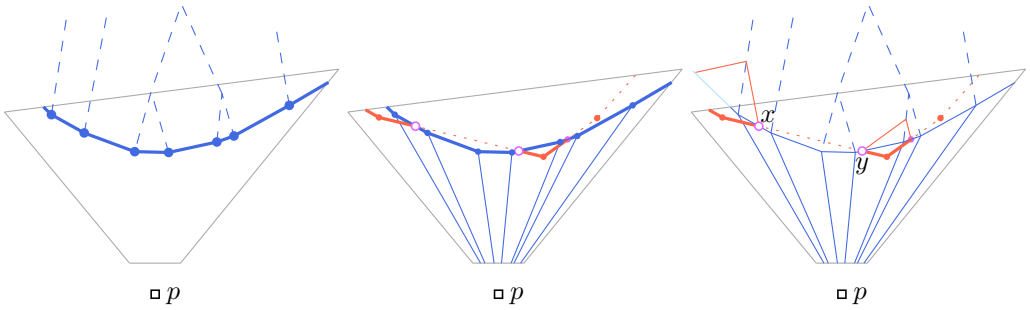
(1) a chain that is portion of $\mathscr{T}(P)$;

(2) a segment of $\mathrm{b_h}(P,Q), Q \in F$; we call it the *Hausdorff segment* of f;

(3) at most two edges of the visibility-based decomposition; we call them the *sides* of f.

In Figure 3.3, the sides are shown in bold, the Hausdorff segments are shown solid and the relevant portion of $\mathscr{T}(P)$ is shown dotted. For brevity we often refer to the chain induced by the Hausdorff segment and the (≤ 2) sides of f as the π-*chain* of f. Point p is called the *owner* of f.

Now we are ready to describe the algorithm to update the conflict graph.



(a) (b) (c)

Figure 3.15. A deleted range f (grey); (a) $\mathscr{L}_v(f,C)$ and the adjacent portion of $\mathscr{T}(C)$; (b) $\mathscr{L}_v(f,Q)$ (red), where Q is a cluster in conflict with f; the components of $\mathscr{L}_v(f,Q)$ of type 1 (red, bold) and of type 2 (red, dotted); ranges in $\mathscr{L}_r(f,C)$, (c) Illustration for tracing $\mathscr{L}_v(f,Q)$

3.4.1 Updating the conflict graph

When inserting cluster C in $\mathrm{HVD}(S)$ a number of ranges get deleted. For each deleted range f, we need to delete its conflicts and to create new conflicts with the new ranges that intersect f. The technical challenge of this task is caused by the non-constant complexity of $\mathscr{L}_v(f,C)$. That is, the number of new ranges related to f is non-constant, as well as the size of the vertex lists of the conflicts of f. We update the conflict graph in a way that all the work done is charged to conflicts that are created or deleted, to the vertices that are deleted from the vertex lists of conflicts, and to number of crossings of the inserted cluster. In the remaining of this section we give the details of this process. The algorithm to update the conflict graph is given as pseudocode in Figure 3.16.

Let f be a range deleted during the insertion of C of owner $p \in P$. Let $\mathscr{L}_r(f)$ denote the list of the (new) ranges of $\mathrm{HVD}(S \cup \{C\})$ that are owned by p and intersect f. The ranges in $\mathscr{L}_r(f)$ are ordered clockwise as their Hausdorff segments are seen from p. $\mathscr{L}_r(f)$ can be easily derived from $\mathscr{L}_v(f,C)$ while updating $\mathrm{HVD}(S)$ to $\mathrm{HVD}(S \cup \{C\})$. For a range $f' \in \mathscr{L}_r(f)$, the second side in this ordering is called the *right* side of f'.

Lines 3–16 in Figure 3.16 process a cluster $Q \in F \setminus (S \cup \{C\})$ that is in conflict with f. To determine the new conflicts of Q, we need to process the vertices of $\mathscr{L}_v(f,Q)$. We need the following distinction. Intersections between $\mathscr{L}_v(f,C)$ and $\mathscr{L}_v(f,Q)$ partition $\mathscr{L}_v(f,Q)$ into components of two types: *type 1:* portions that are closer to p than to C, and *type 2:* portions that are closer to C than to p. Figure 3.15b shows $\mathscr{L}_v(f,C)$ in blue thick lines, and $\mathscr{L}_v(f,Q)$ in red (solid and

dotted lines show its components of type 1 and 2 respectively).

The vertices of the components of type 1 stay in the vertex lists of new conflicts, while the vertices of the components of type 2 are discarded. When processing a component of type 1, we need to create conflicts between Q and the ranges of $\mathscr{L}_r(f)$ intersected by this component. Each vertex of the component must be placed in the appropriate vertex list. To do this, we subdivide the list of vertices of the component by the intersections with the sides of these ranges (Lines 8–11). When processing a component of type 2, we need (i) to determine the starting point of the next component of type 1 (Line 14), and (ii) to create the conflicts between Q and the ranges of C; these conflicts correspond to the face of $\mathrm{hreg}_{S\cup\{C,Q\}}(Q)$ that is incident to the starting point x of this component (Line 13).

Lines 6 and 11 can be performed by binary search in $\mathscr{L}_v(f,Q)$. Line 12 is implemented by a segment query: Let uv denote the roof of f', and let cluster R be a pair $\{p,c\}$, where c is the point in C, that is the farthest to points of uv. Clearly, $\mathrm{d_f}(u,Q) < \mathrm{d_f}(u,R)$ and $\mathrm{d_f}(v,Q) > \mathrm{d_f}(v,R)$; the answer to the segment query for segment uv in $\mathrm{FVD}(Q)$ is exactly the point x. Points x and y from Lines 12 and 14 respectively, are shown in Figure 3.15c.

Lemma 3.4.1. *The algorithm* Update-Conflict-Graph-(Arbitrary-Clusters) *(Figure 3.16) is correct.*

Proof. We need to show that after the algorithm is executed, all the conflicts of the new ranges are created. This is due to two observations: (1) Each face of $\mathrm{hreg}_{S\cup\{C,Q\}}(Q)$ borders the Hausdorff Voronoi region of some cluster from S (not only that of C). Thus, all the conflicts between Q and new ranges owned by C are encountered in Line 13. (2) For a range $f' \in \mathscr{L}_r(f)$, consider the points where $\mathscr{L}_v(f,Q)$ intersects the sides of f. If both points are closer to p than to C, then, since $\mathrm{hreg}_{S\cup\{Q\}}(Q)$ is convex, the chain induced by $\mathscr{L}_v(f,Q)$ does not intersect the roof of f. Hence, Line 8 gives a correct condition to find the next transition point from a component of type 1 to a component of type 2. □

Complexity analysis

We now analyze the time and space complexity. We follow the notation of Section 3.3. At each step i, cluster C_i is inserted into $\mathrm{HVD}(F_{i-1})$ as described in Section 3.4.1.

Lemma 3.4.2. *Updating the conflict graph after the insertion of cluster C_i requires time $O((A(C_i)+L(C_i)+Cr(C_i))\log n)$, where $A(C_i)$ is the number of conflicts created and deleted, $L(C_i)$ is the total number of vertices discarded from the vertex*

Algorithm *Update-Conflict-Graph-(Arbitrary-Clusters)*
(∗ Updates the conflict graph after insertion of $C \in F \setminus S$ into $\mathsf{HVD}(S)$ ∗)
1. **for** $f \in \mathsf{HVD}(S) \setminus \mathsf{HVD}(S \cup \{C\})$ **do**
2. Let $p \in P$ be the owner of f.
3. **for** each cluster $Q \in F \setminus (S \cup \{C\})$ that is in conflict with f **do**
4. Initialize v as the first vertex in $\mathcal{L}_v(f, Q)$.
5. Initialize f' as first range in $\mathcal{L}_r(f)$.
6. Let z be the intersection of $\mathcal{L}_v(f, Q)$ with the ray originating at p and passing through the right wall of f'.
7. **repeat**
8. **while** $\mathsf{d_f}(z, P) < \mathsf{d_f}(z, C)$ **do**
9. Assign the relevant part of $\mathcal{L}_v(f, Q)$ to the left of z, as $\mathcal{L}_v(f', Q)$.
10. Let f' be the next range in $\mathcal{L}_r(f)$.
11. Update point z with respect to new f'.
12. Find point x of intersection between the Hausdorff segment of f' and $\mathcal{L}_v(f, Q)$.
13. Starting at x, trace $\partial\mathsf{hreg}_{S\cup\{C,Q\}}(Q)$ inside $\mathsf{hreg}_{S\cup\{C\}}(C)$, creating new conflicts between Q and the ranges owned by C.
14. Starting at x, trace $\mathcal{L}_v(f, Q)$ until the next intersection y with $\mathcal{L}_v(f, C)$ is found.
15. Let z be the intersection between $\mathcal{L}_v(f, Q)$ and the right wall of the face containing y.
16. **until** all vertices in $\mathcal{L}_v(f, Q)$ or all ranges in $\mathcal{L}_r(f)$ are visited.

Figure 3.16. Algorithm to update the conflict graph after the insertion of cluster C

lists of conflicts, and $Cr(C_i)$ is the total number of crossings between C_i and the clusters in $F \setminus F_i$.

Proof. The algorithm iterates over all the deleted conflicts. At one such iteration, each range in $\mathscr{L}_r(f, C_i)$ is considered at most once, and if considered, a new conflict is created. Processing the components of type 1 (Lines 8–11) for all conflicts thus takes total time $O(A(C_i)\log n)$. All executions of Line 14 and Line 15 require in total $O((A(C_i)+L(C_i)+Cr(C_i))\log n)$ and $O(L(C_i)\log n)$ time, respectively. Each execution of Line 7, Line 12, or Line 13 requires $O(\log n)$ time: two former procedures are binary search, and the latter is a segment query. □

Lemma 3.4.3. *At step i, the total number of vertices in the vertex lists of all the conflicts is $O(n+Cr_i)$, where Cr_i is the total number of crossings between all pairs of clusters such that one cluster is in F_i and the other is in $F \setminus F_i$.*

Proof. Consider the vertex list $\mathscr{L}_v(f, Q)$ of the conflict between a cluster $Q \in F \setminus$

F_i and a range f of $\mathrm{HVD}(F_i)$ whose owner is $p \in P$. Members of $\mathscr{L}_v(f,Q)$ are Q-mixed and P-mixed vertices. The latter ones are crossing mixed vertices on $\mathrm{b_h}(P,Q)$, plus at most two non-crossing mixed vertices. The former ones are Q-mixed vertices in $\mathrm{HVD}(F_i \cup \{Q\})$, which are in total $O(|Q| + Cr(Q))$, where $Cr(Q)$ is the number of crossings between Q and clusters in F_i. Since the total number of points in all clusters is n, the claim follows. □

Theorem 3.4.1. *The Hausdorff Voronoi diagram of a family F of k clusters of total complexity n can be computed in $O((m+n\log k)\log n)$ expected time and $O(m+n\log k)$ expected space.*

Proof. To analyze the expected total number of conflicts created during the course of the algorithm, we need to estimate the expected number of ranges in $\mathrm{HVD}(R)$, where R is a random r-sample of F. This is proportional to the number of mixed Voronoi vertices in $\mathrm{HVD}(R)$ [66]. The number of non-crossing mixed vertices is in turn proportional to the total number of points in the sample, which is expected $O(nr/k)$. A crossing mixed vertex v induced by clusters $P,Q \in F$ appears in $\mathrm{HVD}(R)$ with probability at most $r(r-1)/(k(k-1))$ which is the probability that both P and Q are in R. Summing over all crossings of clusters in S, we have that the expected number of crossing mixed vertices in $\mathrm{HVD}(R)$ is $O(mr^2/k^2)$. Thus, the expected number of ranges in $\mathrm{HVD}(R)$ is $O(nr/k + mr^2/k^2)$. Applying Theorem 2.1.1, we conclude that the expected total number of conflicts created during the course of the algorithm is $O(n\log k + m)$; the expected space required by the algorithm is within same bound, since the additional space to store all the vertex lists of conflicts is $O(n+m)$, see Lemma 3.4.3.

Consider the number $L(C_i)$, the total number of the vertices deleted from the vertex lists of the conflicts during the step i. Expectation of $\sum_{i=1}^{k} L(C_i)$ is as well $O(n\log k + m)$: For each cluster $Q \in F$, there is $O(|Q|)$ non-crossing Q-mixed vertices at one time. Each such vertex can undergo at most k updates over the course of the algorithm. Since the sequence of insertions is random, the expected number of updates is $O(\log k)$; Summing over all clusters in F, we obtain $O(n\log k)$. Crossing mixed vertices contribute an additional $O(m)$ to this sum, since any crossing mixed vertex may be deleted at most once, and they are at most m in total. The sum $\sum_{i=1}^{k} Cr(C_i)$ is $O(m)$, since for each pair of clusters there is only one i such that their crossings are counted by $Cr(C_i)$. The claim is implied by Lemma 3.4.2. □

The above RIC algorithm can be adapted to work on a history graph within the same time and space bounds.

Summary

In this chapter, we have presented two different randomized incremental approaches to construct the Hausdorff Voronoi diagram of a family of k clusters, having n points in total.

In Section 3.2, we presented a RIC for the Hausdorff Voronoi diagram of non-crossing clusters, based on point location. The algorithm runs in expected $O(n \log n \log k)$ time and expected $O(n)$ space. It uses the Voronoi hierarchy data structure, which we augment with the ability to handle the features inherent to the Hausdorff Voronoi diagram of non-crossing clusters. These features are: sites of non-constant complexity, sites not enclosed in their Voronoi regions, and empty Voronoi regions. We also extended the Voronoi hierarchy to perform *parametric point location queries* in expected time $O(\log n \log k)$. We showed how to preprocess the farthest Voronoi diagram of the points of an individual cluster, building the centroid decomposition data structure, that can perform the point location and the *segment* queries in $O(\log n)$ time.

In Section 3.3, we presented the application of the classic RIC framework [26, 25] to the Hausdorff Voronoi diagram of non-crossing clusters. We proposed a simplified definition of conflict, that is based on the properties of the Hausdorff Voronoi diagram. This implies an algorithm to construct the diagram in expected $O(n \log n + k \log n \log k)$ time and deterministic $O(n)$ space. The complexity of our algorithm is considerably better than one that would follow from a straightforward application of the Clarkson-Shor framework.

In Section 3.4 we showed how apply the Clarkson-Shor framework to the Hausdorff Voronoi diagram of arbitrary clusters, which may possibly cross. We addressed the problem of computing a Voronoi diagram with disconnected regions and disconnected bisectors via the randomized incremental construction framework. The resulting algorithm runs in expected $O((m + n \log k) \log n)$ time and expected $O(m + n \log k)$ space, where m is the total number of crossings between pairs of clusters.

Chapter 4

Linear-time construction for the farthest-segment Voronoi diagram

In this chapter we present a linear-time algorithm to construct the farthest-segment Voronoi diagram, once the sequence of its faces at infinity is known. The chapter is based on:

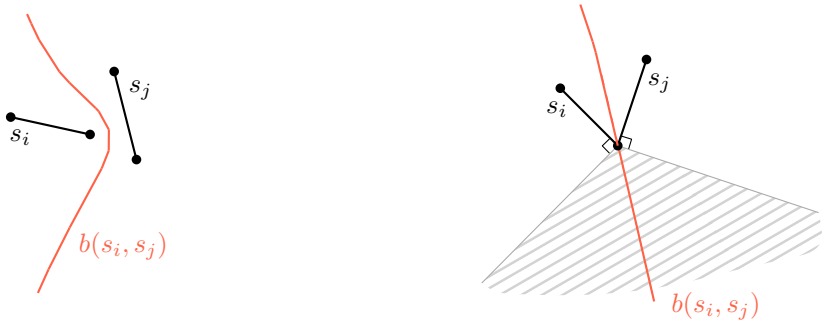
E. Khramtcova and E. Papadopoulou. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In Proc. Algorithms and Computation - 26th International Symposium, ISAAC*, pages 404–414, 2015.*

4.1 Preliminaries and Definitions

Let S be a set of n arbitrary line segments in $\mathbb{R}^2$; segments in S may intersect or touch at a single point. The distance between a point q and a line segment s_i is $d(q,s_i) = \min\{d(q,y) \mid y \in s_i\}$, where $d(\cdot,\cdot)$ denotes the Euclidean distance between two points.

The bisector of two segments $s_i, s_j \in S$ is $b(s_i,s_j) = \{x \in \mathbb{R}^2 \mid d(x,s_i) = d(x,s_j)\}$. For disjoint segments, $b(s_i,s_j)$ is an unbounded curve that consists of a constant number of pieces, where each piece is a portion of an elementary bisector between the endpoints and open portions of s_i, s_j, see Figure 4.1a. If two segments intersect transversally at point p their bisector consists of two such curves intersecting at p.

If segments s_i, s_j have a common endpoint then $b(s_i,s_j)$ contains a 2-dimensional region, see the shaded region in Figure 4.1b. Following standard conventions, see e.g. [8], the two-dimensional portion can be replaced by the piece of the angular bisector of s_i, s_j within this common region, obtaining a single curve, see the red curve in Figure 4.1b.



(a) (b)

Figure 4.1. The bisector of two segments s_i, s_j. Two cases: (a) s_i and s_j are disjoint, and (b) s_i and s_j share an endpoint p.

Note that the unbounded pieces of $b(s_i, s_j)$ are rays that are portions of bisectors of segment endpoints. We assume that such unbounded pieces are oriented towards infinity. For brevity, we refer to the direction of such a portion as a *direction of* $b(s_i, s_j)$. Bisector $b(s_i, s_j)$ has two directions if s_i, s_j are disjoint or have a common endpoint, and four directions if s_i, s_j intersect transversally.

The farthest Voronoi region of a segment s_i is $freg(s_i) = \{x \in \mathbb{R}^2 \mid d(x, s_i) > d(x, s_j), 1 \leq j \leq n, j \neq i\}$. The (non-empty) farthest Voronoi regions of the segments in S, together with their bounding edges and vertices, define a partition of the plane, called the *farthest-segment Voronoi diagram*, denoted $\mathrm{FVD}(S)$; see Figure 2.4a. Any maximally connected subset of a Voronoi region is called a *face*.

Recall from Section 2.2.1 that each segment corresponds to a double wedge in dual space, thus, S induces two arrangements of wedges, one of lower and one of upper wedges, respectively. The upper envelope of the former arrangement and the lower envelope of the latter arrangement correspond to the sequence of faces of $\mathrm{FVD}(S)$ at infinity. Recall also from Section 2.2.1 that these envelopes correspond to the *Gaussian map* of S, denoted $\mathrm{Gmap}(S)$. The Gmap provides a one-to-one correspondence between the faces $\mathrm{FVD}(S)$ at infinity.

4.2 The Farthest Voronoi Diagram of a Sequence

Consider the arrangements of upper and lower wedges of the segments in S in dual space. Let G be a sequence of arcs on the circle of directions K, such that the sequence corresponds to a pair of x-monotone paths in the dual space, one in the arrangement of upper and one in the arrangement of lower wedges. No arcs in G can overlap and no gaps can be present between consecutive arcs. Sequence

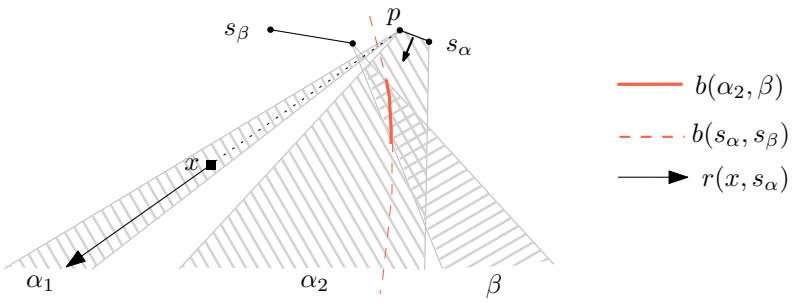


Figure 4.2. Arcs α_1, α_2 of segment s_α, and β of segment s_β; attainable regions $R(\alpha_1), R(\alpha_2), R(\beta)$ (shaded); segment bisector $b(s_\alpha, s_\beta)$ (red, dashed) and arc bisector $b(\alpha_2, \beta)$ (red, bold)

G corresponds to a pair of x-monotone paths in the arrangements of wedges in the same way as Gmap(S) corresponds to the two envelopes in these arrangements (see Section 2.2.1). We refer to such a sequence G as an *arc sequence*. When necessary, we refer to the endpoints of an arc as *arc-endpoints*, to differentiate from endpoints of segments. Throughout this chapter, given an arc α, let s_α denote the segment in S that induces α.

Similarly to Gmap(S), an arc sequence G consists of single-vertex arcs and segment arcs. For a segment arc α, the related hull direction of segment s_α, $\nu(s_\alpha)$, is always included in α; in dual space, it corresponds to the apex of the wedge of segment s_α. G may contain consecutive arcs of the same segment. The maximal union of such arcs is referred to as a *maximal arc*.

Let G^s denote a *simplified version* of an arc sequence G, where any consecutive arcs of the same segment s_α in G have been unified into a single maximal arc α. That is, for any arc α in G^s its neighbors are arcs of a different segment.

In the following we define the farthest Voronoi diagram of an arc sequence G, FVD(G). For $G =$ Gmap(S), FVD$(G) =$ FVD(S). The diagrams of such sequences appear as intermediate diagrams in the process of computing FVD(S), however, they do not correspond to any type of segment Voronoi diagram. We first define such a diagram and then present an arc deletion and arc insertion operation, which constitute the basis for our algorithm.

4.2.1 Defining the FVD(G)

Given an arc $\alpha \in G$ and a point $x \in \mathbb{R}^2$, $x \notin s_\alpha$, let $r(x, s_\alpha)$ denote the ray emanating from x in the direction $\overrightarrow{px}$, where p is the point in s_α closest to x; see Figure 4.2.

Definition 19. A point x, $x \notin s_\alpha$, is said to be attainable from α if the direction

of $r(x, s_\alpha)$ is contained in α. An endpoint of segment s_α is attainable from all its corresponding single-vertex arcs. All points in s_α are attainable from α if α is a segment arc. The locus of points attainable from arc α is called the *attainable region* of α, $R(\alpha)$. See Figure 4.2

Region $R(\alpha)$ is derived by the two rays in the direction of the arc-endpoints of α that emanate from the relevant endpoint(s) of s_α, see Figure 4.2. Within its attainable region, an arc α corresponds to a portion of segment s_α. If α is a single-vertex arc, it corresponds to one endpoint of s_α (see α_1, β in Figure 4.2). If α is a segment arc, it corresponds to both endpoints of s_α and at least one side of s_α represented by $\nu(s_\alpha)$ in α (see α_2 in Figure 4.2). An arc α should not be considered outside its attainable region. By the definition of an attainable region, we make the following remark.

Remark 4.2.1. *For any non-consecutive arcs $\alpha_1, \alpha_2 \in G$ induced by the same segment s_α, the interiors of their attainable regions are disjoint. Moreover, $R(\alpha_1) \cap R(\alpha_2) \setminus \{s_\alpha\} = \emptyset$.*

We define the distance between an arc α and a point $x \in \mathbb{R}^2$ as follows:

$$d(x, \alpha) = \begin{cases} d(x, s_\alpha), & \text{if } x \in R(\alpha); \\ -\infty, & \text{if } x \notin R(\alpha). \end{cases}$$

Definition 20. For two arcs α, β such that $s_\alpha \neq s_\beta$, their *arc bisector* $b(\alpha, \beta)$ is the interior of $b(s_\alpha, s_\beta) \cap R(\alpha) \cap R(\beta)$. If $s_\alpha = s_\beta$ and α, β are consecutive, then their *artificial bisector* $b(\alpha, \beta)$ is the interior of $R(\alpha) \cap R(\beta)$, which is the common boundary of $R(\alpha)$ and $R(\beta)$.

Note that arc bisector $b(\alpha, \beta)$ is always an open (possibly unbounded) connected portion of a curve.

Artificial bisectors involving same segment cannot intersect, see Remark 4.2.1.

The farthest Voronoi region of an arc α can now be defined in the ordinary way:

$$freg(\alpha) = \{x \in \mathbb{R}^2 \mid d(x, \alpha) > d(x, \gamma), \forall \text{ arc } \gamma \in G, \gamma \neq \alpha\}.$$

The subdivision of the plane derived by the farthest Voronoi regions of the arcs in G and their boundaries, is called the *farthest Voronoi diagram* of G, denoted $\mathrm{FVD}(G)$. Let $\overline{\mathrm{freg}}(\alpha)$ denote the closure of $\mathrm{freg}(\alpha)$. Assuming that $\cup_{\alpha \in G} \overline{\mathrm{freg}}(\alpha)$ covers the plane (see Lemma 4.2.1), we define the graph structure of $\mathrm{FVD}(G)$ as follows:

$$\mathscr{T}(G) = \mathbb{R}^2 \setminus \cup_{\alpha \in G} \mathrm{freg}(\alpha).$$

(a) (b)

Figure 4.3. Proper arc sequence $G = \{\alpha, \gamma, \delta\}$, and (a) FVD($G$); (b)FVD($G^s$)

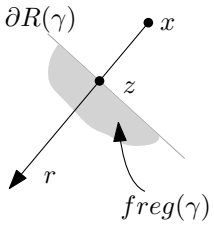


Figure 4.4. Illustration for the proof of Lemma 4.2.1

Figure 4.3a illustrates an arc sequence G and its diagram FVD(G). On the right G is shown as a circle of directions K; solid arrows indicate the arc-endpoints, and dashed arrows indicate hull directions of segments within the segment arcs. The left figure shows FVD(G): its graph structure $\mathcal{T}(G)$ is shown in red lines; its Voronoi regions are labeled with the corresponding arc (inside a little circle); the regions of arcs α and γ of segment s_α are shown shaded. For more examples of arc sequences and their diagrams, see also Figures 4.3b, 4.7.

Consider FVD(G^s), where G^s is the simplified version of G. A (maximal) arc α in G^s corresponds to a series of consecutive sub-arcs of s_α in G. The region freg(α) is split in FVD(G) into subregions by the artificial bisectors of these arcs. Figure 4.3b shows G^s and FVD(G^s) for the sequence G of Figure 4.3a: arcs α and γ of G are united into one maximal arc $\alpha\gamma$. We often alternate between G and G^s in this paper, as needed, and we often use the same notation, G, to denote both the original arc sequence and its simplified version G^s.

Lemma 4.2.1. *If* ∂freg(α), $\forall\alpha \in G^s$, *consists solely of portions of arc bisectors involving* α, *then* $\cup_{\alpha\in G}\overline{\text{freg}}(\alpha) = \mathbb{R}^2$.

Proof. Suppose for the sake of contradiction, that there is a point $x \in \mathbb{R}^2$, such that x does not belong to $\overline{\text{freg}}(\alpha)$ for any $\alpha \in G$, i.e., $x \notin \cup_{\alpha\in G}\overline{\text{freg}}(\alpha)$. Observe that x can not be attainable from any arc in G (otherwise, it would lie either in a Voronoi region of FVD(G), or on an arc bisector; the latter is always the border between two Voronoi regions of FVD(G)). Consider a ray r originating at x, see Figure 4.4,

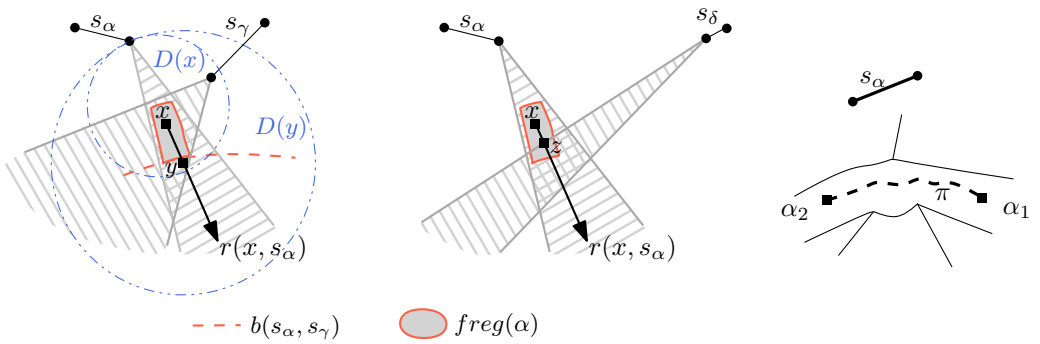


Figure 4.5. Illustration for the proof of Lemma 4.2.2

and let ϕ be the direction of r. Points on r far enough from x are attainable from the arc $\alpha \in G$ such that $\phi \in \alpha$. Thus, when moving on r starting at x, at some point z we for the first time cross the boundary of the attainable region of some arc $\gamma \in G$ (not necessarily $\gamma = \alpha$). Clearly $d(z,\gamma) \neq -\infty$ and $d(z,\beta) = -\infty$ for any arc $\beta \in G, \beta \neq \gamma$. Since the interior of segment xz is outside $R(\gamma)$, z lies on the boundary of $\mathrm{freg}(\gamma)$ (shaded in Figure 4.4). However, z is not attainable from any arc other than γ, thus, z is not on any arc bisector; a contradiction. □

Definition 21. If $\mathscr{T}(G^s)$ consists solely of portions of arc bisectors, then G, $\mathscr{T}(G)$, and $\mathrm{FVD}(G)$ are all called *proper*. See e.g. Figures. 4.3, 4.7.

The diagrams produced by our algorithms at the intermediate steps are always proper. Note, however, that for an arbitrary arc sequence G, $\mathscr{T}(G^s)$ may contain two-dimensional regions, or boundaries of attainable regions that are not bisectors.

Lemma 4.2.2. *For a proper arc sequence G, $\mathscr{T}(G)$ is a tree.*

Proof. Since G is proper, $\mathscr{T}(G)$ is a graph and each of its edges is a portion of the interior of an arc bisector. Let x be a point in $\mathrm{freg}(\alpha)$ for some $\alpha \in G$. We first prove that the entire unbounded ray $r(x, s_\alpha)$ is enclosed in $\mathrm{freg}(\alpha)$, thus, $\mathrm{freg}(\alpha)$ is unbounded.

Suppose first that $G = G^s$. Consider any arc γ, other than α, which is attainable from x. Since $d(x,\alpha) = d(x, s_\alpha) > d(x,\gamma) = d(x, s_\gamma)$, arc bisector $b(\alpha,\gamma)$, which is a portion of $b(s_\alpha, s_\gamma)$, cannot intersect $r(x, s_\alpha)$. This is easy to see by considering a disk $D(y)$ centered at a point y of $r(x, s_\alpha)$, see Figure 4.5a. As y moves along $r(x, s_\alpha)$, $D(y)$ enlarges and it must always intersect s_γ (see [8, Lemma 1]). Thus, no arc bisector $b(\alpha,\gamma)$, where γ is attainable from x, can bound $r(x, s_\alpha)$ as we walk on it towards infinity starting at x. Suppose now that an arc δ that is not attainable

from x, suddenly becomes attainable as we walk along $r(x, s_\alpha)$, because $r(x, s_\alpha)$ intersects $R(\delta)$ at a point z. If $d(z,\delta) < d(z,\alpha)$ then z and a neighborhood around it must remain in $\mathrm{freg}(\alpha)$. Since z is attainable from δ, arc bisector $b(\alpha,\delta)$ cannot intersect $r(z, s_\alpha)$ for the same reason as above. If on the other hand, $d(z,\delta) \geq d(z,\alpha)$ (see Figure 4.5b), then $z \in \mathscr{T}(G)$ without being on the interior of an arc bisector, contradicting our assumption about $\mathscr{T}(G)$. Thus, no interior point of an arc bisector involving α can bound $r(x, s_\alpha)$. Since $\mathscr{T}(G)$ contains only such points, it follows that the entire ray $r(x, s_\alpha) \subset \mathrm{freg}(\alpha)$. To remove the assumption that G equals its simplified version, imagine that G contains consecutive arcs of the same segment, and thus the artificial bisectors. However, any artificial bisector involving α, by its definition, can be either parallel to $r(x, s_\alpha)$, or it can coincide with it, or intersect it in an endpoint of segment s_α.

It remains to show that $\mathscr{T}(G)$ is connected. Suppose otherwise. Then there is a face in $\mathrm{FVD}(G)$, which is unbounded along two directions that belong to two different non-consecutive arcs of G. That is, there are non-consecutive arcs α_1, α_2, such that $\mathrm{freg}(\alpha_1)$ and $\mathrm{freg}(\alpha_2)$ are path-connected. But α_1 and α_2 should be of the same segment s_α, as $\mathrm{freg}(\alpha_1)$ and $\mathrm{freg}(\alpha_2)$ cannot be path connected if $s_{\alpha_1} \neq s_{\alpha_2}$. Then for any point p in $\mathrm{freg}(\alpha_1)$ and point q in $\mathrm{freg}(\alpha_2)$, there is a path π connecting them such that π is entirely contained in $\mathrm{freg}(\alpha_1) \cup \mathrm{freg}(\alpha_2)$, see Figure 4.5c. But then for any point x along π, ray $r(x, s_\alpha)$ would be entirely contained in $\mathrm{freg}(\alpha_1) \cup \mathrm{freg}(\alpha_2)$, i.e., arcs α_1, α_2 would be neighboring; a contradiction.

□

4.2.2 Subsequences and augmented subsequences of $\mathsf{Gmap}(G)$

An arc sequence G is called a *subsequence* of $\mathrm{Gmap}(S)$ if every arc of G entirely contains a corresponding arc of the $\mathrm{Gmap}(S)$ that are induced by the same segment. The arcs in G are expanded versions of the arcs in $\mathrm{Gmap}(S)$. The arcs in $\mathrm{Gmap}(S)$ as well as their expanded versions in G are called *original arcs*. The exact arcs of $\mathrm{Gmap}(S)$ are called *core arcs* or simply *cores*. The core of an arc β in G is denoted as β^*.

In the dual space, a subsequence G corresponds to an x-monotone path in the arrangement of lower wedges. Each maximal portion of a wedge on this path (dual to an arc $\beta \in G$) contains a part that appears in the upper envelope of the arrangement (this part is dual to β^*). In Figure 4.6b, dashed blue lines illustrate such path that corresponds to a subsequence of $\mathrm{Gmap}(S)$.

A sequence G' is called an *augmented subsequence* of $\mathrm{Gmap}(S)$ if the following holds: For every segment in S that contributes an arc to G', there is an original arc in G' induced by this segment. An augmented subsequence consists of *original*

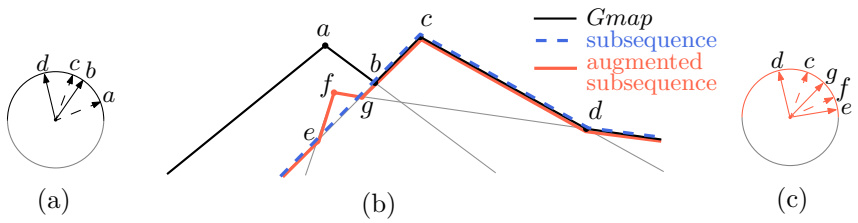


Figure 4.6. (a) The upper Gmap of S from Figure 2.4; (b) the dual arrangement of lower wedges with its upper envelope (black) and two other x-monotone paths, corresponding to a subsequence of Gmap(S) (blue, dashed) and to its augmented subsequence (red); (c) upper Gmap corresponding to the red path

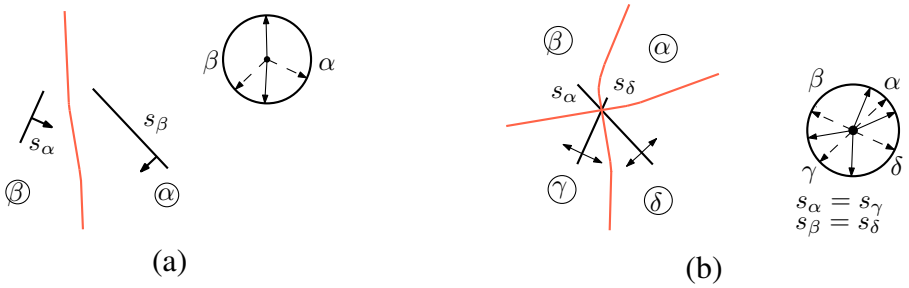


Figure 4.7. FVD(G) and G, where $G = \text{Gmap}(S)$ for (a) set $S = \{s_\alpha, s_\beta\}$ of disjoint segments; (b) set $S = \{s_\alpha, s_\delta\}$ of intersecting segments

arcs, which are expanded versions of the arcs in Gmap(S), and *new* arcs, which do not correspond to arcs of Gmap(S). An augmented subsequence G', which has the same original arcs as G, is said to be *corresponding to* G. In Figure 4.6b, the path shown in red solid lines corresponds to an augmented subsequence of Gmap(S). In this augmented subsequence, all arcs before point g are new arcs, and all arcs after point g are original arcs.

A starting point for our algorithms are subsequences of Gmap(S) consisting of two maximal arcs. That is, the simplified version of such subsequence has two arcs. We refer to such sequence as a *base subsequence of* Gmap(S). Obviously, such sequences are induced by two segments in S.

The simplified version of a base subsequence can be of two types, depending on whether these segments intersect: the diagram of two disjoint segments (Figure 4.7a), or the diagram of two intersecting segments without one of its faces (Figure 4.3b). In the (non-simplified) base subsequence, a single maximal arc may consist of several smaller consecutive sub-arcs of the same segment. In this case its Voronoi region is subdivided by the corresponding artificial bisectors, see Fig-

ure 4.3a.

Remark 4.2.2. *A base subsequence G of* $\mathrm{Gmap}(S)$ *is proper.*

Proof. The simplified version G^s of G consists of two arcs α, β, whose bisector either coincides with the segment bisector $b(s_\alpha, s_\beta)$ (see Figure 4.7a), or it is composed of the two of four branches of $b(s_\alpha, s_\beta)$ that meet at the only vertex of the corresponding farthest-segment Voronoi diagram (see Figure 4.3b). Thus the common boundary of $\mathrm{freg}(\alpha)$ and $\mathrm{freg}(\beta)$, which is the only edge of $\mathrm{FVD}(G^s)$, consists only of portions of arc bisectors. □

4.3 A deletion and insertion operation in a sequence of arcs

Throughout our algorithm to construct $\mathrm{FVD}(S)$ (see Section 4.4) we use a deletion and re-insertion operation for original arcs in sequences derived from $\mathrm{Gmap}(S)$. The deletion operation produces subsequences of $\mathrm{Gmap}(S)$ that are not necessarily proper. The insertion operation introduces *new* arcs, and creates augmented subsequences of $\mathrm{Gmap}(S)$, which are always proper. The insertion operation is tightly coupled with updating the Voronoi diagram computed so far.

For an arc sequence G, and an original arc $\beta \in G$, let $G \ominus \beta$ denote the arc sequence derived from G after deleting β from it. Respectively for an arc sequence G and an original arc $\beta \notin G$, we denote $G \oplus \beta$ the result of insertion of β in G. Let $\mathrm{FVD}(G) \oplus \beta$ denote the operation of inserting $\mathrm{freg}(\beta)$ in $\mathrm{FVD}(G)$, for an original arc $\beta \notin G$. $\mathrm{FVD}(G) \oplus \beta = \mathrm{FVD}(G \oplus \beta)$. Recall that an arc sequence G corresponds exactly to the cyclic sequence of Voronoi faces of $\mathrm{FVD}(G)$, and thus, similarly for $G \oplus \beta$ and $\mathrm{FVD}(G) \oplus \beta$.

4.3.1 Arc deletion

Let G be a subsequence of $\mathrm{Gmap}(S)$. G is derived from $\mathrm{Gmap}(S)$ by deleting arcs. When an arc β is deleted from G, the neighboring arcs α and γ *expand* over β, see Figure 4.8. Either both α and γ expand (see Figures 4.8a,c,d) or one expands while the other shrinks (see Figure 4.8b). During the expansion, α and γ may change from being a single-vertex arc to a segment arc (e.g., arc α in Figure 4.8b, or arc γ in Figure 4.8c). Since α and γ are original arcs, they both remain present in $G \ominus \{\beta\}$, and their common endpoint becomes $\nu(\alpha, \gamma)$.

The direction $\nu(\alpha, \gamma)$ is determined as follows:

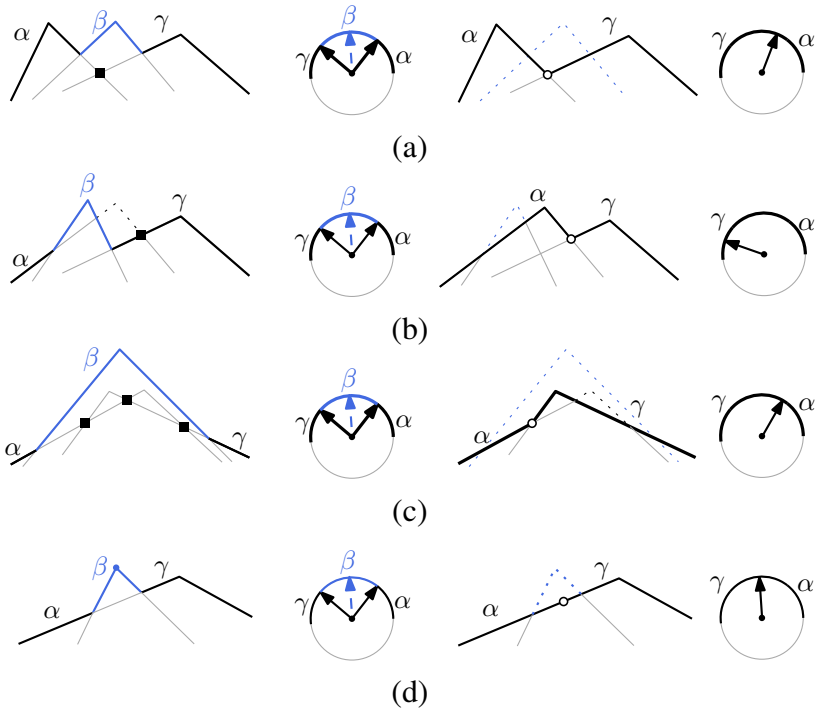


Figure 4.8. The deletion procedure: arcs α, β, γ in G, dual and primal image (left); the result of deletion of β (right)

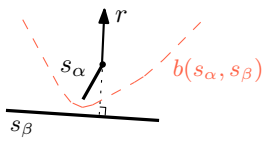


Figure 4.9. Segments s_α, s_β, and the ray r contained in the artificial bisector $b(\alpha,\gamma)$, $s_\alpha = s_\gamma$, after deletion of β

1. If $s_\alpha \neq s_\gamma$, then $\nu(\alpha,\gamma)$ is obtained from the directions of $b(s_\alpha, s_\gamma)$. Among the possibly four such directions (two, if segments s_α and s_γ are disjoint), $\nu(\alpha,\gamma)$ is the first direction of $b(s_\alpha, s_\gamma)$ encountered as we move on K from α^* to γ^*. In Figure 4.8c, $\nu(\alpha,\gamma)$ is shown as unfilled circle in the third figure from the left, and the three (possible) directions of $b(s_\alpha, s_\gamma)$ are shown as square marks in the first figure.
2. If $s_\alpha = s_\gamma \neq s_\beta$, then β must be a segment arc (easy to see in dual space). Then $\nu(\alpha,\gamma)$ is set to $\nu(\beta)$, the hull direction of s_β that must be in β, see Figure 4.8d. This is the direction of the artificial bisector $b(\alpha,\gamma)$ (see Remark 4.3.1).
3. If $s_\alpha = s_\beta = s_\gamma$, then let $\nu(\alpha,\gamma)$ be $\nu(\alpha,\beta)$. That is, two consecutive arcs β and γ of the same segment unite and become one arc γ.

Together with β, we store pointers to α and to γ, plus the direction $\nu(\alpha,\beta)$. This additional information is necessary for correct insertion of β during the insertion (and conquer) phase of our algorithms.

Remark 4.3.1. *The artificial bisector $b(\alpha,\gamma)$ ($s_\alpha = s_\gamma$) is (or contains) the ray perpendicular to s_β, emanating from the relevant endpoint of s_α and extending away from s_β. Figure 4.9 shows such ray r. See also Figure 4.3a: the artificial bisector of α and γ contains ray r_β.*

4.3.2 Arc insertion

Let G' be a proper augmented subsequence of $\mathrm{Gmap}(S)$ and let β be an original arc, $\beta \notin G'$. Let α,γ be two consecutive original arcs in G', such that β^* is between α^* and γ^*. A number of new arcs may lie between α and γ in G'. Inserting arc β in G' corresponds to inserting $\mathrm{freg}(\beta)$ in $\mathrm{FVD}(G')$ resulting in $\mathrm{FVD}(G') \oplus \beta = \mathrm{FVD}(G' \oplus \beta)$.

Suppose first, for simplicity, that α,γ are consecutive in G', i.e., no new arcs lie between them. We need to determine the arc-endpoints of β in $G' \oplus \beta$, i.e., $\nu(\alpha,\beta)$ and $\nu(\beta,\gamma)$. There are three cases to consider.

Case 1: $s_\alpha \neq s_\beta, s_\gamma \neq s_\beta$, and $\nu(\alpha,\gamma) \in \beta^*$. Then $\nu(\alpha,\beta)$ is the first direction of $b(s_\alpha, s_\beta)$ encountered as we walk on K starting at $\nu(\alpha,\gamma)$ and moving towards α^*. Symmetrically for $\nu(\beta,\gamma)$. Since α,β,γ are original arcs, and since β^* lies between α^* and γ^*, directions $\nu(\alpha,\beta)$ and $\nu(\beta,\gamma)$ must exist within α and γ respectively. The result is the sequence $\alpha\beta\gamma$ in place of $\alpha\gamma$. Figure 4.10a illustrates this case: arc ε is inserted between arcs δ and α.

Case 2: $s_\alpha \neq s_\beta, s_\gamma \neq s_\beta$, but $\nu(\alpha,\gamma) \notin \beta^*$. Then β^* is entirely contained in either α or γ. Suppose β^* is in γ. Consider two consecutive directions of $b(s_\beta, s_\gamma)$ that surround β^*. Let ν_1 be that one of them which lies between β^* and γ^*, and let ν_2 be the other one. Since $\beta^* \subset \gamma$, $\nu_1$is in γ. Let $\nu(\beta,\gamma) = \nu_1$. Direction $\nu(\alpha,\beta)$ is determined as follows:

(a) If $\nu_2 \notin \gamma$, then $\nu(\alpha,\beta)$ is a direction of $b(s_\alpha, s_\beta)$ as in case 1, and the result of the insertion is the same as in case 1.

(b) If $\nu_2 \in \gamma$, then γ is split into two parts by β. Let γ' be the part of γ that lies before ν_2 (in counterclockwise order); and $\nu(\alpha,\gamma)$ becomes $\nu(\alpha,\gamma')$. Let $\nu(\beta,\gamma') = \nu_2$. The insertion of β in G' results in $\alpha\gamma'\beta\gamma$ in place of $\alpha\gamma$, where γ' is a new arc.

In $\mathrm{FVD}(G') \oplus \beta$, $\mathrm{freg}(\beta)$ splits $\mathrm{freg}(\gamma)$ into two regions, $\mathrm{freg}(\gamma)$ and $\mathrm{freg}(\gamma')$.

Figure 4.10 illustrates the insertion of an arc ε of segment s_ε into $\mathrm{FVD}(G') = \mathrm{FVD}(G)$, $G = \alpha\gamma\delta$, of Figure 4.7b. The above and the below part show respectively case 1 and case 2b. Figure 4.10a shows $\mathrm{FVD}(G')$ superimposing segment s_ε (dashed), and G'. Fig 4.10b shows $\mathrm{FVD}(G') \oplus \varepsilon$, where $\mathrm{freg}(\varepsilon)$ is shaded, and $G' \oplus \beta$. Respectively for Figures 4.10c,d. For the case of Figures 4.10a,b $G' \oplus \varepsilon = \alpha\gamma\delta\varepsilon$, whereas for Figures 4.10c,d $G' \oplus \varepsilon = \alpha\gamma\delta\varepsilon\delta'$. Note that segment s_ε could contribute two arcs into G', but only one of them is inserted.

Case 3: $s_\alpha = s_\beta$ (symmetrically, if $s_\beta = s_\gamma$). In this case α is split in two parts by $\nu(\alpha,\beta)$, and one part becomes β. Note that $\nu(\alpha,\beta)$ has been determined when α and β become consecutive in a deletion operation. Note also that α,β cannot be neighbors in $\mathrm{Gmap}(S)$.

In $\mathrm{FVD}(G')$, $\mathrm{freg}(\alpha)$ is simply split in two parts by the artificial bisector corresponding to $\nu(\alpha,\beta)$; one part remains $\mathrm{freg}(\alpha)$ and the other part becomes $\mathrm{freg}(\beta)$. See such regions of α and γ in Figure 4.10, left.

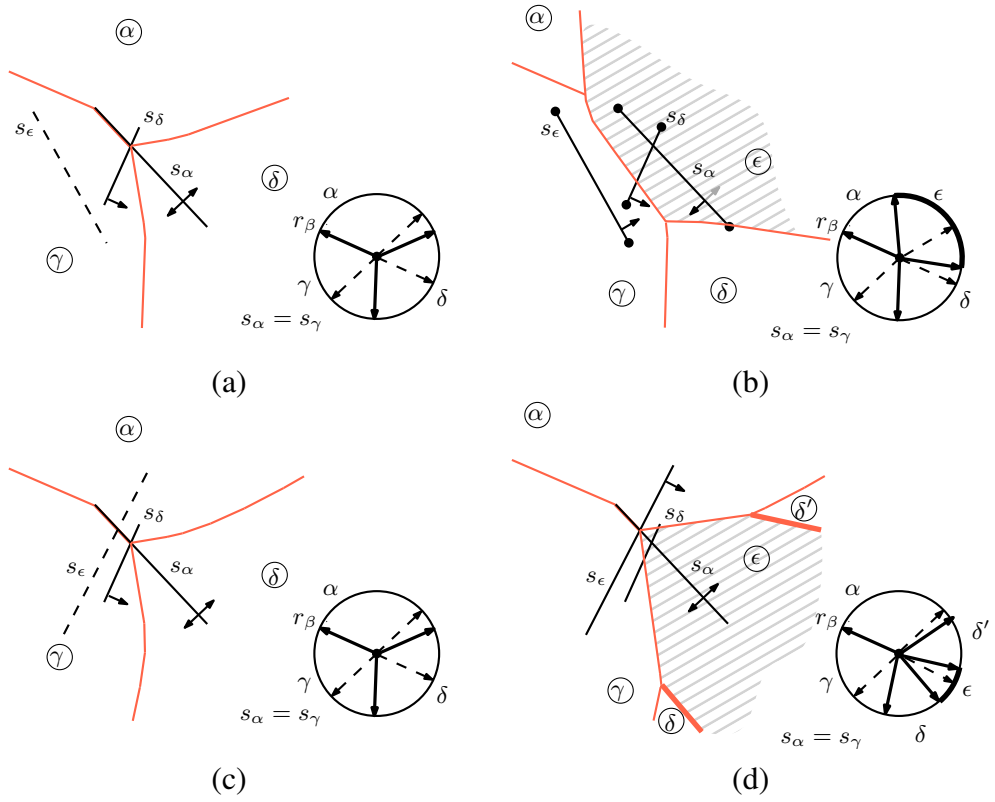


Figure 4.10. Insertion of arc ε in $\mathrm{FVD}(G'), G' = \alpha\gamma\delta$, assuming that the (stored) neighbors of ε are δ and α: case 1 (above) and case 2b (below)

We now remove the initial assumption that the arcs α and γ are consecutive in G', and we suppose that there are several new arcs between them. Some of these new arcs may be deleted by the insertion of β.

The procedure to insert β is as follows. Starting with any point in β^*, we move counterclockwise on G' until we determine an arc δ such that the relevant direction of $b(s_\delta, s_\beta)$ is in δ. Clearly, all the arcs encountered before δ (if any) must be deleted. Note that δ may equal α. Symmetrically, we move clockwise on G', until we encounter an arc ε. Since $\beta^* \in \mathrm{Gmap}(S)$, for any arc $\omega \in G'$ the direction of $b(s_\beta, s_\omega)$ is not in β^*.

Inserting β between δ and ε is analogous to cases (1) to (3), with δ and ε playing the role of α and γ respectively.

Lemma 4.3.1. *The diagram* $\mathrm{FVD}(G' \oplus \beta)$ *can be computed from* $\mathrm{FVD}(G')$ *in* $O(|\partial\mathrm{freg}(\beta)| + |\partial\mathrm{freg}(\omega')| + d(\beta))$ *time, if a new arc* ω' *is created by the insertion of* β. *If no new arcs are created,* $\mathrm{FVD}(G' \oplus \beta)$ *can be computed in* $O(|\mathrm{freg}(\beta)| + d(\beta))$ *time. Here* $d(\beta)$ *is the number of new arcs that get deleted*

by the insertion of β.

Proof. To insert β into $\mathrm{FVD}(G')$, we first determine the neighbors of β in $G' \oplus \beta$ by tracing G' in both directions starting from β^*. Since any visited arc gets deleted from the arc sequence, and since tracing requires $O(1)$ time per visited arc, this step requires overall $O(d(\beta))$ time.

Now we trace $\partial\mathrm{freg}(\beta)$. Suppose first that no new arcs are created by the insertion of β. This means, that the neighbors of β in $\mathrm{FVD}(G' \oplus \beta)$ are two different arcs. Thus there is at least one unbounded edge of $\mathrm{FVD}(G')$ that gets deleted by the insertion of β (e.g., any of the unbounded edges between the regions of the neighbors of β) We trace $\partial\mathrm{freg}(\beta)$ starting from this edge in the standard way, see e.g. [32]. The time complexity of such tracing is proportional to $|\partial\mathrm{freg}(\beta)|$ plus the total complexity of the Voronoi regions of the new arcs that get deleted. Such edges form a forest of total complexity $O(d(\beta))$. Thus the tracing requires $O(|\partial\mathrm{freg}(\beta)| + d(\beta))$ time.

Suppose now that the insertion of β caused the creation of a new arc ω'. In this case, no unbounded edge of $\mathrm{FVD}(G')$ is deleted, and we trace a number of edges on $\partial\mathrm{freg}(\omega')$ in order to find a starting point on $\partial\mathrm{freg}(\beta)$. Thus the tracing in this case requires $O(|\partial\mathrm{freg}(\beta)| + |\partial\mathrm{freg}(\omega')|)$ time.

□

The correctness of the insertion procedure is implied by the following.

Lemma 4.3.2. *Let G' be a proper augmented subsequence of* $\mathsf{Gmap}(S)$ *and let* β *be an original arc, $\beta \notin G'$. Then the sequence $G'' = G' \oplus \beta$ obtained by inserting* β *in G' is also a proper augmented subsequence of* $\mathsf{Gmap}(S)$.

Proof. Let α and γ be the two neighbors of β in G''. They may be original or new arcs. We first eliminate some simple cases.

If $s_\beta = s_\alpha$ or $s_\beta = s_\gamma$ (or both equalities hold), then the simplified versions of G'' and of G' coincide, and since G' is proper, so is G'' (see Definition 21).

If all the segments in S that contribute an arc to G' share the same endpoint, then the number of maximal arcs in G'' is two. By Remark 4.2.2, G'' is proper.

Therefore, from now on we assume that (1) $s_\alpha \neq s_\beta$ and $s_\gamma \neq s_\beta$; and (2) not all the segments in S that contribute an arc to G' share the same endpoint.

Since $\mathrm{FVD}(G')$ is proper, it is enough to show that every edge that lies on $\partial\mathrm{freg}(\beta)$ is a portion of an arc bisector. Equivalently, we need to show that all vertices of $\partial\mathrm{freg}(\beta)$ are proper Voronoi vertices, i.e., each of them is a point incident to at least three segment bisectors. Starting with the two unbounded edges, we trace $\partial\mathrm{freg}(\beta)$ from both sides considering pairs of edges and proving that at least one

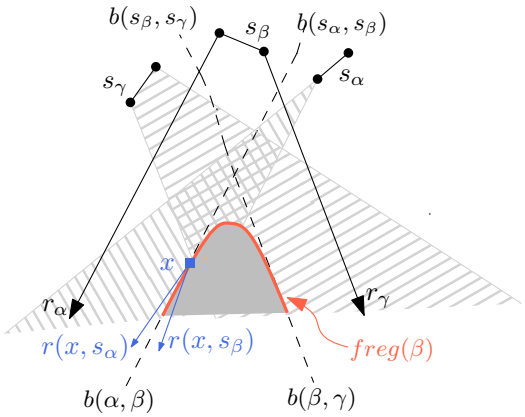


Figure 4.11. Illustrations for the proof of Lemma 4.3.2: three segments $s_\alpha, s_\beta, s_\gamma$; attainable regions $R(\alpha)$ and $R(\gamma)$ marked by a tiling pattern; $R(\beta)$ has rays r_α, r_γ on its boundary; $\mathrm{freg}(\beta)$ is shaded and its boundary is shown in red.

edge from the pair ends in a proper Voronoi vertex. Then we change that edge to its adjacent edge and repeat this argument. We consider the two edges as oriented in the direction of our traversal. At every such step we discover a new Voronoi vertex on $\partial\mathrm{freg}(\beta)$, thus the process will terminate. At the end, the two edges in our pair are adjacent, and this will prove the statement.

We now show, for a pair of (oriented) edges of $\partial\mathrm{freg}(\beta)$, that at least one of them ends in a proper Voronoi vertex. Consider, without loss of generality, the two unbounded edges of $\partial\mathrm{freg}(\beta)$, that are portions of $b(\alpha,\beta)$ and $b(\beta,\gamma)$. We need to prove, that at least one of $b(\alpha,\beta)$, $b(\beta,\gamma)$, as oriented from infinity, hits an edge of $\mathrm{FVD}(G')$ before it hits the boundary of the attainable region of an involved arc ($R(\alpha)$ or $R(\beta)$ for $b(\alpha,\beta)$; and $R(\beta)$ or $R(\gamma)$ for $b(\beta,\gamma)$).

Observe, that since G' is proper, $b(\alpha,\beta)$ cannot hit $\partial R(\alpha)$ before hitting $\partial\mathrm{freg}(\alpha)$. Symmetrically, for $b(\beta,\gamma)$ and $\partial R(\alpha)$.

The boundary of $R(\beta)$ consists of two rays emanating from s_β, one in the direction of $\nu(\alpha,\beta)$, and one in the direction of $\nu(\beta,\gamma)$. Let us denote them respectively by r_α and r_γ, see Figure 4.11a.

We show that $b(\alpha,\beta)$ cannot hit the part of r_α that is not an arc bisector. If s_α and s_β are disjoint, $b(\alpha,\beta)$ cannot hit r_α because of well-known visibility properties of segment bisectors; in particular, for any point x on $b(\alpha,\beta)$, the rays $r(x,s_\alpha)$ and $r(x,s_\beta)$ lie entirely at opposite sides of $b(\alpha,\beta)$, see the blue rays in Figure 4.11a.

If s_α and s_β share an endpoint p, $b(\alpha,\beta)$ is the common boundary of $R(\alpha)$ and $R(\beta)$, and the only point of r_α that is not on an arc bisector is its endpoint, which is p. Due to our assumption (2), not all the arcs in G' are attributed to p. Thus $p \notin \overline{\mathrm{freg}}(\alpha)$, and $b(\alpha,\beta)$ hits an edge of $\mathrm{FVD}(G')$ before it reaches p.

A symmetric argument holds for $b(\beta,\gamma)$ and r_γ.

Finally, suppose that $b(\alpha,\beta)$ hits r_γ before hitting $\partial\mathrm{freg}(\alpha)$. Then, since $\partial\mathrm{freg}(\gamma)$ and $R(\beta)$ are connected curves, $b(\beta,\gamma)$ must hit $\partial\mathrm{freg}(\gamma)$ before hitting $\partial R(\beta)$.

Thus, one of $b(\alpha,\beta)$, $b(\beta,\gamma)$ indeed ends in a proper Voronoi vertex. This completes our argument.

□

4.4 A linear-time algorithm to construct FVD(S)

We now augment the framework of Aggarwal et al. [3] for points in convex position with concepts and techniques from Sections 4.2, 4.3, and derive a linear-time algorithm to compute $\mathrm{FVD}(S)$, given $\mathrm{Gmap}(S)$. Let G be a subsequence of $\mathrm{Gmap}(S)$, and let G' be a corresponding proper augmented subsequence such that the complexity of G' is $O(|G|)$, where $|G|$ denotes the number of arcs of the sequence G. Our algorithm follows the flow of [51], which in turn follows [3].

1. Unite consecutive arcs of the same segment in G into single maximal arcs.
2. Color each arc of G *red* or *blue* by applying the following two rules:

   (a) For each 5-tuple F of consecutive arcs $\alpha,\beta,\gamma,\delta,\varepsilon$ in G, compute $\mathrm{FVD}(F')$ as follows: start with the sequence $\gamma\delta$, and consecutively insert the arcs β,ε,α (in this order) resulting in $\mathrm{FVD}(F')$. (F' is a possibly augmented version of F.) In $\mathrm{FVD}(F')$, if $\mathrm{freg}(\gamma)$ does not neighbor any region of segments s_α and s_ε, color γ red; else color γ blue.

   (b) For each series of consecutive blue arcs, color red every other arc, except the last one.
3. Let B (blue) be the sequence obtained from G by deleting all the red arcs. Recursively compute $\mathrm{FVD}(B')$. (B' is a possibly augmented version of B.)
4. Partition the red arcs into *crimson* and *garnet*: Re-color as *crimson* at least a constant fraction of the red arcs, such that for any two crimson arcs, if they were inserted in $\mathrm{FVD}(B')$, their Voronoi regions would not touch.
5. Insert the crimson arcs one by one in $\mathrm{FVD}(B')$ resulting in $\mathrm{FVD}(V')$.

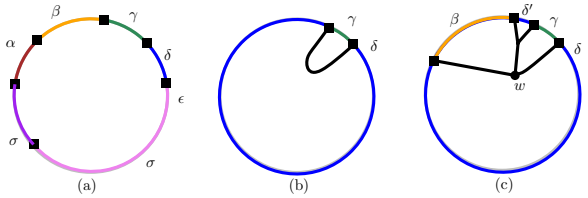


Figure 4.12. (a) a 5-tuple F; scheme of (b) FVD(F_2'), (c) FVD(F_3')

6. Let Gr (garnet) be the sequence obtained from G by deleting all blue and crimson arcs. Recursively compute FVD(Gr').
7. Merge FVD(V') and FVD(Gr') into FVD(G') so that $|G'|$ is $O(|G|)$.
8. For any arcs united in Step 1, subdivide their regions in FVD(G') into finer parts by inserting the corresponding artificial bisectors.

The recursion ends when the number of maximal arcs in G is at most five. Then FVD(G') can be directly computed in $O(1)$ time and also enhanced as indicated in Step 8. If all arcs in G are of the same segment, no diagram is generated but instead G is returned as a list of arcs. In this case, in Step 7, we obtain FVD(G') by inserting this list of arcs in FVD(V') one by one.

Step 2. Rules 2a and 2b guarantee that no two consecutive arcs in G are red and no three consecutive arcs in G are blue. The insertion order in Rule 2a guarantees that γ neighbors at most one new arc. This is ensured by the lemma below, which follows the spirit of [51, Lemma 8] and extends it to include new arcs.

Lemma 4.4.1. *No two consecutive arcs in G are red and no three consecutive arcs in G are blue.*

Proof. Suppose for the sake of contradiction, that there are two consecutive arcs γ,δ in G, which are both colored red. Clearly, they must both be colored red by Rule 2a (not by Rule 2b). Thus,there is a 6-tuple $\alpha,\beta,\gamma,\delta,\varepsilon,\sigma$ in G such that γ and δ satisfy Rule 2a. Since all arcs in G are maximal, any two consecutive ones must be induced by different segment. Further, if $s_\beta = s_\delta$ then by Remark 4.2.1, freg(β) and freg(δ) are not neighboring, and therefore freg(γ) in FVD(F') neighbors a region of some arc induced by either s_α or s_β, which contradicts coloring γ red by Rule 2a. Thus, $s_\beta \neq s_\delta$. Analogously $s_\gamma \neq s_\varepsilon$.

Consider the 5-tuple $F = \alpha,\beta,\gamma,\delta,\varepsilon$ (see Figure 4.12a), and FVD(F'), derived by applying Rule 2a. Building FVD(F') starts from FVD(F_2'), $F_2' = \gamma\delta$, shown schematically in Figure 4.12b. The insertion of β results in sequence $F_3' = \beta,\gamma,\delta$ or $F_3' = \beta,\delta',\gamma,\delta$. Since γ is colored red by Rule 2a, in FVD(F'), freg(γ) borders only regions of arcs of segments s_δ and s_β. Since each of the arc bisectors $b(\beta,\gamma)$, $b(\gamma,\delta)$

and $b(\delta,\varepsilon)$ is an unbounded portion of respectively $b(s_\beta,s_\gamma)$, $b(s_\gamma,s_\delta)$ and $b(s_\delta,s_\varepsilon)$, we consider the latter three curves oriented from the infinite part that is contaied in the corresponding arc bisector. If oriented in such way, bisector $b(s_\gamma,s_\delta)$ must meet $b(s_\beta,s_\gamma)$ first (before it meets $b(s_\delta,s_\varepsilon)$) forming a Voronoi vertex (vertex ω on Figure 4.12c). Applying the above argument to the 5-tuple $D=\beta,\gamma,\delta,\varepsilon,\sigma$, we have that the oriented $b(s_\delta,s_\varepsilon)$ must meet $b(s_\gamma,s_\delta)$ before any other related bisector, i.e., $b(s_\gamma,s_\delta)$ meets $b(s_\delta,s_\varepsilon)$ before $b(s_\beta,s_\gamma)$. A contradiction.

Rule 2b prevents three consecutive blue arcs. □

Step 4. To choose the crimson arcs we apply the *combinatorial lemma* of [3] on (a modified) $\mathscr{T}(B')$. The lemma states that for a binary tree T with n leaves embedded in $\mathbb{R}^2$, if each leaf of T is associated with a subtree of T and if for any two successive leaves these subtrees are disjoint, then in $O(n)$ time we can choose a set of leaves, whose number is at least a constant fraction of n and whose subtrees are pairwise disjoint. We associate each red arc β in G with a unique leaf of $\mathscr{T}(B')$, which would be the entry point for β in $\mathrm{FVD}(B')$. If the insertion of β splits an arc of B' in two, then we also add an artificial bisector to $\mathscr{T}(B')$ to serve as an entry point for β. The leaf in $\mathscr{T}(B')$ associated with β is in turn associated with the incident subtree of $\mathscr{T}(B')$, which would be intersected by $\mathrm{freg}(\beta)$, if β were inserted in $\mathrm{FVD}(B')$. In the following lemma we show that the modified $\mathscr{T}(B')$ satisfies the requirements of the combinatorial lemma, and has complexity proportional to $|B'|$ plus the number of red arcs $|R|$.

Lemma 4.4.2. *For any two successive red arcs in G, if they were inserted in* $\mathrm{FVD}(B')$*, the closures of their Voronoi regions would be disjoint.*

Proof. Suppose for the sake of contradiction that there is a pair of successive red arcs in G which do not satisfy the statement. By Rule 2b, there is either one or two blue arcs between them in G. In the former case, there is a 5-tuple $F=\alpha,\beta,\gamma,\delta,\varepsilon$ of the arcs in G, such that the arcs β and δ are colored red, and $\mathrm{freg}(\beta)$ and $\mathrm{freg}(\delta)$ are adjacent if β and δ are inserted in $\mathrm{FVD}(B')$ (by the insertion process of Section 4.2). Then the regions of β and δ must be adjacent in $\mathrm{FVD}(F')$, and they are obtained from F as described in Rule 2a. Then γ must satisfy Rule 2a and it must have been colored red during Step 2; a contradiction. If there are two blue arcs between the red arcs in question, the same argument works for one of the two corresponding 5-tuples. □

Step 7: Merging two diagrams. We obtain G' and $\mathrm{FVD}(G')$ by merging $\mathrm{FVD}(V')$ and $\mathrm{FVD}(Gr')$. To keep the complexity of G' within $O(|G|)$, we merge

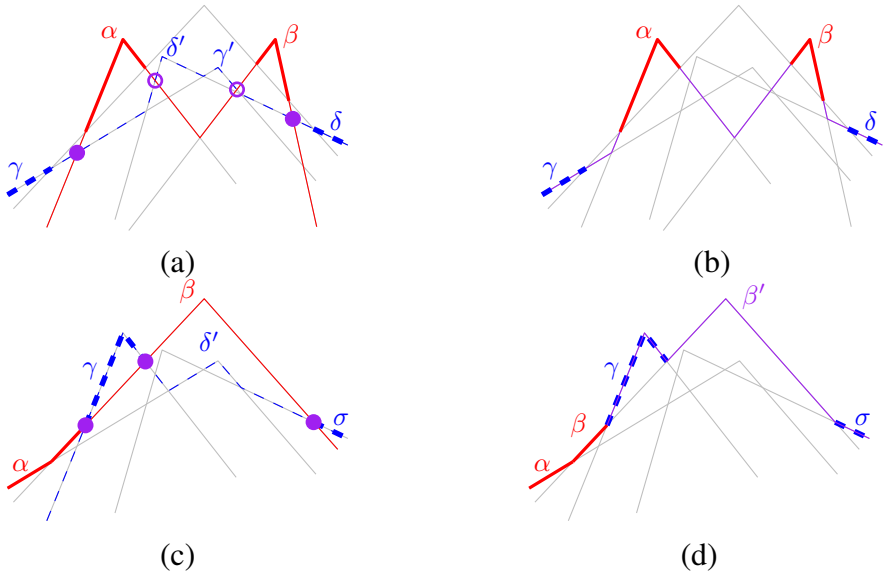


Figure 4.13. Two examples of merging. From left to right: two arc sequences A' (red) and B' (blue), and the result of merging them (purple); core portions are bold; (a) $A' = \alpha\beta$ and $B' = \gamma\delta'\gamma'\delta$ and (b) merging A' and B' gives $\gamma\alpha\beta\delta$; (c) $A' = \alpha\beta$, $B' = \gamma\delta'\sigma$, (d) merging such A' and B' gives $\alpha\beta\gamma\beta'\sigma$, and β' is a new arc, $\beta' \notin A', \beta' \notin B'$.

the two diagrams while discarding parts that are guaranteed to contain no original arcs. Merging is done in two steps: (1) identify starting points for the *merge curves* between the two diagrams, and (2) trace the merge curves. Step (2) can be done in the standard way [32]. Here, we identify starting points only for the merge curves that are related to original arcs of B'. Skipping a merge curve (or a pair of corresponding merge curves) has the effect of discarding the portion of one diagram that is bounded by it. This can be safely done because any portions of the diagram that are associated with only new arcs can not appear in FVD(S).

Below we give the details of obtaining G' and FVD(G') by merging FVD(A') and FVD(B'), where $A' = V'$ and $B' = Gr'$. In particular, we describe how to identify starting points for the merging curves in Step (1) of the merging process. For simplicity we use the dual space. Recall that an arc sequence corresponds to a pair of x-monotone paths in the arrangement of lower and upper wedges; see Figure 4.13, where the (lower) paths of A' and B' are shown in red and blue respectively (blue lines are also made dashed). Let U (resp., L) be the upper (resp., lower) envelope of A' and B' in the arrangement of lower (resp., upper) wedges. U (resp., L) consists of a series of connected components alternating between the

paths of A' and B'. The vertices incident to such connected components reveal the starting points of merge curves, which all can be easily identified in total time $O(|A'|+|B'|)$. Each merge curve has exactly two starting points in $U \cup L$. We identify all such starting points, and among them we choose only those whose incident component of B' contain the core portion of at least one original arc. In Figure 4.13, the core portions of original arcs of A' and of B' are shown in bold using red and blue color respectively (blue lines are made dashed), and the chosen starting points are shown as filled purple disks, whereas the ignored starting points are shown as (unfilled) purple circles. For each merge curve of interest either one or two starting points are identified. For some merge curves no starting points are identified; this happens only in case when the incident component of B' contains no original arcs and it will not appear in $\mathrm{FVD}(S)$. The description can be adapted to work directly on the arc sequences A', B' in primal space, similarly to [64].

The correctness and time complexity of the merging process is established by Lemmas 4.4.3 and 4.4.4.

Lemma 4.4.3. *$|G'|$ is $O(|A'|+|B'|)$. Moreover, G' contains all the original arcs of A' and B'; and the number of new arcs in G', which are neither in A' nor in B', is proportional to the total number of original arcs in A' and B'.*

Proof. (a) Consider the arrangement of lower wedges and its upper envelope U (the case of upper wedges is symmetric). Clearly, every original arc in A' and B' must entirely its core, which is dual to a portion of U. G' in dual space is a path, which lies above or on (but never below) the path dual to A', thus all original arcs from A' are in G'. For each original arc $\beta \in B'$, the core portion of β must be part of a maximal component of B' in U. For each such component, Step (1) of the merging process chooses the starting points of both merge curves incident to this component. Thus, $\beta \in G'$.

(b) A new arc that is neither in A' nor in B' is created when a portion of B' in G' splits an arc of A' (or vice versa). See Figure 4.13c,d, where the arc β' is exactly such an arc. Thus, the number of such new arcs in G' is bounded from above by the number of maximal components of B' and of A' that are contained in G'. This is in turn bounded by the number of original arcs in A' and B', which is $|G|$. □

Lemma 4.4.4. *Suppose that V' and Gr' are proper. Then $\mathrm{FVD}(G')$, obtained by merging $\mathrm{FVD}(V')$ and $\mathrm{FVD}(Gr')$, is also proper.*

Proof. It is enough to show that any merge curve consists solely of arc bisectors. Tracing the merge curve inside a single region $\mathrm{freg}(\alpha)$ of $\mathrm{FVD}(V')$ (resp., $\mathrm{FVD}(Gr')$) is equivalent to tracing the boundary of $\mathrm{freg}(\alpha)$ if it would be inserted in $\mathrm{FVD}(Gr')$ (resp., $\mathrm{FVD}(V')$). Thus, we can repeatedly apply the argument of the proof of Lemma 4.3.2 to any of the merge curves. □

Sequence G' is an augmented subsequence of Gmap(S) corresponding to G, and the recursive algorithm starts with $G = \text{Gmap}(S)$. Thus, at the end of the algorithm, the resulting arc sequence must be $G' = \text{Gmap}(S)$ (easy to see in dual space).

Lemma 4.4.5. *$|G'|$ is $O(|G|)$.*

Proof. Let $m = |G|$ and $S(m) = |G'|$. Since Step 4 is performed by applying the combinatorial lemma of [3], $|Gr| \leq q|R|$, where $0 < q < 1$ and $|R|$ is the number of red arcs ($|R| = |G| - |B|$). Thus, (following [3, 51]) there exist positive constants q_1 and q_2, $q_1 + q_2 < 1$, such that $|B| \leq q_1|G|$ and $|Gr| \leq q_2|G|$. At Step 4, at most one new arc is generated for every crimson arc inserted in B', thus, $|V'| = S(q_1 m) + O(m)$. At Step 7, $|G'| \leq |V'| + |Gr'| + O(m)$. Thus, $|G'| \leq S(q_1 m) + S(q_2 m) + O(m)$. Hence, $S(m) = O(m)$. □

Since the size of the augmented subsequences is always kept bounded, the time complexity can be analyzed similarly to [3]. We conclude:

Theorem 4.4.1. *Given* Gmap(S)*, the* FVD(S) *can be computed in $O(h)$ time, where h is the combinatorial complexity of* FVD(S).

Proof. Since $|G'|$ is $O(|G|)$, the time complexity of the algorithm can be analyzed similarly to [3]. Step 7 is performed in time $O(|V'| + |Gr'| + |G'|)$, which is $O(|G|)$. In step 4 the crimson arcs are chosen based on the combinatorial lemma in time $O(|G|)$. Their regions in FVD(V') are disjoint, thus, the total time to insert them is $O(|V'|)$, i.e., $O(|G|)$. Let $m = |G|$ and let $T(m)$ denote the time to compute FVD(G'). The recursive inequality of [3] holds, i.e., $T(m) \leq T(q_1 m) + T(q_2 m) + O(m)$, which implies $T(m) = O(m)$. □

Summary

In this chapter, we have presented an algorithm to construct the farthest-segment Voronoi diagram, given the sequence of its faces at infinity. This extends the paradigm of the existing linear constructions for tree-structured diagrams beyond the case of points in convex position [3]. Our goal is to extend the fundamental techniques known for points to more general objects so that the computation of their basic diagrams can be unified, despite their structural differences.

Concluding remarks

The technique presented in this chapter is also applicable to similar settings. These are the task to construct the order-$(k+1)$ subdivision within an order-k Voronoi region of segments, and the task to update a nearest-neighbor Voronoi diagram of segments after deletion of one site. See Section A.1 in the Appendix.

Chapter 5

Application of the cluster Voronoi diagrams to the stabbing circle problem

This chapter presents an algorithmic application of the Hausdorff and the farthest-color Voronoi digram to the stabbing circle problem. The material is based on the following publications:

M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for sets of segments in the plane. In LATIN 2016: Theoretical Informatics - 12th Latin American Symposium*, pages 290–305, 2016.*

M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for some sets of Delaunay segments. In Abstracts of the 32th European Workshop on Computational Geometry (EuroCG'16)*, pages 139–142, 2016.*

Some lemmas that are part of the proof of the correctness of our method and of the analysis of its running time are not to be considered part of this dissertation. We provide their proofs for completeness in Section A.2 of the Appendix. Similarly for the technical lemma that is the part of the proof of the running time for parallel segments, see Section A.2.1.

5.1 Preliminaries and Definitions

Let S be a set of n segments in the plane. This chapter discusses the stabbing circle problem for S, introduced in Section 1.4, in the view of its connection to the Hausdorff and the farthest-color Voronoi diagrams (see Definitions 4 and 5). The input families of point clusters for the diagrams are of a special form, as needed in the stabbing circle problem. In particular, clusters are pairs of endpoints of the

segments in S. Slightly abusing the notation of previous chapters, we refer to such diagrams simply as the Hausdorff and the farthest-color Voronoi diagram of S, although S is a set of segments. Below we give the necessary definitions, including some definitions from Chapters 1 and 3 as adapted to the setting of this chapter.

We assume segments in S to be in general position (segments have non-zero length, no three endpoints are collinear, and no four of them are cocircular).

A circle c is called a *stabbing circle* for S if exactly one endpoint of each segment of S is contained in the exterior of the closed disk (region) induced by c.

The *stabbing circle problem* for S consists of:

(1) answering whether a stabbing circle for S exists;

(2) reporting a representation (for the centers) of all the combinatorially different stabbing circles for S; and

(3) finding stabbing circles with minimum and maximum radius.

Note that the stabbing circle of minimum radius does not always exist. On the contrary, there are cases in which any stabbing circle can be shrunk by decreasing its radius or moving its center, but the "limit" circle is not stabbing anymore. In such cases, our task is to find this "limit" circle. The same may happen with the stabbing circles of maximum radius; refer to Lemma 5.3.8 for the details. Note also that our stabbing criterion uses only the segment endpoints, thus, S can be seen as a set of pairs of points, where a segment is simply a convenient representation for such a pair.

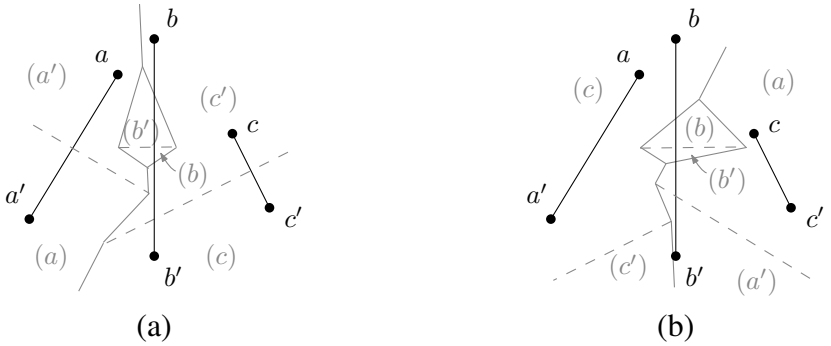
The *Hausdorff Voronoi diagram* of S is a partitioning of $\mathbb{R}^2$ into regions defined as follows:

$$\mathrm{hreg}(aa') = \{p \in \mathbb{R}^2 \mid \forall bb' \in S \setminus \{aa'\} \colon \max\{d(p,a), d(p,a')\} < \max\{d(p,b), d(p,b')\}\};$$
$$\mathrm{hreg}(a) = \{p \in \mathrm{hreg}(aa') \mid d(p,a) > d(p,a')\}.$$

Regions $\mathrm{hreg}(a)$ and $\mathrm{hreg}(a')$ are subregions of $\mathrm{hreg}(aa')$ (see Figure 5.1a). The *graph structure* of this diagram is:

$$\mathrm{HVD}(S) = \mathbb{R}^2 \setminus \bigcup_{aa' \in S} (\mathrm{hreg}(a) \cup \mathrm{hreg}(a')).$$

An edge of $\mathrm{HVD}(S)$ is called *pure* if it is incident to regions of two distinct segments; and it is called *internal* if it separates the subregions of the same segment. A vertex of $\mathrm{HVD}(S)$ is called *pure* if it is incident to three pure edges, and it is called *mixed* if it is incident to an internal edge. The pure vertices are defined by three distinct sites, and the mixed vertices by two distinct sites.



(a) (b)

Figure 5.1. (a) HVD(S), (b) FCVD(S). Pure and internal edges are represented in solid and dashed, respectively. The gray letters in parentheses label the respective regions.

The *farthest-color Voronoi diagram* is a partitioning of $\mathbb{R}^2$ into regions defined as follows:

$$\mathrm{fcreg}(aa') = \{p \in \mathbb{R}^2 \mid \forall bb' \in S \setminus \{aa'\}\colon \min\{d(p,a), d(p,a')\} > \min\{d(p,b), d(p,b')\}\};$$
$$\mathrm{fcreg}(a) = \{p \in \mathrm{fcreg}(aa') \mid d(p,a) < d(p,a')\}.$$

Its graph structure is:

$$\mathrm{FCVD}(S) = \mathbb{R}^2 \setminus \bigcup_{aa' \in S} (\mathrm{fcreg}(a) \cup \mathrm{fcreg}(a')).$$

The edges and vertices of $\mathrm{FCVD}(S)$ are characterized as *pure* or *internal*, and *pure* or *mixed*, analogously to those of $\mathrm{HVD}(S)$ (see Figure 5.1b). Let $\overline{\mathrm{hreg}}(\cdot)$ and $\overline{\mathrm{fcreg}}(\cdot)$ denote the closures of the respective regions.

When the segments in S are pairwise disjoint, clearly, they are non-crossing, thus the combinatorial complexity of $\mathrm{HVD}(S)$ is $O(n)$. It is not known whether this is the case for $\mathrm{FCVD}(S)$. For arbitrary segments, the complexity of both diagrams is $O(n^2)$. See Sections 1.2.1 and 1.2.3.

Definition 22. Given a point p, the *Hausdorff disk* of p, denoted $D_h(p)$, is the closed disk with center at p and radius $d(p,a)$, where $p \in \overline{\mathrm{hreg}}(a)$. Its radius is called the *Hausdorff radius* of p, and is denoted as $r_h(p)$. The *farthest-color disk* $D_f(p)$ and the *farthest-color radius* $r_f(p)$ of p are defined analogously.

The following lemma reveals the connection between the stabbing circle problem and the cluster Voronoi diagrams ($\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$).

Lemma 5.1.1. *Given a point p, there exists a stabbing circle centered at p if and only if $r_f(p) < r_h(p)$.*

Proof. Let c be a circle centered at p with radius r, and let D be the closed disk induced by c. Recall that c is a stabbing circle if and only if (1) each segment in S has an endpoint outside D; and (2) each segment in S has an endpoint in D. Condition (1) is equivalent to $r < r_h(p)$. Condition (2) is equivalent to $r_f(p) \leq r$. The claim follows. □

Now we are ready to define $\mathrm{FCVD}^*(S)$, which is the closure of the locus of the centers of all stabbing circles for S.

Definition 23. The $\mathrm{FCVD}^*(S)$ is the locus of points in $\mathbb{R}^2$ for which the farthest-color radius is less than or equal to the Hausdorff radius, i.e., $\mathrm{FCVD}^*(S) = \{p \in \mathbb{R}^2 : r_f(p) \leq r_h(p)\}$.

For any point on the boundary of $\mathrm{FCVD}^*(S)$, its Hausdorff radius equals its farthest-color radius. Note that this equality also holds for some points in the interior of $\mathrm{FCVD}^*(S)$. Any point p in the interior of $\mathrm{FCVD}^*(S)$ such that $r_h(p) = r_f(p)$ lies on an internal edge of both $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ that separate centers of stabbing circles of two combinatorially different types, refer to Section 5.2.2 for the details.

Lifting transformation for the stabbing circle problem. The stabbing circle problem can be solved by exploiting the *lifting transformation* (see Section 2.2.2) and the well known results on arrangements of 3D hyperlanes [38]. The lifting transformation maps the pair of endpoints of each segment in S to a pair of planes in 3D, where the lower and upper envelope of such a pair form a *lower* and *upper wedge*, respectively. The $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ correspond to the upper envelope of all lower wedges, and to the lower envelope of all upper wedges, respectively (See Properties 2.2.2 and 2.2.3). Thus, $\mathrm{FCVD}^*(S)$ corresponds to the locus of points below $\mathrm{HVD}(S)$ and above $\mathrm{FCVD}(S)$. As shown by Edelsbrunner et al. [38], this is a set of $O(n^2)$ convex cells in 3D with $O(n^2)$ total complexity, and it can be computed in $O(n^2)$ time and space. Moreover, this set is a representation of all combinatorially different *stabbing planes* for S' [38], where S' is the set of 3D segments obtained by lifting the endpoints of each segment in S onto the unit paraboloid, and connecting them by a straight line. We observe that a stabbing circle for S can be transformed into a stabbing plane for S' and vice versa. Thus, we obtain the following result which, to the best of our knowledge, has not been explicitly stated anywhere before:

Theorem 5.1.1. *The stabbing circle problem for a set S of n arbitrary segments can be solved in $O(n^2)$ time and space.*

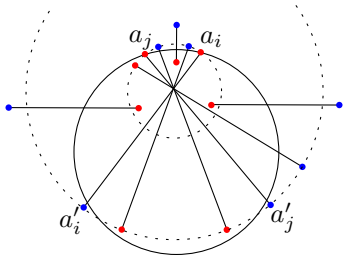


Figure 5.2. A set with $\Theta(n^2)$ combinatorially different stabbing circles, and the stabbing circle defined by $\{a_i, a_j\}$.

Claverol [27] showed that the set S might have $\Theta(n^2)$ combinatorially different stabbing circles; see Figure 5.2. In the construction, each pair $\{a_i, a_j\}$ of points in the upper arc defines a stabbing circle that leaves the endpoints in the upper arc between a_i and a_j outside the circle. Hence, the $\Theta(n^2)$ stabbing circles defined in this way are combinatorially different. We observe that it is possible to perform a small perturbation so that the general position assumptions are satisfied.

5.2 Properties of HVD(S), FCVD(S), and FCVD*(S)

In this section we investigate structural properties of the geometric structures involved in the stabbing circle problem. First, in Section 5.2.1, we list basic properties of the Hausdorff and the farthest-color diagrams. These are later used to derive the structural properties of FCVD*(S) and the correctness of our solution to the stabbing circle problem. The structure of the Hausdorff diagram has been well known, see e.g., [38, 66, 62], however, this has not been the case for the farthest-color diagram. We use Section 5.2.1 to derive such basic structural properties for FCVD(S). Then, in Section 5.2.2 we investigate the structure of FCVD*(S), we characterize its faces and their complexity linking them to features of HVD(S) and FCVD(S). We show that every face of FCVD*(S) corresponds to a unique combinatorially distinct solution of the stabbing circle problem. Finally, in Section 5.2.3, we complete the structural complexity analysis of the FCVD*(S) and count its faces that are not related to vertices of HVD(S) or FCVD(S). The structural properties derived in this section are later used in Section 5.3 to obtain our algorithm that computes all the combinatorially distinct stabbing circles for S.

5.2.1 Properties of HVD(S) and FCVD(S)

We list some structural properties of the Hausdorff and the farthest-color diagrams, which are used by our algorithms. First, a *visibility property* of both diagrams is summarized in the following lemma. For $\mathrm{HVD}(S)$, this property follows directly from [66, Property 2] (item (a)). We prove an equivalent property for $\mathrm{FCVD}(S)$ (item (b)).

Lemma 5.2.1. *Consider* $\mathrm{hreg}(a)$ *and* $\mathrm{fcreg}(a)$*, where* $aa' \in S$ *and* $|S| > 1$.

(a) For a point p in $\mathrm{hreg}(a)$*, the segment ap intersects* $\partial\mathrm{hreg}(a)$ *exactly once and the intersection point lies on an internal edge of* $\mathrm{HVD}(S)$*. For a point q on an internal edge of* $\partial\mathrm{hreg}(a)$*, the segment aq does not intersect* $\mathrm{hreg}(a)$.

(b) For a point p in $\mathrm{fcreg}(a)$*, the segment ap intersects* $\partial\mathrm{fcreg}(a)$ *exactly once and the intersection point lies on a pure edge of* $\mathrm{FCVD}(S)$*. For a point q on a pure edge of* $\partial\mathrm{fcreg}(a)$*, the segment aq does not intersect* $\mathrm{fcreg}(a)$.

Proof. We only prove item (b). If $|S| > 1$, $a \notin \overline{\mathrm{fcreg}}(a)$, thus, ap intersects $\partial\mathrm{fcreg}(a)$ at least once. Suppose ap intersects $\partial\mathrm{fcreg}(a)$ more than once, and let x and y be the first two intersection points that are encountered when moving from p to a. Points x and y are on $\partial\mathrm{fcreg}(a)$ and the segment xy is outside $\mathrm{fcreg}(a)$. Thus, $D_f(x)$ and $D_f(y)$ contain at least one endpoint of every segment in S, their boundary passes through a, and a' is outside both disks. Let w be a point on xy and let $D(w)$ be the closed disk centered at w passing through a. Since the distance of any point on the line through x,y from a increases as we move away from a, $D_f(y) \subset D(w) \subset D_f(x)$. But then $D(w)$ must contain all points in $D_f(y)$, $a' \notin D(w)$, and $a \in \partial D(w)$. Thus, $D(w) = D_f(w)$; hence, $w \in \mathrm{fcreg}(a)$. We obtain a contradiction.

The second claim in item (b) follows directly from the first one by considering a point $p \in \mathrm{fcreg}(a)$ that is infinitesimally close to q. □

Figure 5.1 illustrates a Hausdorff and a farthest-color diagram. Every component of a Hausdorff region $\mathrm{hreg}(aa')$ contains exactly one internal edge [66, Property 3] (see the dashed lines in Figure 5.1a). In the following lemmas we show properties of a similar nature for $\mathrm{FCVD}(S)$.

Lemma 5.2.2. *Any bounded face of* $\mathrm{fcreg}(aa')$ *contains an internal edge.*

Proof. Let f be a bounded face of $\mathrm{fcreg}(aa')$. Suppose for the sake of contradiction that f contains no internal edge. Then f is a face of $\mathrm{fcreg}(a)$, for an endpoint a of aa', such that ∂f consists solely of pure edges. Let p be a point in f and consider

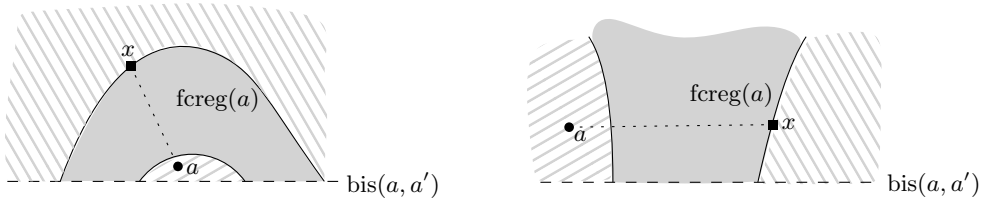


Figure 5.3. Illustration for Lemma 5.2.3: two variants of an impossible situation.

the ray r from a through p. Let q be the first intersection point between r and ∂f, as we move on r starting at p away from a (such point exists because f is bounded). Since f consists solely of pure edges, q is a point on a pure edge and $pq \in f$, yielding a contradiction to the last claim of Lemma 5.2.1b. □

Lemma 5.2.3. *For any face of* $\mathrm{fcreg}(a)$*, the portion of its boundary that is formed by pure edges is connected. (See the solid lines in Figure 5.1b.)*

Proof. Suppose $|S| > 1$ as otherwise the claim holds trivially. Let f be a face of $\mathrm{fcreg}(a)$. Clearly, any internal edge on ∂f is a portion of $\mathrm{b_h}(a,a')$. Since $|S| > 1$, $a \notin \mathrm{fcreg}(a)$, but a lies in the same halfplane induced by $\mathrm{b_h}(a,a')$ as $\mathrm{fcreg}(a)$. This implies that a lies in a region of the plane bounded by $\mathrm{b_h}(a,a')$ and one of the connected components of pure edges of ∂f (such regions are shown with tiling patterns in Figure 5.3). If ∂f contained more than one such connected components of pure edges, then any point x on any additional component would violate Lemma 5.2.1b. See Figure 5.3. □

The following property of $\mathrm{FCVD}(S)$ is only used in Section 5.4.

Lemma 5.2.4. $\mathrm{FCVD}(S)$ *has* $O(n)$ *unbounded faces.*

Proof. We observe that each unbounded face of $\mathrm{FCVD}(S)$ corresponds exactly to one face of the *farthest-segment Voronoi diagram* of S, see Sections 1.2.4 and 2.2.1. To make the correspondence 1-1 we remove the internal unbounded edges of $\mathrm{FCVD}(S)$ and merge the incident faces of $\mathrm{fcreg}(a)$ and $\mathrm{fcreg}(a')$, for some $aa' \in S$, into one face of $\mathrm{fcreg}(aa')$. Then a face f in $\mathrm{FCVD}(S)$ is unbounded in a direction ϕ if and only if a face f' in $\mathrm{FsVD}(S)$ is unbounded in the same direction ϕ. See [64] for the properties of the faces of $\mathrm{FsVD}(S)$ at infinity, which are identical to those of $\mathrm{FCVD}(S)$. The total number of faces in $\mathrm{FsVD}(S)$ is $O(n)$ [8], thus, the same holds for the unbounded faces of $\mathrm{FCVD}(S)$. Note that a component of $\mathrm{fcreg}(aa')$ can contain at most two unbounded portions of $\mathrm{b_h}(a,a')$, thus, having removed the internal unbounded edges of $\mathrm{FCVD}(S)$ has no effect on the derived bound. □

5.2.2 Properties of $\mathrm{FCVD}^*(S)$

In this section, we first characterize the boundary of $\mathrm{FCVD}^*(S)$. Then we define *faces* of $\mathrm{FCVD}^*(S)$, and we observe that they are in one-to-one correspondence with the combinatorially different solutions for the stabbing circle problem for S. Finally, we characterize faces of $\mathrm{FCVD}^*(S)$ with respect to their intersection with $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$, which is the basis of our algorithm to compute $\mathrm{FCVD}^*(S)$ (and thus to solve the stabbing circle problem) presented in Section 5.3.

We refer to the connected components of $\mathrm{FCVD}^*(S)$ as *components* of $\mathrm{FCVD}^*(S)$. Notice that a component is unbounded in a direction ϕ if and only if there exists a stabbing line for S that is orthogonal to ϕ. We proceed with describing the boundary of the components of $\mathrm{FCVD}^*(S)$.

The vertices of $\partial\mathrm{FCVD}^*(S)$ are incident to edges of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$. Indeed, any vertex of $\mathrm{FCVD}^*(S)$ is caused by switching the region of either $\mathrm{HVD}(S)$ or of $\mathrm{FCVD}(S)$ (in a traversal of $\partial\mathrm{FCVD}(S)$), which happens exactly when $\partial\mathrm{FCVD}^*(S)$ meets an edge of one of the diagrams. Thus we distinguish three types of vertices of $\partial\mathrm{FCVD}^*(S)$: (1) vertices incident to pure edges of $\mathrm{HVD}(S)$, called *h-vertices*; (2) vertices incident to pure edges of $\mathrm{FCVD}(S)$, called *fc-vertices*; and (3) vertices incident to internal edges of both diagrams, called *mixed vertices*.

The following lemma implies that each mixed vertex of $\partial\mathrm{FCVD}^*(S)$ is a mixed vertex of either $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, hence, the name for such vertices. Moreover, this lemma and its corollary lead to the definition of a face of $\mathrm{FCVD}^*(S)$, which will then be treated as an atomic piece of $\mathrm{FCVD}^*(S)$.

Lemma 5.2.5. *Point $p \in \mathrm{b_h}(a,a')$ is in $\mathrm{FCVD}^*(S)$ if and only if p lies on an internal edge of both $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$. If p is on $\partial\mathrm{FCVD}^*(S)$, then p is a mixed vertex of either $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$.*

Proof. Since $p \in \mathrm{b_h}(a,a')$, either both a,a' are outside $D_h(p)$ or they lie on its boundary. Symmetrically, either both a,a' are in the interior of $D_f(p)$ or on its boundary.

Suppose $p \in \mathrm{FCVD}^*(S)$. Then $D_f(p) \subseteq D_h(p)$. By the above argument, $D_f(p) = D_h(p)$ and both a,a' lie on the boundary of this disk. Therefore, $r_h(p) = r_f(p) = pa = pa'$, and the claim follows.

Suppose that p lies on an internal edge of both $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ that separates the respective regions of a and of a'. Since p lies on such edge of $\mathrm{HVD}(S)$, $r_h(p) = pa$, and since p lies on such edge of $\mathrm{FCVD}(S)$, $r_f(p) = pa$. Thus $r_h(p) = r_f(p)$, hence, $p \in \mathrm{FCVD}^*(S)$.

Now we prove the second part of the statement. Since $p \in \mathrm{b_h}(a,a')$ is on $\partial\mathrm{FCVD}^*(S)$, $D_h(p) = D_f(p)$, and the boundary of this disk passes through a,a'.

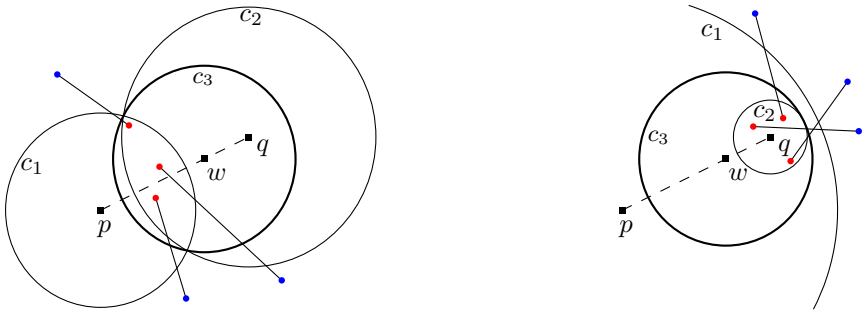


Figure 5.4. Illustration for the proof of Lemma 5.2.6.

It is easy to see that this boundary also passes through an endpoint c of some segment $cc' \in S$ (otherwise, the center of this disk could move in any direction while still being in FCVD*(S), contradicting the fact that $p \in \partial\text{FCVD}^*(S)$). If the other endpoint c' of cc' is inside this disk, then p is a mixed vertex of HVD(S), and if it is outside, p is a mixed vertex of FCVD(S). □

Corollary 5.2.1. *No mixed vertex of* HVD(S) *or* FCVD(S) *may lie in the interior of* FCVD*(S).

Proof. Suppose p is a mixed vertex of HVD(S) such that $p \in \text{FCVD}^*(S)$. Let e denote the internal edge of HVD(S) incident to p; e is a portion of $\text{b}_\text{h}(a,a')$, for some $aa' \in S$. Since p is a vertex of e, there is a point p' on $\text{b}_\text{h}(a,a')$ infinitesimally close to p such that p' does not belong to e. Observe that $r_h(p') < p'a$, and both a,a' are outside the Hausdorff disk of p'. This implies that p' is not in FCVD*(S), and therefore p is not in the interior of FCVD*(S). A similar argument (with $r_f(p') > p'a$) proves the case of p being a mixed vertex of FCVD(S). □

The common portion of an internal edge of HVD(S) and one of FCVD(S) within a component of FCVD*(S) is called an *internal edge* of FCVD*(S). Internal edges partition the components of FCVD*(S) into disjoint open *faces* of FCVD*(S). Thus there are different types of components of FCVD*(S): (1) Components that consist of several faces separated by internal edges, called (*multiple-face components*); (2) Components that contain no internal edges; the interior of each such a component is also a face of FCVD*(S), and the component is called a *single-face component*.

Lemma 5.2.6. *Two stabbing circles are combinatorially different if and only if their centers lie in different faces of* FCVD*(S).

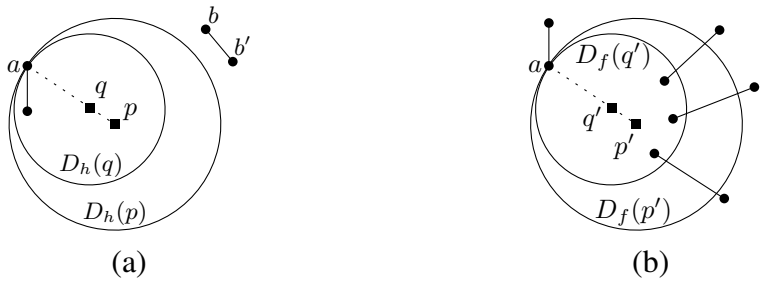


Figure 5.5. Illustration for the proof of Lemma 5.2.7.

Proof. Let c_1 and c_2 be two combinatorially different stabbing circles, and let respectively p and q be their centers. There is a segment $aa' \in S$ such that a is enclosed in c_1, and a' is enclosed in c_2. Observe that p and q lie in different halfplanes with respect to $\mathrm{b_h}(a,a')$. If p and q were in the same face f of FCVD*(S), then there would be a path π connecting p and q such that π lies entirely in f. Since p and q are separated by $\mathrm{b_h}(a,a')$, path π would need to cross $\mathrm{b_h}(a,a')$ at a point t. However, point t cannot lie inside f due to Lemma 5.2.5, which would derive a contradiction.

Next, suppose that the stabbing circles c_1 and c_2 are combinatorially equivalent. Let w be a point on the segment pq. We show that w is the center of a stabbing circle c_3 that is combinatorially equivalent to c_1 and c_2. Indeed, if c_1 and c_2 intersect in two points, then let c_3 be the circle centered at w and passing through these two intersection points; see Figure 5.4a. If c_2 is enclosed in c_1, let c_3 be the minimum circle centered at w and enclosing c_2; see Figure 5.4b. Let D_1, D_2 and D_3 be the disks corresponding to c_1, c_2 and c_3 respectively. Observe that $D_1 \cap D_2 \subset D_3$ and $\big((\mathbb{R}^2 \setminus D_1) \cap (\mathbb{R}^2 \setminus D_2)\big) \subset (\mathbb{R}^2 \setminus D_3)$. Thus all the endpoints of segments in S that are enclosed in c_1 and c_2 are also enclosed in c_3, and the ones that lie outside of c_1 and c_2 also lie outside of c_3. This implies that c_3 is a stabbing circle that is combinatorially the same as c_1 and c_2. By Lemma 5.1.1, $r_f(w) < r_h(w)$, and thus w is in the interior of FCVD*(S) and does not lie on an internal edge of it. This implies that the closed segment pq lies in one face of FCVD*(S). □

The second part of the above proof implies that, for any pair of points p, q within one face f of FCVD*(S), the segment pq lies in f. This proves the following property:

Corollary 5.2.2. *The faces of* FCVD*(S) *are convex.*

FCVD*(S) has the following useful visibility property:

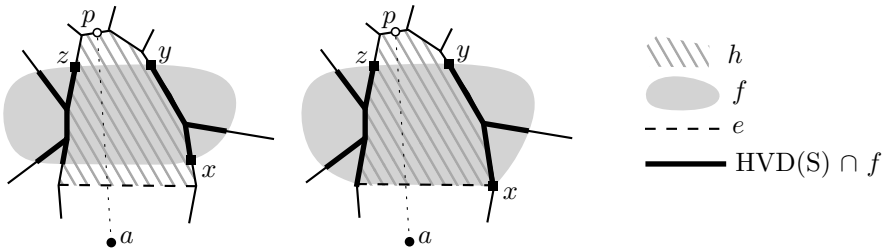


Figure 5.6. A face f of $\mathrm{FCVD}^*(S)$ in an impossible situation where $f \cap \mathrm{HVD}(S)$ is disconnected. Two variants with respect to the internal edge e: (a) e is outside $\overline{f}$; (b) e is on ∂f.

Lemma 5.2.7. *(a) Let p be a point outside $\mathrm{FCVD}^*(S)$, and let aa' be a segment in S such that $p \in \overline{\mathrm{hreg}}(a)$. Then the entire segment $(pa \cap \overline{\mathrm{hreg}}(a))$ is outside $\mathrm{FCVD}^*(S)$.*

(b) Let p' be a point in $\mathrm{FCVD}^*(S)$, and let aa' be a segment in S such that $p' \in \overline{\mathrm{fcreg}}(a)$. Then the entire segment $(p'a \cap \overline{\mathrm{fcreg}}(a))$ is in $\mathrm{FCVD}^*(S)$.

Proof. By Lemma 5.2.1, $(pa \cap \overline{\mathrm{hreg}}(a))$ and $(p'a \cap \overline{\mathrm{fcreg}}(a))$ are segments.

(a) Let q be a point on $pa \cap \overline{\mathrm{hreg}}(a)$; see Figure 5.5a. Clearly, $D_h(q) \subseteq D_h(p)$. Since $p \notin \mathrm{FCVD}^*(S)$, it follows that $r_f(p) > r_h(p)$ and, in consequence, there exists a segment $bb' \in S$ such that $b, b' \notin D_h(p)$. Since $D_h(q) \subseteq D_h(p)$, points b and b' are also outside $D_h(q)$. Thus, $q \notin \mathrm{FCVD}^*(S)$.

(b) Let q' be a point in $p'a \cap \overline{\mathrm{fcreg}}(a)$; see Figure 5.5b. Since $p' \in \mathrm{FCVD}^*(S)$, we have that $r_f(p') \leq r_h(p')$ and thus each segment from S has at least one endpoint outside $D_f(p')$ or on its boundary. Since $D_f(q') \subset D_f(p')$, the same holds for $D_f(q')$. Thus, $D_f(q') \subset D_h(q')$, that is, $q' \in \mathrm{FCVD}^*(S)$. □

The following two lemmas analyze the intersection of a face of $\mathrm{FCVD}^*(S)$ with $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$. In particular, we characterize connectedness and non-emptiness of such intersections, the properties that enable us to efficiently compute the faces of $\mathrm{FCVD}^*(S)$ in Section 5.3.

Lemma 5.2.8. *For a face f of $\mathrm{FCVD}^*(S)$, both $f \cap \mathrm{HVD}(S)$ and $f \cap \mathrm{FCVD}(S)$ are connected.*

Proof. Suppose for the sake of contradiction that $f \cap \mathrm{HVD}(S)$ is disconnected, see Figure 5.6. Then there is a face h of $\mathrm{hreg}(a)$ such that $f \cap \partial h$ has at least two connected components. By [66, Property 3], ∂h contains exactly one internal edge

e, $e \subseteq \mathrm{b_h}(a, a')$. By the definition of a face of FCVD*(S), e cannot intresect the interior of f; it can only border f or it must be outside $\overline{f}$, see Figure 5.6a and b.

Consider the first connected component of $f \cap \partial h$ that follows e in a counterclockwise traversal of ∂h. Let the endpoints of this component be called x and y (in the order of the traversal), see Figure 5.6. Let z be the first point of f encountered as we continue traversing ∂h counterclockwise beyond y. That is, z is a point in another connected component of $f \cap \partial h$. Consider the portions of ∂f and ∂h respectively from point y to point z. The portion of ∂f from y to z is inside h. The portion of ∂h from y to z consists solely of pure edges. Thus a point p in this portion of ∂h infinitesimally close to y is outside FCVD*(S). Then by Lemma 5.2.7a $pa \cap h$ is outside of FCVD*(S). We obtain a contradiction since $(pa \cap h) \cap f$ is not empty.

The proof for $f \cap \mathrm{FCVD}(S)$ is similar. In particular, if $f \cap \mathrm{FCVD}(S)$ is disconnected, then $f \cap \partial h$ is disconnected, for a face h of $\mathrm{fcreg}(a)$. Similarly to the above, by Lemma 5.2.3 and Lemma 5.2.1b, there is a point p on a pure edge on $\partial\mathrm{fcreg}(a)$, such that $p \notin \mathrm{FCVD}^*(S)$, and the supporting line of the segment pa intersects f inside $\mathrm{fcreg}(a)$. Let p' be a point in this intersection. Then point $p \in p'a$, $p \in \overline{\mathrm{fcreg}}(a)$ and $p \notin \mathrm{FCVD}^*(S)$, which contradicts Lemma 5.2.7b. □

(a) (b)

Figure 5.7. Illustration for the proof of Lemma 5.2.9. An impossible situation where a face f of FCVD*(S) (shaded) is such that (a) $\mathrm{HVD}(S) \cap f$ is empty; (b) $\mathrm{FCVD}(S) \cap f$ is empty.

Lemma 5.2.9. *For a face f of* FCVD*(S), *either $f \cap \mathrm{HVD}(S)$ or $f \cap \mathrm{FCVD}(S)$ is non-empty. If f is bounded and one of $f \cap \mathrm{HVD}(S)$, $f \cap \mathrm{FCVD}(S)$ is empty, then f belongs to a component of* FCVD*(S) *with multiple faces.*

Proof. To prove the first part of the statement, recall that any vertex on the boundary of a component of FCVD*(S) is incident to an edge of HVD(S) or of FCVD(S) within that component. Same holds for h- and fc-vertices on the boundary of a face of FCVD*(S) (but not for the mixed vertices). Thus we need to show that a face

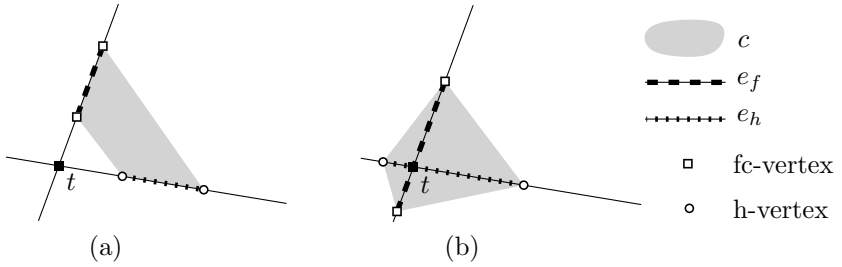


Figure 5.8. Illustration for the proof of Lemma 5.2.10. A bounded component c of FCVD*(S) that does not contain a vertex of HVD(S) or of FCVD(S): (a) an impossible situation; (b) the only possible situation.

f of FCVD*(S) has at least one h- or fc-vertex on its boundary. Suppose to the contrary, that ∂f contains only mixed vertices. These mixed vertices cannot all lie on the bisector of the same segment in S. But between any pair of consecutive mixed vertices incident to bisectors of different segments, ∂f must cross a pure edge separating the regions of these segments in one of the diagrams. Such crossing corresponds to an h- or an fc-vertex on ∂f. We obtain a contradiction.

Now we prove the second part of the statement. Suppose for the sake of contradiction that f is bounded, $f \cap \mathrm{HVD}(S)$ is empty, and $\overline{f}$ is a single-face component of FCVD*(S). Since $f \cap \mathrm{HVD}(S)$ is empty, f is entirely contained in $\mathrm{hreg}(a)$ for an endpoint a of some segment $aa' \in S$. Since segments in S do not share endpoints, no pure edge of HVD(S) can overlap with an edge of ∂FCVD*(S) (they can only intersect in one point). Then there is a point $p \in \mathrm{hreg}(a) \setminus f$ such that pa intersects f and $p \notin \mathrm{FCVD}^*(S)$ (see Figure 5.7a). But by Lemma 5.2.7a, since $p \notin \mathrm{FCVD}^*(S)$ the entire segment $(pa \cap \overline{\mathrm{hreg}}(a))$ must be outside FCVD*(S). We obtain a contradiction.

The symmetric statement about $f \cap \mathrm{FCVD}(S)$ can be shown as follows. Suppose that $f \subset \mathrm{fcreg}(a)$, see Figure 5.7b. Similarly to the above, edges of ∂f and $\partial\mathrm{fcreg}(a)$ do not overlap. Pick a point $p' \in f$. By Lemma 5.2.7b, the entire segment $p'a \cap \mathrm{fcreg}(a)$ lies in f. Recall that $a \notin \overline{\mathrm{fcreg}}(a)$ (since segments in S do not share endpoints), and thus f must intersect FCVD(S). We obtain a contradiction. □

Finally, in the next lemma, we explore a special type of single-face components of FCVD*(S): the ones that contain no vertices of HVD(S) or FCVD(S). The faces that correspond to bounded components of this type create a major difficulty in computing FCVD*(S).

Lemma 5.2.10. *Let c be a component of* FCVD*(S) *that contains no vertex of* HVD(S) *or* FCVD(S). *If c is bounded, then c contains exactly one intersection of*

two pure edges (one of $\mathrm{HVD}(S)$*, and one of* $\mathrm{FCVD}(S)$*), and* ∂c *is a quadrilateral. If* c *is unbounded, then* c *contains an unbounded portion of a pure edge of either* $\mathrm{HVD}(S)$ *or* $\mathrm{FCVD}(S)$.

Proof. Since c does not contain any mixed vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, by Lemma 5.2.5, c does not intersect any internal edge of the diagrams. Thus ∂c may only contain h- and fc-vertices (no mixed vertices). Further, the interior of c is a single face f of $\mathrm{FCVD}^*(S)$ ($\overline{f} = c$).

Suppose first that c is bounded. Lemmas 5.2.8 and 5.2.9 imply that $f \cap \mathrm{HVD}(S)$ is, respectively, connected and non-empty. Together with the fact that f contains no vertices of $\mathrm{HVD}(S)$, this implies that $f \cap \mathrm{HVD}(S)$ is an (open) line segment, whose endpoints are exactly the two h-vertices on ∂c. By the analogous argument for $f \cap \mathrm{FCVD}(S)$, ∂c has exactly two fc-vertices. Therefore $\partial c = \partial f$ is a quadrilateral. See Figure 5.8b.

Let e_h and e_f denote respectively the open line segments $f \cap \mathrm{HVD}(S)$ and $f \cap \mathrm{FCVD}(S)$; see Figure 5.8. Let t be the point of intersection between the supporting lines of e_h and e_f. We will show that $t \in f$. Note that it is impossible that an h-vertex is followed by the other h-vertex on ∂c (see Figure 5.8a), as it would imply that e_h and e_f lie entirely on ∂c, thus $f \cap \mathrm{HVD}(S)$ would be empty (as well as $f \cap \mathrm{FCVD}(S)$), what contradicts Lemma 5.2.9. Thus the different type of vertices interleave on ∂f (see Figure 5.8b), which implies that $t \in f$. This completes the proof of the first statement.

Now suppose that c is unbounded. Note that ∂c consists of at least two edges, and c is convex (see Corollary 5.2.2). Thus there are two unbounded edges on ∂c that have different supporting lines. Denote these edges as e and g. If c would contain no unbounded portions of pure edges of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, then both e and g would be contained in $\mathrm{hreg}(a)$ and in $\mathrm{fcreg}(b)$, for some endpoints a and b of segments in S. But then both e and g must be portions of $\mathrm{b_h}(a,b)$. We obtain a contradiction. □

We have explored different types of components of $\mathrm{FCVD}^*(S)$, and we saw that a component may or may not be comprised of multiple faces, may or may not contain a vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, and it may or may not be bounded. The cumulative complexity of all components that either contain multiple faces, or contain a vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, or are unbounded, is $O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$ (see the proof of Theorem 5.2.1 in Section 5.2.3). However, the bounded single-face components that do not contain a vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$ (i.e., the quadrilateral components of Lemma 5.2.10) do not fall under this bound. We bound their number in the next section. Identifying these components poses the main challenge to our algorithm in Section 5.3.

5.2.3 Complexity of $\mathrm{FCVD}^*(S)$

With some abuse of notation, we denote by $|\mathrm{HVD}(S)|$, $|\mathrm{FCVD}(S)|$ and $|\mathrm{FCVD}^*(S)|$ respectively the number of edges of $\mathrm{HVD}(S)$, $\mathrm{FCVD}(S)$, and $\partial\mathrm{FCVD}^*(S)$. We aim to connect $|\mathrm{FCVD}^*(S)|$ with $|\mathrm{HVD}(S)|$ and $|\mathrm{FCVD}(S)|$.

We classify the segments in S with respect to a portion of a fixed pure edge of $\mathrm{HVD}(S)$. Let e be a connected portion of pure edge of $\mathrm{HVD}(S)$, that separates $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$, for two segments $aa', bb' \in S$. In particular, $e \subseteq \mathrm{b_h}(a,b)$. For the rest of this section, it is convenient to perform a rotation of the coordinate system so that e is horizontal. Let u and respectively v be the left and right endpoints of e.

If u is a mixed vertex of $\mathrm{HVD}(S)$, we redefine u as a point on e infinitesimally to the right, so that u is in the boundary of only $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$. We proceed analogously with v.

The Hausdorff disks $D_h(u)$ and $D_h(v)$ have a, b on the boundary, they contain aa', bb', and they do not contain any other segment of S. The same holds for the Hausdorff disk of any point of e. Hence, every segment $cc' \in S \setminus \{aa', bb'\}$ can be classified as follows (see Figure 5.9, left):

- cc' is of type *out* if both c and c' are outside $D_h(u) \cup D_h(v)$;
- cc' is of type *in* if either c or c' is contained in $D_h(u) \cap D_h(v)$ and the other endpoint is outside $D_h(u) \cup D_h(v)$;
- cc' is of type *left* if either c or c' is contained in $D_h(u) \setminus D_h(v)$ and the other endpoint is outside $D_h(u) \cup D_h(v)$;
- cc' is of type *right* if either c or c' is contained in $D_h(v) \setminus D_h(u)$ and the other endpoint is outside $D_h(u) \cup D_h(v)$;
- cc' is of type *middle* if either c or c' is contained in $D_h(u) \setminus D_h(v)$ and the other endpoint is contained in $D_h(v) \setminus D_h(u)$.

For any point $p \in e$, we use p_ℓ and p_r to denote two points in e infinitesimally close to p and lying to the left and right of p, respectively. Additionally, let M^e_{bis} denote the set of segments $cc' \in S$ of type *middle* for edge e such that e intersects an internal edge of $\mathrm{FCVD}(S)$ separating $\mathrm{fcreg}(c)$ from $\mathrm{fcreg}(c')$. Notice that this internal edge is a portion of $\mathrm{b_h}(c,c')$. We also set $m^e_{bis} = |M^e_{bis}|$.

Let m_{bis} denote the total number of pairs formed by a pure edge e of $\mathrm{HVD}(S)$ and a segment $cc' \in S$ such that $cc' \in M^e_{bis}$.

Lemma 5.2.11. *Let e be a portion of a pure edge of $\mathrm{HVD}(S)$. The number of faces of $\mathrm{FCVD}^*(S)$ intersected by e is at most $1 + m^e_{bis}$.*

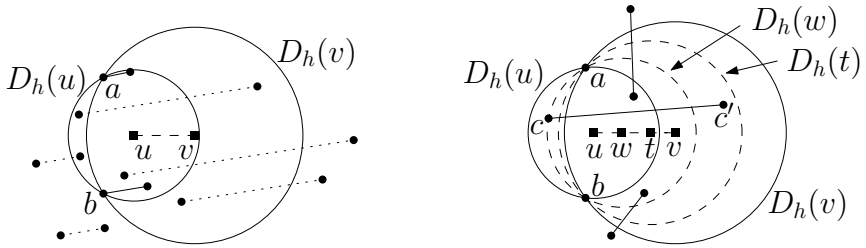


Figure 5.9. Left: From top to bottom, the types of the dotted segments are middle, left, in, right, and out. Right: Illustration for the proof of Lemma 5.2.11.

Proof. Let e' be the left-most portion (if any) of the interior of e contained in $\mathrm{FCVD}^*(S)$, and let w be its right endpoint. Since $w \in \mathrm{FCVD}^*(S)$, each segment from S has at least one endpoint inside $D_h(w)$. Let cc' be a segment in S such that $w_r \in \overline{\mathrm{fcreg}}(c)$. Since $w_r \notin \mathrm{FCVD}^*(S)$, both c and c' are outside $D_h(w_r)$. This implies that one of c or c' (say, c) lies in the portion of the boundary of $D_h(w)$ contained in $D_h(u)$, and the other endpoint lies outside $D_h(w)$ (see Figure 5.9, right). Consequently, for any point t in $w_r v$, $D_h(t)$ does not contain c. Suppose that there are more portions of e contained in $\mathrm{FCVD}^*(S)$, and let e'' be the left-most one. Then, for any point t in e'', $D_h(t)$ contains c' but not c. In particular, c' lies in $D_h(v) \setminus D_h(u)$ and cc' is of type *middle* for e. Hence, $\mathrm{b_h}(c,c')$ intersects e at a point q leaving e' to its left and e'' to its right. If q lies on an internal edge of $\mathrm{FCVD}(S)$ separating $\mathrm{fcreg}(c)$ from $\mathrm{fcreg}(c')$, then $cc' \in M^e_{bis}$ and we assign e'' to cc'.

Otherwise, q lies in $\overline{\mathrm{fcreg}}(d)$, for some $dd' \in S \setminus \{aa', bb', cc'\}$. Since $w_r \in \overline{\mathrm{fcreg}}(c)$, the segment $w_r q$ crosses at least one edge of $\mathrm{FCVD}(S)$. We start traversing the segment $w_r v$ starting from w_r. Suppose that we leave $\overline{\mathrm{fcreg}}(c)$ and we enter $\overline{\mathrm{fcreg}}(f)$, for some $ff' \in S$. Notice that this happens before we reach e''. Since $w \in \mathrm{FCVD}^*(S)$, $w_r \notin \mathrm{FCVD}^*(S)$ and $f \neq c'$, the point f does not lie in $D_h(u) \cap D_h(v)$, $D_h(v) \setminus D_h(u)$ or outside $D_h(u) \cup D_h(v)$. Hence, it lies in $D_h(u) \setminus D_h(v)$. Furthermore, since $e'' \subseteq \mathrm{FCVD}^*(S)$, ff' is of type *middle* and, for any point t in e'', $D_h(t)$ contains f'. Hence, $\mathrm{b_h}(f,f')$ intersects e at a point q' leaving e' to its left and e'' to its right. If q' lies on an internal edge of $\mathrm{FCVD}(S)$ separating $\mathrm{fcreg}(f)$ from $\mathrm{fcreg}(f')$, then $ff' \in M^e_{bis}$ and we assign e'' to ff'. Otherwise, we continue traversing $w_r v$. The left endpoint of e'' is in the farthest-color region of a point in $D_h(v) \setminus D_h(u)$. Thus, as we traverse $w_r v$ and simultaneously $\mathrm{FCVD}(S)$, at some point we cross an edge of $\mathrm{FCVD}(S)$ separating the farthest-color region of a point in $D_h(u) \setminus D_h(v)$ from the farthest-color region of a point in $D_h(v) \setminus D_h(u)$. This is only possible when this edge is an internal edge of $\mathrm{FCVD}(S)$

separating the regions of two endpoints of a segment of type *middle*. We assign ee' to this segment in M^e_{bis}.

Next, we select the right endpoint of e'' and perform the same analysis. By repeating the same argument until we reach v, we obtain an assignment of the portions of the interior of e contained in $\mathrm{FCVD}^*(S)$ (except for e') to segments in M^e_{bis}. Furthermore, in this assignment no segment is charged more than one portion. This completes the proof of the lemma. □

Theorem 5.2.1. *Let S be a set of n segments in the plane in general position. Then* $|\mathrm{FCVD}^*(S)| = O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + m_{bis})$.

Proof. Consider a face f of $\mathrm{FCVD}^*(S)$. By Lemma 5.2.8, $f \cap \mathrm{HVD}(S)$ is a connected graph. Each vertex of this graph (if any) has degree three, since it is a vertex of $\mathrm{HVD}(S)$. By Euler's formula the number of h-vertices on ∂f, is $O(H+1)$, where H is the number of vertices of $\mathrm{HVD}(S)$ inside f. Analogously, the number of fc-vertices on ∂f is $O(F+1)$, where F is the number of vertices of $\mathrm{FCVD}(S)$ inside f.

Any vertex on ∂f that is neither an h-vertex nor an fc-vertex, is a mixed vertex of either $\mathrm{HVD}(S)$ or of $\mathrm{FCVD}(S)$. Clearly, each pure vertex of $\mathrm{HVD}(S)$ and of $\mathrm{FCVD}(S)$ lies in at most one face of $\mathrm{FCVD}^*(S)$, and each mixed vertex of one of these diagrams lies on the boundary of exactly two faces of $\mathrm{FCVD}^*(S)$. Thus the total complexity of the boundary of all components of $\mathrm{FCVD}^*(S)$ that contain at least one vertex of either $\mathrm{HVD}(S)$ or of $\mathrm{FCVD}(S)$ is $O(\mathrm{HVD}(S) + \mathrm{FCVD}(S))$.

What remains is to bound the number of components of $\mathrm{FCVD}^*(S)$ that do not contain a vertex of $\mathrm{HVD}(S)$ or of $\mathrm{FCVD}(S)$, and whose complexity is thus $O(1)$, see the above argument of Lemma 5.2.10. We consider separately the unbounded and the bounded components of this type. The total number of unbounded ones is $O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$: by Lemma 5.2.10, each such unbounded component contains an unbounded portion of an edge of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, and clearly one edge can correspond to at most one component. By Lemma 5.2.10, each bounded component of $\mathrm{FCVD}^*(S)$ with no vertex of $\mathrm{HVD}(S), \mathrm{FCVD}(S)$ intersects exactly one pure edge of $\mathrm{HVD}(S)$, and this intersection is connected. Thus the number of such components of $\mathrm{FCVD}^*(S)$ can be upper-bounded by the total number of intersections of all pure edges of $\mathrm{HVD}(S)$ with $\mathrm{FCVD}^*(S)$. By Lemma 5.2.11, this is $O(|\mathrm{FCVD}(S)| + |\mathrm{HVD}(S)| + m_{bis})$.

Summing up the total complexity of the boundary for all types of components of $\mathrm{FCVD}^*(S)$ implies the claim. □

5.3 Computing FCVD*(S)

Recall from Section 5.2.2 that any unbounded component of $\mathrm{FCVD}^*(S)$ corresponds to a stabbing line for S. All the stabbing lines and the corresponding components of $\mathrm{FCVD}^*(S)$ can be found in $O(n \log n)$ time [36]. From now on we assume that S has no stabbing line, and therefore all components and faces of $\mathrm{FCVD}^*(S)$ are bounded.

5.3.1 General algorithm

By Lemmas 5.2.5, 5.2.9 and 5.2.10, any bounded face f of $\mathrm{FCVD}^*(S)$ has at least one of the following properties: (1) f is incident to a mixed vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, (2) f contains a pure vertex of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$, or (3) f contains exactly one segment of a pure edge of both $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$. Our algorithm, described in Figure 5.10, has three parts. All faces of type (1) are computed in the first part (Steps 1–7). Among the faces that have not been computed yet, the ones that are of type (2) are computed in the second part (Steps 8–12). After that the faces that satisfy only property (3) are left, and they are computed in Steps 13–14.

Algorithm *Computing FCVD*$^*(S)$

1. compute $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$;
2. **for** each mixed vertex v of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$ **do**
3. **if** $v \in \partial\mathrm{FCVD}^*(S)$ **then**
4. **for** each face f of $\mathrm{FCVD}^*(S)$ incident to v;
5. **if** f has not been visited yet **then**
6. trace ∂f starting from v;
7. mark all vertices of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ in $\overline{f}$;
8. **for** each non-marked pure vertex u of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$ **do**
9. **if** $u \in \mathrm{FCVD}^*(S)$ **then**
10. let f be the face of $\mathrm{FCVD}^*(S)$ that contains u;
11. find a point on ∂f and trace ∂f;
12. mark all vertices of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ in $\overline{f}$;
13. **for** each segment e of a pure edge of $\mathrm{HVD}(S)$ that is outside the computed faces of $\mathrm{FCVD}^*(S)$ **do**
14. compute all faces of $\mathrm{FCVD}^*(S)$ that intersect e;

Figure 5.10. Algorithm to compute $\mathrm{FCVD}^*(S)$.

We next make some remarks about the first two parts of the algorithm in Fig-

ure 5.10.

Steps 3 and 9 are simple, after $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ are computed and preprocessed for point location queries. For example, if v is a vertex of say $\mathrm{HVD}(S)$, we first locate it in $\mathrm{FCVD}(S)$. Once we obtain a segment $aa' \in S$ such that $v \in \overline{\mathrm{fcreg}}(aa')$, we can compare the Hausdorff and farthest-color radii of v in constant time.

The loop in Step 4 performs exactly two iterations, since each mixed vertex on $\partial\mathrm{FCVD}^*(S)$ has exactly two incident faces of $\mathrm{FCVD}^*(S)$ (see Section 5.2.2).

In Step 11, given a point u inside face f, we proceed as follows to find a point on ∂f. We first locate u in both $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$. Then we trace in both diagrams the vertical ray originating at u, until we reach the boundary of f. To perform this efficiently, we preprocess $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ for ray-shooting queries.

The third part of the algorithm (Steps 13–14) is discussed in Section 5.3.2, its correctness in Section 5.3.3, and its time complexity in Section 5.3.4.

5.3.2 Searching in a pure edge of $\mathrm{HVD}(S)$

In this section, we present an algorithm to compute all faces of $\mathrm{FCVD}^*(S)$ intersected by a given portion of a pure edge of $\mathrm{HVD}(S)$.

Due to the assumption that S does not have any stabbing line, all faces of $\mathrm{FCVD}^*(S)$ we are searching for are bounded. Thus, the input portion of a pure edge of $\mathrm{HVD}(S)$ is assumed to be a line segment. The algorithm is given as a pseudocode in Figure 5.12.

Without loss of generality, we assume that the input segment $e = uv$ is horizontal, and that u is to the left of v. The algorithm assumes that, if e is shrunk infinitesimally from both sides (see Step 1), the resulting segment $u_r v_\ell$ has both endpoints outside $\mathrm{FCVD}^*(S)$. To proceed with the description of the algorithm, we need to introduce some notation. For convenience, we assume that the segment uv is already shrunk.

Let $e = uv$ be a line segment on a pure edge of $\mathrm{HVD}(S)$ separating $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$, such that u and v are outside $\mathrm{FCVD}^*(S)$. Additionally, if the segment ab intersects the interior of e, this intersection divides e into two portions, which we process separately. Note that neither u nor v are mixed vertices of $\mathrm{HVD}(S)$ (since uv is the result of shrinking a portion of a pure edge of $\mathrm{HVD}(S)$ from both sides).

We classify the segments in $S \setminus \{aa', bb'\}$ with respect to uv as segments of types *left*, *right*, *middle*, *in*, *out*, as in Section 5.2.3 (see Figure 5.9). Using this classification, for any point w in e, we define $type(w)$ as a set containing one element per each $cc' \in S$ such that $w \in \overline{\mathrm{fcreg}}(cc')$. The elements of $type(w)$ are

defined as follows: Let cc' be a segment in S such that $w \in \overline{\text{fcreg}}(cc')$. If cc' is not of type *middle*, then we add the type of cc' to $type(w)$. If cc' is of type *middle*, then either c or c' (say, c) is contained in $D_h(u) \setminus D_h(v)$, and the other endpoint (c') is contained in $D_h(v) \setminus D_h(u)$. We further differentiate the classification *middle* as follows: If w lies on $\text{b}_\text{h}(c,c')$, then $mm \in type(w)$. Otherwise, if $w \in \overline{\text{fcreg}}(c)$, then $ml \in type(w)$; if $w \in \overline{\text{fcreg}}(c')$, then $mr \in type(w)$. When we need to specify cc', we do as follows: Imagine that $w \in \overline{\text{fcreg}}(cc')$ and cc' is of type *in*. Then we say $in \in type(w)$ *caused by* cc'.

Further, we use $\tilde{l}$ to denote types *left* and *ml*, and we use $\tilde{r}$ to denote *right* and *mr*.

Definition 24. A point w in e is a *changing point* if $\{\tilde{r},\tilde{l}\} \subseteq type(w)$ (see Figure 5.11).

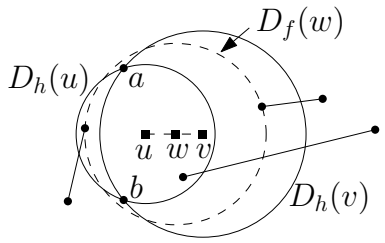


Figure 5.11. w is a changing point (not in FCVD*(S)).

A changing point might or might not be in $\text{FCVD}^*(S)$. Note that a changing point w is an intersection point between a pure edge of $\text{HVD}(S)$ and a pure edge of $\text{FCVD}(S)$. Intuitively, at w the point giving the farthest-color radius changes from being in $D_h(v) \setminus D_h(u)$ to being in $D_h(u) \setminus D_h(v)$, i.e., $\tilde{r} \in type(w_\ell)$ and $\tilde{l} \in type(w_r)$ (see Figure 5.11). It is easy to see that a point w' where a change in the other direction happens is of type *mm*, i.e., w' is in the intersection between a pure edge of $\text{HVD}(S)$ and an internal edge of $\text{FCVD}(S)$. Then we have the following.[1]

Observation 5.3.1. *(a) If e intersects a face of* $\text{FCVD}^*(S)$ *in a segment e', then there exists a point w in e' such that $in \in type(w)$ or w is a changing point. (b) If there is a point $w \in e$ such that $out \in type(w)$, then e does not intersect* $\text{FCVD}^*(S)$. *(c) If there is a point $w \in e$ such that $in \in type(w)$, then $w \in \text{FCVD}^*(S)$.*

[1] Even though Observation 5.3.1 is the base of the algorithm *Search-In e*, it is actually not needed to prove its correctness, given in Lemma 5.3.3, because Lemma 5.3.3 in a way reproves Observation 5.3.1. Therefore, we omit the proof of Observation 5.3.1.

Thus, it is enough to examine the changing points of e, and the points w such that $in \in type(w)$. To find such points, we use the *find-change query* subroutine, defined next. When dealing with a subsegment ts of e, or a pair (t,s) of points in e, we write the left-most point first.

Definition 25 (Find-change query)**.** The input of the query is a pair (t,s) of points in e, such that $type(t)$ contains $\tilde{r}$ but not $\tilde{l}$, and $type(s)$ contains $\tilde{l}$ but not $\tilde{r}$. The query returns a point w in the segment ts such that one of the following holds: (i) w is a changing point; (ii) $in \in type(w)$; (iii) $out \in type(w)$.

For the sake of clarity, we defer the proof that the find-change query is well-defined, as well as the proof of correctness of the remaining pieces of the algorithm, to the next subsection.

Algorithm *Search-In* $e = uv$
1. $uv \leftarrow u_r v_\ell$; ($*$ shrink the segment uv infinitesimally $*$)
2. **if** $\tilde{r}$ is the only element in $type(u)$ **and** $\tilde{l}$ is the only element in $type(v)$ **then**
3. perform a find-change query on (u,v);
4. let w be the point returned by the query;
5. **if** $out \in type(w)$ **then**
6. **return**;
7. **if** $w \in \mathrm{FCVD}^*(S)$ **then**
8. trace ∂f, where f is the face of $\mathrm{FCVD}^*(S)$ that contains w;
9. let q, q' be the points of $\partial f \cap uv$, where q is to the left of q';
10. **else** ($*$ w is a changing point in uv **and** $w \notin \mathrm{FCVD}^*(S)$ $*$)
11. let q, q' both be w;
12. *Search-In* uq;
13. *Search-In* $q'v$;
14. **return**;
15. **else return**;

Figure 5.12. Algorithm to compute all faces of $\mathrm{FCVD}^*(S)$ intersected by $e = uv$.

We are now ready to describe our algorithm *Search-In* e, which computes all the faces of $\mathrm{FCVD}^*(S)$ intersected by e. The algorithm is illustrated in Figure 5.12; it uses the characterization from Observation 5.3.1. At any time, the algorithm processes a subsegment of e. It first shrinks e infinitesimally to ensure that its endpoints do not belong to $\mathrm{FCVD}^*(S)$ (Step 1). Then it performs a check that

allows to eliminate some execution paths, when it is guaranteed that e does not intersect $\mathrm{FCVD}^*(S)$. If the check is passed, the search continues as follows. It performs a find-change query on uv that returns a point w (Steps 3–4). If $out \in type(w)$, the algorithm stops (Step 6), since it is guaranteed that $e \cap \mathrm{FCVD}^*(S) = \emptyset$. Otherwise, the algorithm proceeds. In case w is in $\mathrm{FCVD}^*(S)$, the face containing w is traced (Step 8). In both cases (w is in $\mathrm{FCVD}^*(S)$ or not), the algorithm calls itself recursively for two disjoint subsegments uq and $q'v$ of e (Steps 12–13). Initially, the algorithm is called for $e = uv$.

We remark that the faces of $\mathrm{FCVD}^*(S)$ intersecting e are not found in a left-to-right or any other "natural" order, as the find-change query finds *some* point of the desired property. This is the reason why, after finding a changing point $w \in e$, the algorithm continues searching on both sides of w. We remark as well that, at every recursive call of the algorithm, the function $type(\cdot)$ is re-defined according to the extremes of the segment on which the procedure is called. Therefore, for a point $x \in e$, the value of $type(x)$ may change as the algorithm proceeds.

5.3.3 Correctness

We now prove that the algorithm presented in Sections 5.3.1 and 5.3.2 is correct. First we show that Find-change query is well-defined:

Lemma 5.3.1. *If the pair (t,s) satisfies the conditions of the input of Find-change query, then there exists a point w in the segment ts such that w is a changing point, $in \in type(w)$, or $out \in type(w)$.* (See the proof in Section A.2 of the Appendix.)

The following lemma refers to Step 2 of the algorithm *Search-In e*:

Lemma 5.3.2. *Let $e = uv$, where $u, v \notin \mathrm{FCVD}^*(S)$. If there is an element x in $type(u)$ such that $x \neq \tilde{r}$ or there is an element y in $type(v)$ such that $y \neq \tilde{l}$, then $uv \cap \mathrm{FCVD}^*(S) = \emptyset$.*[2]

Proof. Recall that, if a point w belongs to $\mathrm{FCVD}^*(S)$, then $D_h(w)$ contains at least one endpoint of every segment in S. Since $u \notin \mathrm{FCVD}^*(S)$, each segment $cc' \in S$ such that $u \in \overline{\mathrm{fcreg}}(cc')$ has both endpoints outside $D_h(u)$. Therefore $in \notin type(u)$, $\tilde{l} \notin type(u)$, and $mm \notin type(u)$. Analogously, since $v \notin \mathrm{FCVD}^*(S)$, we have that $in \notin type(v)$, $\tilde{r} \notin type(v)$, and $mm \notin type(v)$. On the other hand, observe that the Hausdorff disk of any point in e is contained in $D_h(u) \cup D_h(v)$. This implies that, if out belongs to $type(u)$ or $type(v)$, then $uv \cap \mathrm{FCVD}^*(S) = \emptyset$. □

[2]Note that it is not always possible to shrink uv infinitesimally so that $type(u)$ and $type(v)$ consist of one element each. In particular, this is not possible when uv lies on an edge of $\mathrm{FCVD}(S)$, and this situation does not contradict our general position assumption.

The following lemma proves that the algorithm *Search-In e* is correct:

Lemma 5.3.3. *The algorithm* Search-In *e computes all faces of* $\mathrm{FCVD}^*(S)$ *intersected by e.* (See the proof in Section A.2 of the Appendix.)

5.3.4 Running time

Below we analyze the time complexity of the algorithm to compute $\mathrm{FCVD}^*(S)$, and thus the time complexity to solve the stabbing circle problem.

We start with the following result concerning the find-change query.

Lemma 5.3.4. *A find-change query can be performed in* $O(\log^2 n)$ *time.*

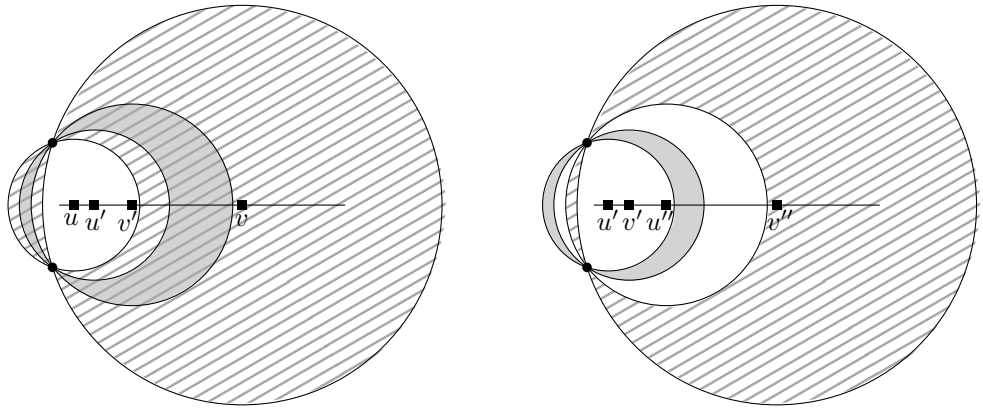
Proof. For a pair (t,s) of points, we perform the find-change query as follows. We use a point-location data structure for $\mathrm{FCVD}(S)$, such that the point location for a query point q is performed by a sequence of $O(\log n)$ atomic questions of the form "is q above or below (respectively, to the left or right of) a line ℓ?" (e.g., the data structure by Edelsbrunner et al. [39], or the one by Kirkpatrick [49]). Notice that, in our case, instead of a fixed point q, we only have a pair (t,s) such that the segment ts contains a desired point (a changing point, a point of type *in* or a point of type *out*). An atomic question is processed as follows. If $ts \cap \ell = \emptyset$, the answer is the same for any point in ts, and we continue with the pair (t,s). Otherwise, let point p be $ts \cap \ell$. First, the standard point location for p gives us $type(p)$. If $\tilde{r} \in type(p)$ and $\tilde{l} \notin type(p)$, we continue with the pair (p,s). Symmetrically, if $\tilde{l} \in type(p)$ and $\tilde{r} \notin type(p)$, we continue with the pair (t,p). If $type(p) = \{mm\}$, then we continue with either (p,s) or (t,p). Otherwise, we stop the procedure, and return p. Clearly, this happens in one of the following cases: (i) $\{\tilde{l},\tilde{r}\} \subseteq type(p)$; (ii) $in \in type(p)$; or (iii) $out \in type(p)$.

Answering one atomic question within the procedure takes $O(\log n)$ time, and the whole find-change query takes $O(\log^2 n)$ time.
A similar idea of simulating a point location for an unknown point is used in Cheong et al. [23], and in Cheilaris et al. [22, Section 7]. □

By an argument similar to the proof of Lemma 5.2.11, we can show that the running time of *Search-In e* is related to m_e:

Lemma 5.3.5. *Let e be a segment on a pure edge of* $\mathrm{HVD}(S)$. *The number of changing points in e is* $O(1+m_e)$.

Lemma 5.3.6. *Let e be a segment on a pure edge of* $\mathrm{HVD}(S)$*, and* cc' *be a segment in S.*



(a) (b)

Figure 5.13. Illustration for the proof of Lemma 5.3.6.

(a) If cc' is of type middle for a segment $e' \subset e$, then cc' is of type middle for e.

(b) Let e', e'' be two disjoint subsegments of e. Then cc' is of type middle for at most one of e', e''.

Proof. Let $e = uv$, $e' = u'v'$, and $e'' = u''v''$.

(a) If cc' is of type middle for e', then one of its endpoint is in $D_h(u') \setminus D_h(v')$ and the other one is in $D_h(v') \setminus D_h(u')$ (see the two gray areas in Figure 5.13a). These areas are contained in $D_h(u) \setminus D_h(v)$ and $D_h(v) \setminus D_h(u)$, respectively (shown with tiling pattern in Figure 5.13a).

(b) If cc' was of type middle for both e' and e'', then one of its endpoints would lie in both $D_h(u') \setminus D_h(v')$ and $D_h(u'') \setminus D_h(v'')$. But these two areas do not intersect (see the shaded and the tiled area in Figure 5.13b), which yields a contradiction.

□

The next lemma allows to bound the total time of all executions of Step 11 of *Computing* $\mathrm{FCVD}^*(S)$:

Lemma 5.3.7. *Let $f_1, f_2, \ldots, f_k$ be the faces of $\mathrm{FCVD}^*(S)$. The total number of edges of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ intersected by these faces is $O(k + |\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$.* See the proof in Section A.2 of the Appendix.

We are finally ready to prove the main theorem of this section.

Let m denote the number of pairs formed by a segment $aa' \in S$ and a pure edge e of $\mathrm{HVD}(S)$ such that aa' is of type *middle* for e. Let $\mathscr{T}_{\mathrm{HVD}(S)}$ and $\mathscr{T}_{\mathrm{FCVD}(S)}$ denote the time to compute $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$, respectively.

Theorem 5.3.1. *Let S be a set of n segments in the plane in general position. Then,* $\mathrm{FCVD}^*(S)$ *can be computed in time* $O(\mathcal{T}_{\mathrm{HVD}(S)} + \mathcal{T}_{\mathrm{FCVD}(S)} + (|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + m)\log^2 n)$.

Proof. Computing $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ (Step 1 of *Computing* $\mathrm{FCVD}^*(S)$) requires time $\mathcal{T}_{\mathrm{HVD}(S)} + \mathcal{T}_{\mathrm{FCVD}(S)}$.

After computing the diagrams, in time $O(|\mathrm{HVD}(S)|\log n)$ and $O(|\mathrm{FCVD}(S)|\log n)$ we can preprocess them to answer point-location queries in $O(\log n)$ time. Then Steps 3 and 9 of the algorithm require logarithmic time.

We also preprocess $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ to answer ray-shooting queries in $O(\log n)$ time. The preprocessing can be done in time $O(|\mathrm{HVD}(S)|\log n)$ and $O(|\mathrm{FCVD}(S)|\log n)$, respectively [20]. After such preprocessing, Step 11 requires $O((1+I(f))\log n)$ time, where $I(f)$ is the number of edges of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ intersected by f (see Section 5.3.1 for more details on Step 11). By Lemma 5.3.7, the total time spent for Step 11 is $O((|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + |\mathrm{FCVD}^*(S)|)\log n)$.

After a point on ∂f is known, ∂f can be traced in $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ in a standard way. Thus the total time required for tracing the boundary of all the faces of $\mathrm{FCVD}^*(S)$ (in Steps 6 and 11 of *Computing* $\mathrm{FCVD}^*(S)$, and Step 8 of *Search-In e*) is $O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + |\mathrm{FCVD}^*(S)|)$.

Therefore, the total time complexity of Steps 2–12 of *Computing* $\mathrm{FCVD}^*(S)$ is $O((|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + |\mathrm{FCVD}^*(S)|)\log n)$.

The third loop (Steps 13–14) of *Computing* $\mathrm{FCVD}^*(S)$ performs $O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$ iterations. Indeed, the number of faces computed in Steps 2–12 is $O(|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$, and each face has connected intersection with $\mathrm{HVD}(S)$ by Lemma 5.2.8.

In Step 14, *Search-In* $e = uv$ is called. *Search-In e* calls itself recursively for two subsegments of e if and only if the find-change query for e returned a point $w \in e$ ($w \neq u, v$) such that at least one of two conditions hold: (1) w belongs to a face f of $\mathrm{FCVD}^*(S)$, or (2) w is a changing point. If (1) holds, the two segments uq and $q'v$ for which the algorithm is called recursively do not intersect f. We charge the two recursive calls to f. Otherwise, w is a changing point for uv, and we charge these two calls to w (note that uq and $q'v$ do not contain w in their interior, so w will not be charged any other call in future iterations of the algorithm). Therefore, the total number of recursive calls caused by *Search-In* $e = uv$ is proportional to the number of changing points in uv plus the number of faces of $\mathrm{FCVD}^*(S)$ intersected by uv. By Lemma 5.2.11, the latter number is proportional to the former one. Thus the number of (recursive) calls of *Search-In* performed on portions of e is $O(1+m_e)$. Summing up m_e for all pieces of edges of $\mathrm{HVD}(S)$ on which the procedure is called

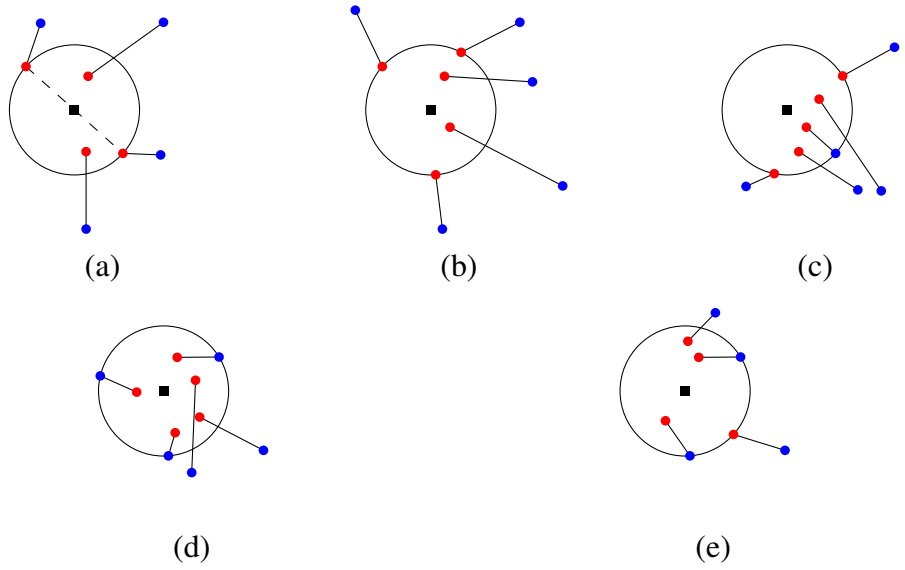


Figure 5.14. All possibilities for the stabbing circles of minimum and maximum radius.

results in $O(m)$ due to Lemma 5.3.6. Since any face of $\mathrm{FCVD}^*(S)$ computed by *Search-In e* intersects only one edge of $\mathrm{HVD}(S)$ and one edge of $\mathrm{FCVD}(S)$ and its boundary has constant complexity (see Lemma 5.2.10), Step 8 requires constant time. Therefore, one execution of *Search-In e*, except for the recursive calls, is dominated by the find-change query, and thus requires $O(\log^2 n)$ time (see Lemma 5.3.4). This globally amounts to $O((|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + m)\log^2 n)$ time for Steps 13–14 of *Computing* $\mathrm{FCVD}^*(S)$. □

The next lemma shows how to find the largest and smallest stabbing circles, once $\mathrm{FCVD}^*(S)$ is known. We observe that, in some cases, the stabbing circle of minimum radius does not exist, because any stabbing circle can be shrunk by decreasing its radius or slightly moving its center. Moreover, the "limit" circle is not stabbing because the closed disk induced by the circle contains both endpoints of a segment in S (see Case 1c of the proof below). Similarly, it is easy to see that the stabbing circle of maximum radius never exists, since any stabbing circle can be slightly enlarged, but the "limit" circle is not stabbing. In these cases, even though these circles are not stabbing, to simplify the notation we call these "limit" circles the *stabbing circles with infimum* or *supremum radius*.

Lemma 5.3.8. *After computing* $\mathrm{FCVD}^*(S)$*, the stabbing circles of minimum (or infimum) and supremum radius can be determined in time* $O((|\mathrm{FCVD}^*(S)| + |\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)\log n)$.

Proof. We list all the possibilities for such circles below; see Figure 5.14.

Given a segment aa' and a circle c, we call "red" the endpoint of aa' which is closer to the center of c, and we call "blue" the other endpoint (ties are broken arbitrarily). Notice that, if c is stabbing, then the red endpoint of aa' lies inside the closed disk induced by c, and the blue endpoint lies outside. Standard geometric arguments show the following.

1. The stabbing circle of minimum or infimum radius is a circle passing through:
   (a) Two red points that are diametrically opposite. In this case the center of the circle is at the intersection point between an edge of $\mathrm{FCVD}(S)$ separating two regions $\mathrm{fcreg}(a)$ and $\mathrm{fcreg}(b)$ (for some $aa', bb' \in S$), and the segment connecting a to b. This is a stabbing circle of minimum radius.
   (b) Three red points. The center is at a vertex of $\mathrm{FCVD}(S)$. This is also a stabbing circle of minimum radius.
   (c) Two red points and one blue point. The center is on the boundary of $\mathrm{FCVD}^*(S)$. This is a stabbing circle of infimum radius.
2. The stabbing circle of supremum radius is a circle passing through:
   (a) Three blue points. In this case the center of the circle is at a vertex of $\mathrm{HVD}(S)$.
   (b) Two blue points and one red point. The center is on the boundary of $\mathrm{FCVD}^*(S)$.

The stabbing circles with minimum (or infimum) and supremum radius can thus be found by checking the vertices of $\mathrm{HVD}(S)$ or $\mathrm{FCVD}(S)$ lying in $\mathrm{FCVD}^*(S)$, the edges of $\mathrm{FCVD}(S)$ intersecting $\mathrm{FCVD}^*(S)$, and the boundary of $\mathrm{FCVD}^*(S)$. Since in all cases the radius of the corresponding circle can be computed in $O(\log n)$ time, the claim follows. □

As a conclusion, we put together the above results and connect them with the stabbing circle problem as stated in Section 1.4. By Lemma 5.2.6, distinct faces of $\mathrm{FCVD}^*(S)$ correspond to combinatorially different stabbing circles. Thus, the above results yield the following:

Corollary 5.3.1. *Let S be a set of n segments in the plane in general position. All the combinatorially different stabbing circles for S, and the ones with minimum and maximum radius, can be computed in $O(\mathscr{T}_{\mathrm{HVD}(S)} + \mathscr{T}_{\mathrm{FCVD}(S)} + (|\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)| + m)\log^2 n)$ time.*

5.4 Parallel Segments

Let S be a set of parallel segments. The goal of this section is to prove the following theorem.

Theorem 5.4.1. *The stabbing circle problem for a set S of n parallel segments in general position can be solved in $O(n\log^2 n)$ time and $O(n)$ space.*

We prove Theorem 5.4.1 using Theorem 5.3.1 and Corollary 5.3.1. Since the segments in S are parallel, they must by pairwise disjoint, and thus $\mathrm{HVD}(S)$ is an instance of *abstract Voronoi diagrams*; hence, $|\mathrm{HVD}(S)|$ is $O(n)$ and $\mathscr{T}_{\mathrm{HVD}(S)}$ is $O(n\log n)$ [50]. In Section 5.4.1 we show that $|\mathrm{FCVD}(S)|$ is also $O(n)$ and $\mathscr{T}_{\mathrm{FCVD}(S)}$ is $O(n\log n)$. In Section A.2.1 (in the Appendix) we show that $m = O(n)$. Thus, the algorithm of Section 5.3 for this particular case has time complexity $O(n\log^2 n)$, and we derive Theorem 5.4.1.

5.4.1 The farthest-color Voronoi diagram for a set of parallel segments

Lemma 5.4.1. *If all segments in S are parallel, then for each $aa' \in S$, $\mathrm{b_h}(a,a')$ contributes at most one internal edge to $\mathrm{FCVD}(S)$.*

Proof. Suppose for the sake of contradiction that there are two internal edges e and e' in $\mathrm{FCVD}(S)$ which are portions of $\mathrm{b_h}(a,a')$. Assume without loss of generality that the segments in S are vertical, thus, $\mathrm{b_h}(a,a')$ is horizontal, and e is to the left of e'. Let $v,u \in \mathrm{b_h}(a,a')$ be the mixed vertices that are respectively the right endpoint of e and the left endpoint of e'. The farthest-color disk $D_f(v)$ contains an endpoint of every segments in S, and $\partial D_f(v)$ passes through points a,a' and an endpoint b of some segment $bb' \in S$. Observe that bb' is to the left of aa', and $bb' \cap D_f(v) = \{b\}$ (see Figure 5.15 (left), where $\mathrm{fcreg}(aa')$ is shown shaded, and $\mathrm{b_h}(a,a')$ is shown dashed). Since u is to the right of v, the closed disk centered at u with radius $d(u,a)$ does not contain b nor b', thus, $u \notin \overline{\mathrm{fcreg}}(aa')$. We obtain a contradiction. □

Lemma 5.4.2. *If all segments in S are parallel, the structural complexity of $\mathrm{FCVD}(S)$ is $O(n)$.*

Proof. By Lemma 5.2.4, the number of unbounded faces of $\mathrm{FCVD}(S)$ is $O(n)$. Since all segments in S are parallel, by Lemma 5.4.1, the number of all internal edges in $\mathrm{FCVD}(S)$ is at most n. By Lemma 5.2.2, every bounded face of $\mathrm{FCVD}(S)$ has an internal edge on its boundary, thus there are at most $2n$ bounded faces in

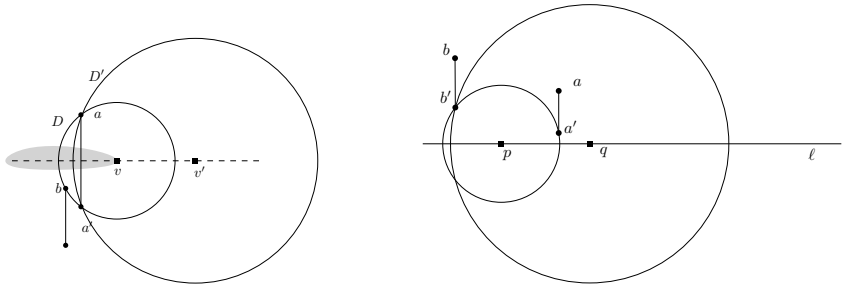


Figure 5.15. Left: Illustration for the proof of Lemma 5.4.1. Right: Illustration for the proof of Lemma 5.4.3.

$\mathrm{FCVD}(S)$. Thus, $\mathrm{FCVD}(S)$ is a planar graph with $O(n)$ faces whose vertices have degree three. By Euler's formula, $|\mathrm{FCVD}(S)| = O(n)$. □

Lemma 5.4.3. *If all segments in S are parallel, then* $\mathrm{FCVD}(S)$ *can be computed in* $O(n \log n)$ *time.*

Proof. Assume without loss of generality that the segments in S are vertical. We use the divide-and-conquer technique: We divide S by a vertical line into two halves, S_{left} and S_{right}, and recursively compute $\mathrm{FCVD}(S_{left})$ and $\mathrm{FCVD}(S_{right})$. Below we prove that the merge curve between $\mathrm{FCVD}(S_{left})$ and $\mathrm{FCVD}(S_{right})$ is y-monotone. Such a y-monotone merge curve can be constructed in $O(n)$ time by standard arguments on the divide-and-conquer construction of Voronoi diagrams, see e.g., [10]. The claim follows.

Suppose, for the sake of contradiction, that the merge curve is not y-monotone, that is, there are two points p, q on the merge curve with the same y-coordinate. Since p and q lie on the merge curve, the farthest-color radius of p with respect to S_{left} equals the farthest-color radius of p with respect to S_{right}, and the same holds for q. Let ℓ be the horizontal line through p and q. We redefine p and q as the left-most and second left-most points on ℓ that lie on the merge curve, respectively. In $\mathrm{FCVD}(S)$, the point at minus infinity on ℓ lies in the farthest-color region of a segment from S_{right}. Thus p lies on the boundary between $\mathrm{fcreg}(aa')$ (to the left of p) and $\mathrm{fcreg}(bb')$ (to the right of p) such that $aa' \in S_{right}$ and $bb' \in S_{left}$; see Figure 5.15 (right). Then the farthest-color disk $D_f(q)$ intersects or touches bb', and touches a segment $cc' \in S_{right}$. But in this case cc' is outside the farthest-color disk of p. We obtain a contradiction. □

5.5 Segments with the Delaunay property

We say that *S satisfies the Delaunay property* if its segments are edges of the Delaunay triangulation of some set of points. Let us assume that S satisfies this property.

Lemma 5.5.1. $\mathrm{FCVD}(S)$ *is a tree of* $O(n)$ *complexity.*

Proof. We show that $\mathrm{FCVD}(S)$ for such a segment set S is an instance of the *farthest abstract Voronoi diagram* (FAVD); the claim then follows automatically from [58]. To prove that $\mathrm{FCVD}(S)$ is FAVD, we consider the *nearest-color* Voronoi diagram of S, which reveals the nearest site (segment in S), where the distance from a point $p \in \mathbb{R}^2$ to some $aa' \in S$ is $\min\{d(p,a), d(p,a')\}$. We need to prove that the system of bisectors for farthest/nearest color Voronoi diagram satisfies the following axioms: (1) each bisector is an unbounded Jordan curve; (2) any two bisectors intersect finite number of times; (3) regions of the nearest-color Voronoi diagram are (a) non-empty, (b) path-connected, and (c) cover $\mathbb{R}^2$. Note that the nearest-color Voronoi diagram is related to the nearest-point Voronoi diagram of all endpoints of S: the region of $aa' \in S$ in the former diagram is the union of the regions of a and a' in the latter.

Our bisector system satisfies axioms (2), (3a) and (3c) since so does the bisector system of the nearest/farthest point Voronoi diagram. Further, since each $aa' \in S$ is an edge of the Delaunay triangulation of all endpoints of S, the regions of a and a' in the nearest-point Voronoi diagram are adjacent, thus their union is path-connected, implying axiom (3b). A bisector in our system satisfies axiom (1), since it separates two unions of pairs of adjacent regions in the diagram of four points. □

The faces of $\mathrm{FCVD}(S)$ near infinity coincide with the faces of the farthest-segment Voronoi diagram of S, thus, their sequence at infinity can be computed in $O(n \log n)$ time by divide and conquer (and other methods) [64]. Based on this observation, it is simple to derive a divide and conquer algorithm for $\mathrm{FCVD}(S)$. (Note that the approach in [58] yields an expected $O(n \log n)$ time algorithm for $\mathrm{FCVD}(S)$.)

Lemma 5.5.2. $\mathrm{FCVD}(S)$ *can be constructed in* $O(n \log n)$ *time and* $O(n)$ *space.*

Let $\mathrm{b_h}(a,b)$ denote the bisector of a and b.

Lemma 5.5.3. *For a point* $p \in \mathbb{R}^2$*, let* r_p *be the open ray with origin at* p *and direction* $\overrightarrow{ap}$*, where* a *is the endpoint of* $aa' \in S$ *such that* $p \in \overline{\mathrm{fcreg}}(a)$*. Let* $p \notin \mathrm{b_h}(a,a')$*. If* $r_p \cap \mathrm{b_h}(a,a') = \{q\}$*, then* $\mathrm{fcreg}(aa')$ *contains the open segment* pq*, as well as one of the two (unbounded) portions of* $\mathrm{b_h}(a,a')$ *starting at* q*. Otherwise,* $r_p \subset \mathrm{fcreg}(aa')$*.*

Figure 5.16. (a) $r_p \cap \mathrm{b_h}(a,a') = \emptyset$ and $D_p \subset D_x$. (b) $r_p \cap \mathrm{b_h}(a,a') = \{q\}$ and $D_{q\ell} \subset D_y$.

Proof. For any point $z \in \mathbb{R}^2$, let D_z be the disk centered at z of radius $d(z,a)$; see Figure 5.16.

Suppose that r_p does not intersect $\mathrm{b_h}(a,a')$. Since $p \in \overline{\mathrm{fcreg}}(a)$, disk D_p contains an endpoint of every segment in S. For a point $x \in r_p, x \neq p$, $D_p \subset D_x$. Thus D_x contains in its interior an endpoint of every segment in $S \setminus \{aa'\}$, that is, $x \in \mathrm{fcreg}(a) \subseteq \mathrm{fcreg}(aa')$.

Suppose next that r_p intersects $\mathrm{b_h}(a,a')$ in a point q. For any point $x \in pq$, $x \neq p$, we have $x \in \mathrm{fcreg}(a) \subset \mathrm{fcreg}(aa')$ by the above argument. In particular, disk D_q contains an endpoint of every segment in S. Point q breaks $\mathrm{b_h}(a,a')$ into two rays r_u and r_ℓ, which are respectively above and below q (see Figure 5.16b), and aa' breaks disk D_q into two parts D_{qu} and $D_{q\ell}$ that are above and below aa' respectively. (We assume that aa' is not vertical, otherwise the above/below relation can be replaced by left/right.) Observe that, if $\mathrm{fcreg}(aa')$ does not contain r_u (resp., r_ℓ), then $D_{q\ell}$ (resp., D_{qu}) contains an endpoint of some segment in $S \setminus \{aa'\}$. If $\mathrm{fcreg}(aa')$ contained neither r_u nor r_ℓ, there would be an endpoint of a segment in $S \setminus \{aa'\}$ inside $D_{q\ell}$, and an endpoint inside D_{qu}. A contradiction to aa' being an edge of the Delaunay triangulation of the set of endpoints of S. □

Lemma 5.5.4. $\mathrm{FCVD}(S)$ *can be preprocessed in* $O(n \log n)$ *time and* $O(n)$ *space so that a find-change query is answered in* $O(\log n)$ *time.*

Proof. By Lemma 5.5.1 $\mathrm{FCVD}(S)$ is a tree, and thus the *centroid decomposition* [22] can be built for it in $O(n \log n)$ time, and used to answer the find-change query. See Section 3.2.1.

To perform a query, we follow a root-to-leaf path (of length $O(\log n)$) in the tree of the centroid decomposition, at every node of the path one of the node's three subtrees is to be chosen. We can make a decision related to one node in $O(1)$ time, thus answering a find-change query in $O(\log n)$ time. Indeed, Lemma 5.5.3 if applied to v and each of the three regions of $\mathrm{FCVD}(S)$ incident to v, induces a decomposition of $\mathbb{R}^2$ into three regions of $O(1)$ combinatorial complexity, each of which contains one subtree of v in $\mathrm{FCVD}(S)$, see Figure 5.17a. Out of these three

Figure 5.17. (a) set $S = \{aa', bb', cc'\}$ (black); FCVD(S) (gray); the decomposition of $\mathbb{R}^2$ induced by its vertex v (red, dashed); (b) Illustration for the proof of Lemma 5.5.6

regions, in constant time we choose the only one that may contain the answer to the find-change query. □

We next bound the parameter m in Theorem 5.3.1.

Consider a pure edge $e = uv$ of $\mathrm{HVD}(S)$ separating $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$, for two segments $aa', bb' \in S$. Then $e \subseteq \mathrm{b_h}(a,b)$. We assume that segment ab is vertical with a on top of b, and that ab does not intersect the interior of e (otherwise e could be broken into two parts, considered separately). For any segment $cc' \in S$, we denote its supporting line by $\ell(cc')$.

Lemma 5.5.5. *If $cc' \in S$ is of type middle for S, then $\ell(cc')$ lies either above both aa', bb' or below them.*

Proof. One endpoint of cc' is in $D_h(u) \setminus D_h(v)$, and the other in $D_h(v) \setminus D_h(u)$. These two areas are separated by the vertical line $\ell(ab)$, so cc' is not vertical.

We first prove that it is impossible that a, b, c, c' are in convex position with c and c' not consecutive along the convex hull of the four points. Assume otherwise. The center of the circle through a, b and c lies on e; hence c' is outside this circle. Thus a and b are adjacent in the Delaunay triangulation of a, b, c, c'. Since this triangulation is plane, c and c' are not adjacent, and therefore they are not adjacent in the Delaunay triangulation of all endpoints of S; a contradiction.

Since c' (resp., c) is outside the circle through a, b and c (resp., c'), the convex hull of a, b, c, c' cannot be a triangle with c' (resp., c) in its interior. Hence, a and b are on the same side of $\ell(cc')$. Recall that a' and b' lie in $D_h(u) \cap D_h(v)$. Segment cc' either does not intersect $D_h(u) \cap D_h(v)$, or it divides $D_h(u) \cap D_h(v)$ in two portions, and both a, b lie in one of them. In both cases, the claim follows. □

Lemma 5.5.6. *If S satisfies the Delaunay property and all segments in S are of the same length, then an edge e of $\mathrm{HVD}(S)$ has at most two segments of type middle.*

Proof. We show that there is at most one segment of type middle whose supporting line is above aa', bb'. Then the claim follows from Lemma 5.5.5.

Suppose for contradiction that cc', dd' are segments of type middle for e such that $\ell(cc')$ and $\ell(dd')$ lie above aa', bb'. A vertical ray shot from a hits both cc' and dd'. Assume that it hits cc' first. Let D denote the disk through c, c', a. See Figure 5.17b. Since cc' is a Delaunay edge, cc' and dd' are pairwise disjoint, and a and at least one of d, d' are on opposite sides of cc', disk D contains none of d, d'.

We have $\angle dad' > \pi/2$: it is greater than the angle β formed by the two tangents to $D_h(u)$ and $D_h(v)$ at a (see blue dashed lines in Figure 5.17b) and $\beta \geq \pi/2$ by our assumption that segment ab does not intersect e in its interior. Let $s(cc')$ be the closed strip formed by two lines perpendicular to $\ell(cc')$ and passing through c and c' (tiled area in Figure 5.17b). We have: d, d' are outside D; d, d' are separated by $\ell(ab)$; $\angle dad' > \pi/2$; and $\ell(dd')$ lies above cc'. All this together imply that d and d' lie outside $s(cc')$ and on different sides of it. Thus $d(d, d') < d(c, c')$; a contradiction. □

Recall now Theorem 5.3.1, and recall that $|\mathrm{HVD}(S)| = O(n)$ and $\mathscr{T}_{\mathrm{HVD}(S)} = O(n \log n)$, since the segments in S are disjoint. By Lemma 5.5.4, $\mathscr{T}_{fc} = O(\log n)$; by Lemma 5.5.2, $|\mathrm{FCVD}(S)| = O(n)$ and $\mathscr{T}_{\mathrm{FCVD}(S)} = O(n \log n)$. If all segments in S are parallel, then $m = O(n)$, see Section A.2.1. By Lemma 5.5.6, m is also $O(n)$ if the segments in S have the same length. We conclude:

Theorem 5.5.1. *If S satisfies the Delaunay property and either all segments in S are parallel, or all segments in S are of equal length, then the stabbing circle problem can be solved in $O(n \log n)$ time and $O(n)$ space.*

Summary

The main result of this chapter is the connection between cluster Voronoi diagrams and the stabbing circle problem, and the method to solve the stabbing circle problem based on this connection.

For the case when all segments are parallel, we prove that our method works in $O(n \log^2 n)$ time, which improves considerably upon the best previously known bound, which was $O(n^2)$.

For segments that are disjoint edges of the Delaunay triangulation of some set of points (in this case we say that these segments *satisfy the Delaunay property*), and in addition that either have equal length, or are parallel to each other, we show that the stabbing circle problem can be solved in optimal $O(n \log n)$ time.

For the case of arbitrary segments, we note that the stabbing circle problem can be reduced to the problem of stabbing plane for a set of segments in $\mathbb{R}^3$. The latter problem can be solved in $O(n^2)$ time [38]. Since a matching $\Omega(n^2)$ lower bound on the number of combinatorially different stabbing circles is known [27], this is the tight worst-case time complexity for the stabbing circle problem.

In addition, we study the farthest-color Voronoi diagram for a family of pairs of endpoints of a segment set S. We prove: (1) If all segments in S are parallel to each other, the farthest-color Voronoi diagram on their endpoints has $O(n)$ combinatorial complexity and can be constructed in $O(n \log n)$ time. (2) If the segments in S satisfy the above Delaunay property, then the farthest-color Voronoi diagram is an instance of the *farthest abstract Voronoi diagram* [58]. Thus its combinatorial complexity is $O(n)$. (3) For segments satisfying the Delaunay property we give an algorithm to construct the farthest-color Voronoi diagram in deterministic $O(n \log n)$ time.

Chapter 6

Conclusion and Future directions

This dissertation has studied certain cluster Voronoi diagrams from the perspective of their construction algorithms and algorithmic applications. We focused on the (planar) Hausdorff Voronoi diagram and the farthest-segment Voronoi diagram, and additionally of the farthest-color Voronoi diagram (for particular inputs). The construction techniques we explore include two variations of the randomized incremental construction [26, 35, 48] and the linear-time divide-and-conquer technique of Aggarwal et al. [3, 51].

We have investigated the randomized incremental construction (RIC) paradigm for the Hausdorff Voronoi diagram (HVD). We presented the following algorithms: (1) a RIC for the HVD of *non-crossing*[1] clusters, based on point location; (2) a classic RIC for the HVD of non-crossing clusters (variants for a conflict graph and a history graph); and (3) a classic RIC for the HVD of arbitrary clusters, using a conflict graph. The expected time complexity of the first two algorithms is respectively $O(n\log^2 n)$ and $O(n\log n + k\log n\log k)$, where k is the number of input clusters, and n is the total number of points in all clusters. The third algorithm runs in expected $O((m+n\log k)\log n)$ time and $O(m+n\log k)$ space, where m is the total *number of crossings* between clusters. These results improve the state of the art on algorithms for constructing the Hausdorff Voronoi diagram with zero or small number of crossings. A small number of crossings was the case of interest in our motivating applications.

For the farthest-segment Voronoi diagram, we showed that it is possible to construct the diagram of n segments in $O(n)$ time, once the sequence of its faces at infinity is known. This result augments the Aggarwal et al. technique, granting it the ability to handle Voronoi regions with multiple faces.

We pointed out the connection between the stabbing circle problem and two

[1] For the definition of non-crossing clusters and the number of crossings, refer to Definition 7.

cluster Voronoi diagrams (the Hausdorff and the farthest-color Voronoi diagram). This is an important algorithmic application for these two diagrams. It implies a method to solve the stabbing circle problem. We prove that our method results in efficient algorithms for the certain sets of segments, i.e., when segments: (1) all are parallel to each other, (2) all are parallel to each other and satisfy the *Delaunay property*,[2] or (3) all have the same length and satisfy the Delaunay property. The time complexity of our algorithms is $O(n\log^2 n)$ for the first case, and $O(n\log n)$ for the two other cases. This considerably improves the known result prior to this dissertation.

In addition, we explored the farthest-color Voronoi diagram in the special cases, inspired by the stabbing circle problem. In particular, clusters are endpoints of segments, that are either all parallel to each other, or that satisfy the Delaunay property. We proved that the combinatorial complexity of the diagram in these cases is $O(n)$, and derive $O(n\log n)$ time construction algorithms.

6.1 Future directions

The results in this dissertation open several directions for further research. Some of them are listed below.

Directions regarding the Hausdorff Voronoi diagram

There is still a gap in the complexity of constructing the HVD of non-crossing clusters between our $O(n\log n + k\log n\log k)$ expected-time algorithm and the well-known $\Omega(n\log n)$ time lower bound. An open problem is to close or reduce this gap. It is interesting that in the L_∞ metric a simple $O(n\log n)$ time $O(n)$ space algorithm is known [67], which is based on a two-phase plane sweep.

Further, the concept of the Hausdorff Voronoi diagram is not restricted to clusters of points: we can generalize it to clusters of other simple objects, e.g., clusters of line segments or polygons. Such generalizations can be useful for the applications in VLSI computer-aided design [46]. More generally, it can be interesting to extend the framework of abstract Voronoi diagrams to handle clusters of sites, rather than individual sites. This was successfully accomplished for higher-order Voronoi diagrams [13].

[2] A set of segments satisfy the Delaunay property if all segments are edges of the Delaunay triangulation of some set of points, see Section 5.5.

The stabbing circle problem

Chapter 5 presents a solution to the stabbing circle problem as an application of cluster Voronoi diagrams. What is left behind the scene is a $\Omega(n \log n)$ lower bound for finding the stabbing circle of minimum radius [30]. Several questions are open regarding the stabbing circle problem.

One open problem is to find an algorithm that would operate more directly with $\mathrm{FCVD}^*(S)$, rather than searching for the candidate points which may or may not lie in $\mathrm{FCVD}^*(S)$. Such algorithm would better use the connection between the problem and cluster Voronoi diagrams. It would be desirable that the complexity of the solution would match better the complexity of the problem. Such an approach would be efficient when $|\mathrm{FCVD}^*(S)|$ is linear, as opposed to particular cases when the technical conditions on segments of type middle hold (see Lemma A.2.1 and Lemma 5.5.6).

Another interesting question for further investigation is how simple the following decision problem is: Given an arbitrary set of segments in the plane, decide if this set has at least one stabbing circle. Using our result in Chapter 5 we can answer this question in $O(n^2)$ time; however, together with answering this question our technique provides much more information, i.e., all the combinatorially different stabbing circles, and the ones with minimum and maximum radius. Is it possible to answer solely the decision problem in $o(n^2)$ time? This question is twofold: Working towards a positive answer means to design a new algorithm for the decision problem. Alternatively, working towards a negative answer would include, as a major step, proving that the decision part of the stabbing circle problem in *3SUM-hard* [43].

Preprocessing of proximity graphs for segment queries

Throughout this dissertation, multiple times there appeared a need for efficient queries that can be formulated as follows. Given a Voronoi diagram, and an edge e of another Voronoi diagram, find a point on e that satisfies certain conditions regarding the first Voronoi diagram. The examples of such queries in this dissertation include the *parametric point location*, the *segment query*, and *find-change query*. See Definitions 10, 11, and 25 respectively. In fact the time complexity of the algorithms in Chapters 3 and 5 depend on how fast such queries are performed.

The segment queries that have arised in our work can be seen as a generalization of the *point location queriy* which is the most widely used and demanded query when manipulating geometric and proximity structures. In case the Voronoi diagram under consideration is a tree, we were able to preprocess the diagram for

efficient segment and find-change queries, based on *centroid decomposition*, see Section 3.2.1. This resulted in an $O(n \log n)$ time algorithm to preprocess the corresponding Voronoi diagram (respectively, the farthest-point Voronoi diagram, and the farthest-color Voronoi diagram) for answering the above segment queries in $O(\log n)$ time. This matches the standard optimal results for point location [49, 32], although the segment queries are substantially more difficult. An interesting and important question is to drop the assumption that the Voronoi diagram is a tree and design a general preprocessing technique that would work for any plane straight-line proximity graphs.

Bibliography

[1] Manuel Abellanas, Gregorio Hernandez, Rolf Klein, Victor Neumann-Lara, and Jorge Urrutia. A combinatorial property of convex sets. *Discrete Comput. Geom.*, 17(3):307–318, 1997.

[2] Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. The farthest color Voronoi diagram and related problems. In *Proceedings of the 17th European Workshop on Computional Geometry (EWCG)*, pages 113–116, 2001. Full version (technical report): `http://tizian.cs.uni-bonn.de/publications/ahiklmps-fcvdr-06t.pdf`.

[3] 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(1):591–604, 1989.

[4] Helmut Alt, Otfried Cheong, and Antoine Vigneron. The voronoi diagram of curved objects. *Discrete & Computational Geometry*, 34(3):439–453, 2005.

[5] Lars Arge, Gerth Stolting Brodal, and Loukas Georgiadis. Improved dynamic planar point location. In *47th Ann. IEEE Symp. Found. Comput. Sci. (FOCS)*, pages 305–314, 2006.

[6] Boris Aronov, Prosenjit Bose, Erik D Demaine, Joachim Gudmundsson, John Iacono, Stefan Langerman, and Michiel Smid. Data structures for halfplane proximity queries and incremental Voronoi diagrams. In *LATIN 2006: Theor. Inf.*, pages 80–92, 2006.

[7] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[8] Franz Aurenhammer, Robert L. Scot Drysdale, and Hannes Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006.

[9] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. *Handbook of computational geometry*, 5:201–290, 2000.

[10] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, Singapore, 2013.

[11] David Avis, J. M. Robert, and Rephael Wenger. Lower bounds for line stabbing. *Inform. Process. Lett.*, 33(2):59–62, 1989.

[12] Hanna Baumgarten, Hermann Jung, and Kurt Mehlhorn. Dynamic point location in general subdivisions. *J. Algorithm.*, 17(3):342–380, 1994.

[13] Cecilia Bohler, Panagiotis Cheilaris, Rolf Klein, Chih-Hung Liu, Evanthia Papadopoulou, and Maksym Zavershynskyi. On the complexity of higher order abstract voronoi diagrams. *Computational Geometry*, 48(8):539–551, 2015.

[14] Cecilia Bohler, Rolf Klein, and Chih-Hung Liu. Forest-like abstract Voronoi diagrams in linear time. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG*, 2014.

[15] Jean-Daniel Boissonnat, Olivier Devillers, René Schott, Monique Teillaud, and Mariette Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete & Computational Geometry*, 8(1):51–71, 1992.

[16] Jean-Daniel Boissonnat, Camille Wormser, and Mariette Yvinec. Curved Voronoi diagrams. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 67–116. Springer Berlin Heidelberg, 2006.

[17] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*. Cambridge university press, 1998.

[18] Voronoi CAA: Voronoi Critical Area Analysis. IBM VLSI CAD Tool, IBM Microelectronics Division, Burlington, VT. Distributed by Cadence. Patents: US6178539, US6317859, US7240306, US7752589, US7752580, US7143371, US20090125852.

[19] CGAL. Computational Geometry Algorithms library. website: `http://www.cgal.org`.

[20] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[21] Bernard Chazelle, Leonidas J. Guibas, and Der-Tsai Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

[22] Panagiotis Cheilaris, Elena Khramtcova, Stefan Langerman, and Evanthia Papadopoulou. A randomized incremental algorithm for the Hausdorff Voronoi diagram of non-crossing clusters. *Algorithmica*, 2016. DOI 10.1007/s00453-016-0118-y.

[23] Otfried Cheong, Hazel Everett, Marc Glisse, Joachim Gudmundsson, Samuel Hornus, Sylvain Lazard, Mira Lee, and Hyeon-Suk Na. Farthest-polygon Voronoi diagrams. *Comput. Geom.*, 44(4):234–247, 2011.

[24] Francis Chin, Jack Snoeyink, and Cao An Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[25] Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185–212, 1993.

[26] Kenneth L. Clarkson and Peter Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4(1):387–421, 1989.

[27] Merce Claverol. *Problemas geométricos en morfología computacional*. PhD thesis, Universitat Politècnica de Catalunya, 2004.

[28] Merce Claverol, Delia Garijo, Clara I. Grima, Alberto Márquez, and Carlos Seara. Stabbers of line segments in the plane. *Comput. Geom.*, 44(5):303–318, 2011.

[29] Merce Claverol, Delia Garijo, Matias Korman, Carlos Seara, and Rodrigo I. Silveira. Stabbing segments with rectilinear objects. In Adrian Kosowski and Igor Walukiewicz, editors, *FCT 2015*, volume 9210 of *LNCS*, pages 53–64. Springer, 2015.

[30] Merce Claverol, Elena Khramtcova, Evanthia Papadopoulou, Maria Saumell, and Carlos Seara. Stabbing circles for some sets of delaunay segments. In *Abstracts of the 32th European Workshop on Computational Geometry (EuroCGâĂŹ16)*, 2016. to appear.

[31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

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

[33] Frank Dehne, Anil Maheshwari, and Ryan Taylor. A coarse grained parallel algorithm for Hausdorff Voronoi diagrams. In *Proceedings of the 35th International Conference on Parallel Processing (ICPP)*, pages 497–504, 2006.

[34] Olivier Devillers. An introduction to randomization in computational geometry. *Theoretical Computer Science*, 157(1):35 – 52, 1996.

[35] Olivier Devillers. The Delaunay Hierarchy. *Int. J. Found. Comput. S.*, 13(2):163–180, 2002.

[36] H. Edelsbrunner, H.A. Maurer, F.P. Preparata, A.L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22(3):274–281, 1982.

[37] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.

[38] Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, 4(1):311–336, 1989.

[39] Herbert Edelsbrunner, Lionidas J Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[40] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.

[41] Herbert Edelsbrunner and Roman Waupotitsch. Computing a ham-sandwich cut in two dimensions. *Journal of Symbolic Computation*, 2(2):171–178, 1986.

[42] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1–4):153–174, 1987.

[43] Anka Gajentaan and Mark Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[44] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM transactions on graphics (TOG)*, 4(2):74–123, 1985.

[45] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[46] Puneet Gupta and Evanthia Papadopoulou. Yield analysis and optimization. In *Handbook of Algorithms for Physical Design Automation*. 2008.

[47] Daniel P. Huttenlocher, Klara Kedem, and Micha Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput. Geom.*, 9(1):267–291, 1993.

[48] Menelaos I. Karavelas and Mariette Yvinec. The Voronoi diagram of planar convex objects. In *Algorithms - ESA 2003, 11th Annual European Symposium*, volume 2832 of *Lecture Notes in Computer Science*, pages 337–348. Springer Berlin Heidelberg, 2003. Full version (technical report): `http://hal.inria.fr/inria-00071561/PDF/RR-5023.pdf`.

[49] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[50] Rolf Klein. *Concrete and abstract Voronoi diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989.

[51] Rolf Klein and Andrzej Lingas. Hamiltonian abstract Voronoi diagrams in linear time. In *ISAAC*, pages 11–19, 1994.

[52] Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom.*, 3(3):157–184, 1993.

[53] Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Computers*, 31(6):478–487, 1982.

[54] Chi-Yuan Lo, Jiří Matoušek, and William Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11(4):433–452, 1994.

[55] Daniel N. Maynard and Jason D. Hibbeler. Measurement and reduction of critical area using Voronoi diagrams. In *Advanced Semiconductor Manufacturing Conference and Workshop, 2005 IEEE/SEMI*, pages 243–249, 2005.

[56] Nimrod Megiddo, Arie Tamir, Eitan Zemel, and Ramaswamy Chandrasekaran. An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems. *SIAM J. Comput.*, 10(2):328–337, 1981.

[57] Kurt Mehlhorn, Stefan Meiser, and Colm Ó'Dúnlaing. On the construction of abstract voronoi diagrams. *Discrete & Computational Geometry*, 6(1):211–224, 1991.

[58] Kurt Mehlhorn, Stefan Meiser, and Ronald Rasch. Furthest site abstract Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 11(6):583–616, 2001.

[59] Ketan Mulmuley. *Computational geometry: An introduction through randomized algorithms*, volume 54. Prentice-Hall Englewood Cliffs, NJ, 1994.

[60] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams, 2nd Ed.* John Wiley & Sons, 2000.

[61] Evanthia Papadopoulou. Critical area computation for missing material defects in VLSI circuits. *IEEE T. Comput. Aid. D.*, 20(5):583–597, 2001.

[62] Evanthia Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.

[63] Evanthia Papadopoulou. Net-aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. *IEEE T. Comput. Aid D.*, 30(5):704–716, 2011.

[64] Evanthia Papadopoulou and Sandeep K. Dey. On the farthest line-segment Voronoi diagram. *Int. J. Comput. Geom. Ap.*, 23(6):443–459, 2013.

[65] Evanthia Papadopoulou and Der-Tsai Lee. Critical area computation via Voronoi diagrams. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(4):463–474, 1999.

[66] Evanthia Papadopoulou and Der-Tsai Lee. The Hausdorff Voronoi diagram of polygonal objects: a divide and conquer approach. *Int. J. Comput. Geom. Ap.*, 14(6):421–452, 2004.

[67] Evanthia Papadopoulou and Jinhui Xu. The L_∞ Hausdorff Voronoi diagram revisited. *Int. J. Comput. Geom. Ap.*, 25(2):123–141, 2015.

[68] Evanthia Papadopoulou, Jinhui Xu, and Lei Xu. Map of geometric minimal cuts with applications. *Handbook of Combinatorial Optimization*, pages 1815–1869, 2013.

[69] Evanthia Papadopoulou and Maksym Zavershynskyi. The higher-order Voronoi diagram of line segments. *Algorithmica DOI 10.1007/s00453-014-9950-0*, 2014.

[70] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., 1985.

[71] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[72] Jacob T. Schwartz and Micha Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, 37(1):157–169, 1988.

[73] Raimund Seidel. *New Trends in Discrete and Computational Geometry*, chapter Backwards Analysis of Randomized Geometric Algorithms, pages 37–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[74] Raimund Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19(1):1–17, 1998.

[75] Michael I. Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.

[76] Micha Sharir. The Clarkson-Shor technique revisited and extended. *Comb. Probab. Comput.*, 12(2):191–201, 2003.

Appendix A

Complementary material

A.1 Appendix for Chapter 4: Computing the Order-(k+1) Subdivision within an Order-k Voronoi Region

Let f be a face of the order-k Voronoi region k-$reg(H)$, $H \subset S$, $|H| = k$. Let $S_f \subseteq S \setminus H$ consist of segments that induce the boundary ∂f. Consider the order-1 Voronoi diagram of S_f within f, $\mathcal{V}_1(S_f)$. As shown recently [69], $\mathcal{V}_1(S_f)$ is a tree structure, and the sequence of its faces along ∂f forms a Davenport-Schinzel sequence of order 4 (order 2 for non-intersecting segments). We can compute $\mathcal{V}_1(S_f)$ in time linear in the complexity of ∂f by slightly adapting the algorithms in the previous sections. This directly implies that the order-k Voronoi diagram of S can be computed in $O(k^2 n + n \log n)$ time, improved from $O(k^2 n \log n)$, by iteratively computing higher-order diagrams, starting at $\mathcal{V}_1(S)$.

The boundary ∂f can be viewed as a sequence of arcs, where each arc is a portion of the bisector $b(s,s')$ between a segment $s \in H$ and a segment $s' \in S_f$; see Figure A.2. An arc is delimited by two consecutive Voronoi vertices on ∂f, or is unbounded in one or two directions.

For a point $x \in \mathbb{R}^2$ and a segment $s \in S$, we let $\hat{r}(x,s)$ denote a line segment that realizes the Euclidean distance between x and s. Figure A.1 shows $\hat{r}(x,s)$ dashed. The following is a well-known property of the segment bisectors, which is the basis of the applicability of the algorithm of Section 4.4 to this setting.

Remark A.1.1. *Let s, s' be two line segments and $b(s,s')$ be their bisector. For any point $x \in \mathbb{R}^2$ such that $d(x,s) > d(x,s')$, the line segment $\hat{r}(x,s)$ intersects $b(s,s')$ in exactly one point. See Figure A.1.*

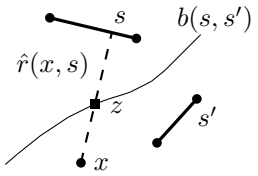


Figure A.1. Segments s, s'; $b(s, s')$ intersects $\hat{r}(x, s)$ (dashed) in point z

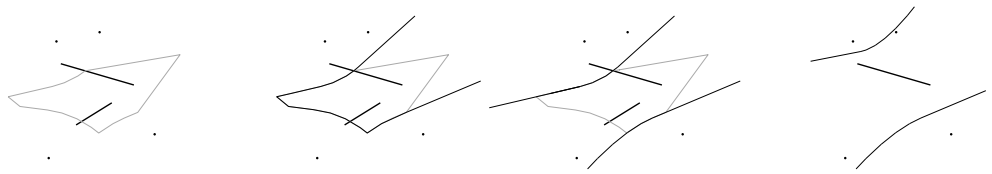
Similarly to the case of farthest-segment Voronoi diagram, the main object of our algorithms is an *arc sequence* F. Sequence F can be a subsequence of ∂f obtained by deleting several arcs, or an augmented subsequence of ∂f, obtained from a base subsequence by a number of operations, which are insertions of original arcs and merging with other augmented subsequences of ∂f. Such arc sequences are not necessarily connected, however, our algorithm will be operating with their connected components. Thus, unless clearly stated otherwise, we assume an arc sequence to be connected.

Basic operations with arc sequences and their Voronoi diagrams

Arc deletion. Let F be a subsequence of ∂f such that F has at least two maximal arcs. For an arc $\beta \in F$, β can be deleted F as follows. Suppose first that β has two neighbors α, γ in F. To delete β we remove its incident Voronoi vertices and extend α and γ along the corresponding bisectors. Two cases are possible: (1) the extended α and γ intersect, creating a new Voronoi vertex; or (2) they do not intersect, so the incident Voronoi edges become unbounded. In case (1), the arc sequence $F \ominus \beta$ is a subsequence of ∂f that has same arcs as F except β. In case (2), one of the following happens: (a) If F was a closed curve, then after deletion of β it becomes unbounded curve, and we do the necessary renaming so that the resulting arc sequence $F \ominus \beta$ has γ as the first arc, and α as the last arc. (b) If F was an unbounded curve, then the deletion of β breaks it into two curves, considered as two separate arc sequences F_1 and F_2, such that F_1 coincides with F from its beginning until the arc α, and α is unbounded; arc sequence F_2 coincides with F from γ (that is made unbounded) till the end of F.

Lemma A.1.1. *An arc sequence F, which is a subsequence or an augmented subsequence of ∂f may consist of at most two connected components.*

Proof. We prove the statement by induction on k, the order of the diagram, and thus the number of line segments in set H. We first prove the base case for $H = \{s\}$ ($k = 1$).



(a) (b) (c) (d)

Figure A.2. Two line segments and (a) their order-2 Voronoi region; (b) the first opening of the boundary after removing two elements; (c) the disconnection of the boundary into two connected components; (d) removing one of the two segments does not reduce the number of connected components

Consider two consecutive arcs α and γ of F_i that are not incident to each other, but they both extend to infinity. The unbounded portion of α towards infinity corresponds to an unbounded portion of $b(p,s_1)$ for an endpoint p of s_α and an endpoint s_1 of s. Similarly, the unbounded portion of γ is induced by an endpoint q of s_γ and s_1, corresponding to $b(q,s_1)$. But then p,s_1,q must be on the convex hull of s_α,s,s_γ in this order, thus, p and q must lie at opposite sides of the line l through s. Since any pair of consecutive unbounded arcs on ∂F_i must be induced by segments with endpoints at opposite sides of l, and since l has exactly two sides, we can have at most two such pairs of consecutive unbounded arcs along ∂F_i. Thus, ∂F_i can have at most two components.

Suppose the claim is true for a cluster H of k segments. Assume for the sake of contradiction that there is a cluster H of $k+1$ line segments, for which there exists a set of singleton-neighbors N such that the boundary of the Hausdorff region of H has at least three connected components. Each of the connected components separates H from at least one element of N. Now, if we remove a line segment from H, the Hausdorff region of H can only grow (see Figure A.2c,d), thus, no two connected components of its boundary may unite. No connected component may disappear, since the singletons in N, which are separated from H by these connected components, remain. Thus, the number of connected components remains at least three for a cluster of size k. We obtain a contradiction. □

If during the deletion process the arc sequence breaks in two components then we are initially computing two different overlapping diagrams, one for each component. If the arc that caused the splitting is re-inserted, these diagrams are merged into one. In the following, we consider a connected arc sequence, and define its Voronoi diagram in the same spirit as in Section 4.2.

Let F be a connected arc sequence. F breaks $\mathbb{R}^2$ into two connected domains; we let $D(F)$ be the one of these domains that contains H. Note that H never intersects F, and thus our definition is correct.

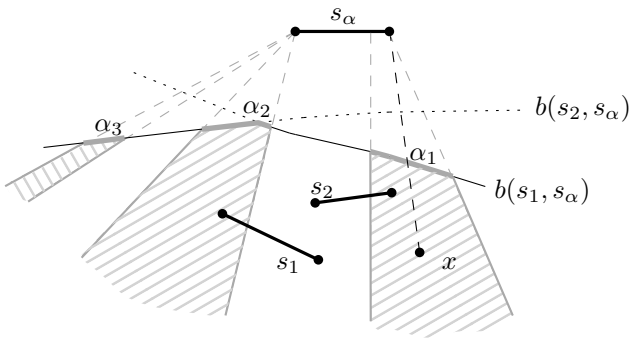


Figure A.3. Arcs $\alpha_1, \alpha_2, \alpha_3$, their attainable regions (tiling pattern); a point x attainable from α_1 and the segment $\hat{r}(x, s_\alpha)$

A point $x \in D(F)$ is said to be *attainable* from arc $\alpha \in F$, if $d(x, s_\alpha)$ is realized as a segment passing through α. Figure A.3 depicts the arcs $\alpha_1, \alpha_2, \alpha_3$ as bold gray portions of corresponding bisectors; the dashed black segment is $\hat{r}(x, s_\alpha)$. Similarly to Section 4.2, we define attainable region $R(\alpha)$ of arc α as the locus of points attainable from α. It is easy to see that the analog of Remark 4.2.1 holds for this setting, see Figure A.3.

Let $d(x, \alpha) = d(x, s_\alpha)$, if x is attainable from α, and $d(x, \alpha) = \infty$, otherwise. $\mathcal{V}_1(F)$ is a subdivision of $D(F)$ indused by the regions defined for each $\alpha \in F$: $reg(\alpha) = \{x \in D(F) \mid d(x, \alpha) < d(x, \gamma), \forall \gamma \in F, \gamma \neq \alpha\}$. Analogously to the proof of Lemma 4.2.1, $\cup_{\alpha \in F}(\overline{reg}(\alpha)) = D(F)$.

The graph structure of $\mathcal{V}_1(F)$ is a collection of trees. If we consider an artificial point at infinity and connect to it all unbounded bisectors, including the unbounded sides of F, then $\mathcal{V}_1(F)$ is a tree.

An arc bisector $b(\alpha, \gamma)$ ($s_\alpha \neq s_\gamma$) is a semicurve defined as $b(s_\alpha, s_\gamma) \cap D(F)$. For $s_\alpha = s_\gamma$, we can define an artificial bisector $b(\alpha, \gamma)$ using a point x on an arc β between α and γ visible from s_α. Let $b(\alpha, \gamma)$ be the ray r emanating from s_β, realizing $d(s_\beta, x)$, extending towards infinity away from s_β.

Both the randomized and the deterministic algorithm can now be directly applied obtaining a linear construction algorithm for $\mathcal{V}_1(F)$. There are two important arc removals and re-insertions: a simple one that converts ∂F_i into an open curve; and a crucial one that splits the open ∂F_i into two components. The re-insertion of this arc, requires time linear in the size of A_i^1 and A_i^2 to merge the two diagrams. In the randomized algorithm this is performed once. Similarly in the deterministic algorithm, an arc can participate in such an operation at most once (see Theorem A.1.1).

Note that ∂F_i^* need not be explicitly computed. Any arc bisector incident to

∂F_i^* intersects it after its origin, and no two neighboring bisectors can intersect before ∂F_i^*. Thus, we can create the tree structure of $\mathcal{V}_1(A_i^*)$ without computing its enclosing boundary.

Theorem A.1.1. *The order-$(k+1)$ subdivision in a face F of the order-k Voronoi diagram can be computed in time proportional to the complexity of ∂F.*

Proof. We only clarify some technical details that differ from Section 4.4. In Step 4 we apply the combinatorial lemma of [3] to $\mathcal{T}(A_i^*)$, which denotes the graph structure of $\mathcal{V}_1(A_i^*)$. Recall that the root of $\mathcal{T}(A_i^*)$ is an artificial point at infinity of arbitrary degree d, however, the combinatorial lemma assumes that all internal nodes are of degree 3. We can easily enforce this condition by inserting $d-1$ artificial nodes of degree 3 each, that would connect the children of the root. The artificial nodes have no effect in the algorithm that applies the combinatorial lemma (see [51]), and Lemma 4.4.2 is satisfied. This operation at most doubles the nodes of the tree, and thus, it has no effect in the complexity of the algorithm.

Another difference is the rebuilding operation, which in Step 7 may unite two connected components of ∂F_i, and their Voronoi diagrams, into one. In the recursion tree of the deterministic algorithm, each path from the root to a leaf corresponds to a specific permutation A_h of the set A. By Lemma A.1.1, for any permutation of A (and thus for any path in the recursion tree) there is at most one arc that disconnects ∂F_i into two connected components. Each original arc corresponds to one leaf of the recursion tree, and thus, to only one such path. Thus, each original arc participates in at most one of the expensive rebuilding operations. Each rebuilding operation takes time linear in the number of arcs participating in it, which is in turn linear in the number of original arcs involved. Therefore, in total during the whole execution of the algorithm, the rebuilding operations take time $O(h)$. □

Theorem A.1.1 includes updating a nearest-neighbor segment Voronoi diagram after the deletion of one segment in time proportional to the complexity of the deleted region.

A.2 Appendix for Chapter 5

Proof of Lemma 5.3.1

Lemma 5.3.1. *If the pair (t,s) satisfies the conditions of the input of Find-change query, then there exists a point w in the segment ts such that w is a changing point, $in \in type(w)$, or $out \in type(w)$.*

Proof. There are five generic types for the points of e, namely $\{in, out, \tilde{l}, \tilde{r}, mm\}$. The only situation where the lemma is not fulfilled is the following: segment ts is partitioned into subsegments of points whose only type is either $\tilde{r}$ or $\tilde{l}$, and two consecutive such subsegments are separated by a point having *mm* as type. In this case, when traveling along ts from left to right, we first find a subsegment of points containing t whose only type is $\tilde{r}$, then a point w having *mm* as type, and then a subsegment of points (possibly containing s) whose only type is $\tilde{l}$. We next argue that in this situation w is a changing point.

Suppose that *mm* with multiplicity one is the only element in $type(w)$ (due to our assumption that no four endpoints of segments in S are cocircular, *mm* cannot be in $type(w)$ with multiplicity greater than one). Let cc' be the segment in S such that $mm \in type(w')$ caused by cc'. We assume that c' is contained in $D_h(u) \setminus D_h(v)$, and c in $D_h(v) \setminus D_h(u)$. Then $w \in \mathrm{fcreg}(cc')$ and the farthest-color disk $D_f(w)$ has on the boundary c, c', and no other endpoint of any segment in S. Consequently, $w_\ell \in \mathrm{fcreg}(c')$. This implies that $type(w'_\ell) = \{ml\}$, which contradicts the fact that the subsegment to the left of w has only type $\tilde{r}$. Thus, there is another element in $type(w)$ apart from *mm*, and this element can only be $\tilde{r}$. Arguing analogously with w_r, we find that $\tilde{l} \in type(w)$.[1] We conclude that w is a changing point. □

Proof of Lemma 5.3.3

Lemma 5.3.3. *The algorithm* Search-In *e computes all faces of* $\mathrm{FCVD}^*(S)$ *intersected by e.*

Proof. Let $e = uv$. Segment uv satisfies the input condition of the algorithm: If uv is shrunk infinitesimally from both sides, its endpoints are outside $\mathrm{FCVD}^*(S)$. This follows from Steps 13–14 of the algorithm *Computing* $\mathrm{FCVD}^*(S)$.

The condition checked in Step 2 of the algorithm is justified by Lemma 5.3.2: If the condition is not satisfied, then $uv \cap \mathrm{FCVD}^*(S) = \emptyset$ and we can safely stop the search at Step 15. Otherwise, the input conditions of the find-change query are satisfied.

By Lemma 5.3.1, the find-change query is well-defined, and there are three possible outputs when we perform it on (u, v). If the query returns a point w such that $out \in type(w)$, then $e \cap \mathrm{FCVD}^*(S) = \emptyset$. In this case, the algorithm stops (see Steps 5–6). Otherwise, a face of $\mathrm{FCVD}^*(S)$ containing w is traced if and only if $w \in \mathrm{FCVD}^*(S)$.

[1] At this point, it is also possible to derive a contradiction to the general position assumption that no four endpoints of segments of S are cocircular. However, for the sake of generality, we refrain from relying on this assumption.

The algorithm is called recursively for two subsegments of uv, which are uq and $q'v$ (Steps 12–13). We now show that each of these subsegments satisfies the input condition of the algorithm. Indeed, points u_r and v_ℓ are outside $\mathrm{FCVD}^*(S)$ because uv was satisfying this condition in the first place. Points q and q' are determined in Step 9 or Step 11, depending on the condition in Step 7. In particular, if the condition is satisfied, i.e., w lies in a face f of $\mathrm{FCVD}^*(S)$, then $e \cap f$ is a line segment (see Lemma 5.2.10). In this case, Step 9 is executed, and it assigns to q and q' the left and right endpoints of the segment $e \cap f$, respectively. This implies that q_ℓ and q'_r are outside $\mathrm{FCVD}^*(S)$. Otherwise, Step 11 assigns both q, q' to w and, since w is not in $\mathrm{FCVD}^*(S)$, neither are w_ℓ, w_r.

In Theorem 5.3.1, we analyze the running time of the algorithm, and in particular we prove the algorithm terminates. □

Proof of Lemma 5.3.7

Lemma 5.3.7. *Let $f_1, f_2, \ldots, f_k$ be the faces of $\mathrm{FCVD}^*(S)$. The total number of edges of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ intersected by these faces is $O(k + |\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$.*

Proof. Let I denote the total sum $\sum_{i=1}^{k} I(f_i)$, where $I(f_i)$ denotes the number of edges of $\mathrm{HVD}(S)$ and $\mathrm{FCVD}(S)$ that are intersected by f_i. We need to show that $I = O(k + |\mathrm{HVD}(S)| + |\mathrm{FCVD}(S)|)$. By Lemma 5.2.8, for each face f_i, $f_i \cap \mathrm{HVD}(S)$ is a connected component. Such connected component can be either (1) a portion of a single edge of $\mathrm{HVD}(S)$; or (2) a component containing portions of at least two edges. Depending on this, we call face f_i a type-1 face, or a type-2 face respectively. Let t be the number of type-1 faces, and r be the number of type-2 faces; then $t + r = k$. Clearly, all type-1 faces contribute t to the total sum I. Further, one edge of $\mathrm{HVD}(S)$ intersects at most two type-2 faces. Thus all type-2 faces contribute at most $2 * |\mathrm{HVD}(S)|$ to I. Since $t \leq k$, $\mathrm{HVD}(S)$ contributes at most $k + 2 * |\mathrm{HVD}(S)|$ to I. By an analogous argument, $\mathrm{FCVD}(S)$ contributes at most $k + 2 * |\mathrm{FCVD}(S)|$ to I. The claim follows. □

A.2.1 Segments of type middle for a set of parallel segments

The key result of this subsection is the following:

Lemma A.2.1. *A segment $gg' \in S$ is of type middle for at most one pure edge of $\mathrm{HVD}(S)$.*

We first prove an easy property of the segments of type middle:

Lemma A.2.2. *Suppose that all segments in S are vertical. Let e be a pure edge of $\mathrm{HVD}(S)$ in the boundary of $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$, for two segments $aa', bb' \in S$. Suppose that segment $gg' \in S$ is of type middle for e. Then:*

(a) $x(a) \neq x(b)$.

(b) $\min\{x(a), x(b)\} < x(g) < \max\{x(a), x(b)\}$.

Proof. To prove (a), we suppose for the sake of contradiction that $x(a) = x(b)$. Then e is horizontal. Let u and respectively v be the left and right endpoints of e. Since the segment gg' is of type middle, it has one endpoint to the left of the line $x = x(a)$, in $D_h(u) \setminus D_h(v)$, and the other endpoint to the right of the line $x = x(a)$, in $D_h(v) \setminus D_h(u)$ (see Figure A.4, left). This contradicts the fact that gg' is vertical.

To prove (b), we suppose without loss of generality that $x(a) < x(b)$. Then e is not horizontal, and we denote by u and respectively v the top and bottom endpoints of e.

For any disk D, we can divide its boundary ∂D into the left-most point of ∂D, the open circular arc containing the upper half portion of ∂D (called *top chain*), the right-most point of ∂D, and the open circular arc containing the lower half portion of ∂D (called *bottom chain*). Since aa' is vertical and $D_h(u)$ contains both a and a', a is not the left-most or right-most point of $\partial D_h(u)$. Suppose that a belongs to the top chain of $\partial D_h(u)$. Then we deduce that $y(a') < y(a)$. Since $D_h(v)$ also contains both a and a', we get that a belongs to the top chain of $\partial D_h(v)$. So a belongs to the top (resp., bottom) chain of $\partial D_h(u)$ if and only if a belongs to the top (resp., bottom) chain of $\partial D_h(v)$. The same argument applies to b.

Now there are several possibilities, depending on a and b being in the top or bottom chains of $\partial D_h(u)$ and $\partial D_h(v)$. The arguments for all cases are similar, so we only explain the case where a and b belong to the top chains of $\partial D_h(u)$ and $\partial D_h(v)$. In this case, u and v lie in the portion of $\mathrm{b_h}(a,b)$ below the lines $y = y(a)$ and $y = y(b)$ (and recall that $y(u) > y(v)$) (see Figure A.4, center). Either g or g' lies in $D_h(u) \setminus D_h(v)$, so in particular in the portion of $D_h(u)$ above the segment ab. Since a and b belong to the top chain of $\partial D_h(u)$, this portion lies between the lines $x = x(a)$ and $x = x(b)$. Thus we obtain $x(a) < x(g) < x(b)$. □

We are now ready to prove Lemma A.2.1.

Proof of Lemma A.2.1. We assume that all segments in S are vertical. We proceed by contradiction. So let us assume that the segment gg' is of type middle for two pure edges of $\mathrm{HVD}(S)$, namely e_1 and e_2. Let e_1 be in the boundary of $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$, for two segments $aa', bb' \in S$ (see Figure A.4, right). Analogously, e_2 is in the boundary of $\mathrm{hreg}(c)$ and $\mathrm{hreg}(d)$, for two segments $cc', dd' \in S$. By

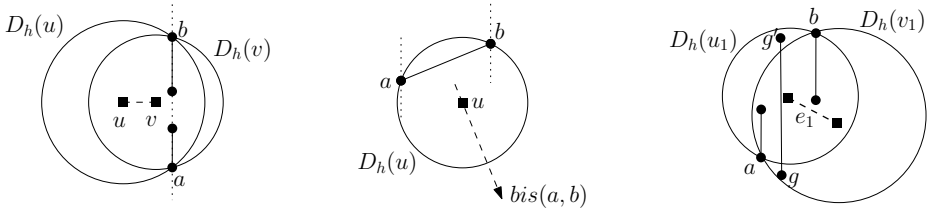


Figure A.4. Left: Case where $x(a) = x(b)$. Center: Either g or g' lies in a portion of $D_h(u)$ between the lines $x = x(a)$ and $x = x(b)$. Right: The segment gg' is of type middle for e_1.

Lemma A.2.2a, $x(a) \neq x(b)$ and $x(c) \neq x(d)$. Without loss of generality, we suppose that $x(a) < x(b)$ and $x(c) < x(d)$. We also assume that $y(g) < y(g')$.

Consider the disk having a, b and g on the boundary; this disk corresponds to a disk $D_h(w_1)$, for some point w_1 on the edge e_1. Analogously, the disk having c, d and g on the boundary corresponds to a disk $D_h(w_2)$, for some point w_2 on e_2.

Since, by Lemma A.2.2b, $x(a) < x(g) < x(b)$, the line $x = x(g)$ intersects $\partial D_h(w_1)$ twice. One of these intersection points is g, and we next show that the second intersection point, called t_1, is above g. Since the line through a and b leaves g and g' on opposite sides, and since $x(a) < x(g) < x(b)$, the segment ab intersects the segment gg'. The intersection point lies above g and, by convexity, it is contained in $D_h(w_1)$. This implies that g is in the bottom chain of $\partial D_h(w_1)$ and, consequently, t_1 is above g (see Figure A.5, left). Analogously, the second intersection point t_2 between $\partial D_h(w_2)$ and $x = x(g)$ also lies above g. Without loss of generality, we assume that $y(t_1) \geq y(t_2)$. We divide the rest of the argument into several cases.

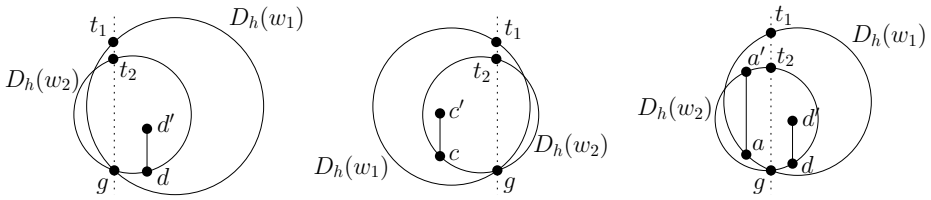


Figure A.5. Left: Case where the four segments are distinct and the second intersection point between $\partial D_h(w_1)$ and $\partial D_h(w_2)$ is to the left of $x = x(g)$. Middle: Case where the four segments are distinct and the second intersection point between $\partial D_h(w_1)$ and $\partial D_h(w_2)$ is to the right of $x = x(g)$. Right: Case where $a = c'$.

We start by considering the case where the four segments aa', bb', cc', and dd' are distinct. Because w_1 is in the boundary of $\mathrm{hreg}(a)$ and $\mathrm{hreg}(b)$ and the four segments are distinct, $D_h(w_1)$ contains at most one of $\{c,c'\}$ and at most one of $\{d,d'\}$. Analogously, $D_h(w_2)$ contains at most one of $\{a,a'\}$ and at most one of $\{b,b'\}$. Consequently, none of $D_h(w_1), D_h(w_2)$ contains the other, and $\partial D_h(w_1)$ and $\partial D_h(w_2)$ intersect at g and at a second point. If this second point lies to the left of the line $x = x(g)$, then the portion of $D_h(w_2)$ to the right of $x = x(g)$ is contained in $D_h(w_1)$ (see Figure A.5, left). Consequently, dd' is in $D_h(w_1)$, yielding a contradiction. If the second intersection point between $\partial D_h(w_1)$ and $\partial D_h(w_2)$ lies to the right of the line $x = x(g)$, then cc' is in $D_h(w_1)$ (see Figure A.5, center). If the intersection point lies on $x = x(g)$, then $y(t_1) = y(t_2)$, and we obtain that $D_h(w_1)$ contains cc' or dd'.

Let us look at the remaining cases. Since $x(a) < x(g) < x(b)$ and $x(c) < x(g) < x(d)$, the intervals $(x(a),x(b))$ and $(x(c),x(d))$ have non-empty intersection. This implies that $aa' \neq dd'$ and $bb' \neq cc'$ (and obviously $aa' \neq bb'$ and $cc' \neq dd'$). Therefore, the two remaining cases are $aa' = cc'$ and $bb' = dd'$. We divide the first one into two subcases, namely, $a = c$ and $a = c'$.

If $a = c$, the second intersection point between $\partial D_h(w_1)$ and $\partial D_h(w_2)$ is a, which lies to the left of the line $x = x(g)$. If $bb' \neq dd'$, then dd' is contained in $D_h(w_1)$, yielding a contradiction. If $bb' = dd'$, then, since b lies on $\partial D_h(w_1)$ and d lies on $\partial D_h(w_2)$, we have that $b \neq d$. But then $b = d'$ lies on the portion of $\partial D_h(w_1)$ to the right of $x = x(g)$, that is, outside $D_h(w_2)$, yielding a contradiction.

If $a = c'$, due to the assumption that $y(t_1) \geq y(t_2)$, we have that a is in the bottom chain of $\partial D_h(w_1)$ and $a' = c$ is in the top chain of $\partial D_h(w_2)$ (see Figure A.5, right). Then the second intersection point between $\partial D_h(w_1)$ and $\partial D_h(w_2)$ lies to the left of $x = x(a)$, and we also have that dd' is in $D_h(w_1)$ (if $bb' \neq dd'$) or that d' is outside $D_h(w_2)$ (if $bb' = dd'$).

In the last case, $bb' = dd'$. This case is symmetric to the previous one, and it also yields a contradiction. □