Voronoi diagrams in the max-norm: Algorithms, implementation, and applications

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

Sandeep Kumar Dey

under the supervision of

Prof. Evanthia Papadopoulou

June 2015

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.

Sandeep Kumar Dey
Lugano, June 2015

To my loved ones

I have no special talent. I am only
passionately curious . . .

Albert Einstein

Abstract

Voronoi diagrams and their numerous variants are well-established objects in computational geometry. They have proven to be extremely useful to tackle geometric problems in various domains such as VLSI CAD, Computer Graphics, Pattern Recognition, Information Retrieval, etc. In this dissertation, we study generalized Voronoi diagram of line segments as motivated by applications in VLSI Computer Aided Design. Our work has three directions: *algorithms, implementation, and applications of the line-segment Voronoi diagrams*. Our results are as follows:
(1) Algorithms for the *farthest Voronoi diagram* of line segments in the L_p metric, $1 \leq p \leq \infty$. Our main interest is the L_2 (Euclidean) and the L_∞ metric. We first introduce the *farthest line-segment hull* and its *Gaussian map* to characterize the regions of the farthest line-segment Voronoi diagram at infinity. We then adapt well-known techniques for the construction of a convex hull to compute the farthest line-segment hull, and therefore, the farthest segment Voronoi diagram. Our approach unifies techniques to compute farthest Voronoi diagrams for points and line segments.
(2) The implementation of the L_∞ Voronoi diagram of line segments in the Computational Geometry Algorithms Library (CGAL)[1]. Our software (approximately 17K lines of C++ code) is built on top of the existing CGAL package on the L_2 (Euclidean) Voronoi diagram of line segments. It is accepted and integrated in the upcoming version of the library CGAL-4.7 and will be released in september 2015. We performed the implementation in the L_∞ metric because we target applications in VLSI design, where shapes are predominantly rectilinear, and the L_∞ segment Voronoi diagram is computationally simpler.
(3) The application of our Voronoi software to tackle proximity-related problems in VLSI pattern analysis. In particular, we use the Voronoi diagram to identify critical locations in patterns of VLSI layout, which can be faulty during the printing process of a VLSI chip. We present experiments involving layout pieces that were provided by IBM Research, Zurich. Our Voronoi-based method was able to find *all* problematic locations in the provided layout pieces, very fast, and without any manual intervention.

[1]http://www.cgal.org/

Acknowledgements

I feel privileged and take this opportunity to express my sincere gratitude to the supervisor of this dissertation, Evanthia Papdopoulou. Her command over the area of my work has been of great help for my research. She not only suggested directions towards the solution of the problems but also helped me in all aspects including the preparation of this manuscript, and have also given me full freedom to think and work independently. This work has been possible only because of her continuous suggestions, inspiration, and motivation given to me to incorporate my ideas.

I also take this opportunity to thank the members of my dissertation committee - Gill Barequet, Menelaos Karavelas, Kai Hormann, and Michael Bronstein - for dedicating their time to evaluate my work, and providing their valuable feedback. I am grateful to Menelaos Karavelas for providing insight on CGAL way of coding and to Gill Barequet for discussions on combinatorial bounds of farthest line-segment Voronoi diagram.

I am grateful to Maria Gabrani, Nathalie Casati, and Henri Saab of IBM Research, Zurich, for their contribution in the work on VLSI pattern analysis, which is an important portion of this thesis. Their suggestions and comments were invaluable.

It was a pleasure to work with our research team at USI - Panagiotis Cheilaris, Maksym Zavershynskyi, and Elena Khramtcova. Your continous support and encouragement helped me during difficult phases of my work. My special thanks to Panagiotis Cheilaris who is also my co-author in two papers, mentored me for the CGAL implementation of the line-segment Voronoi diagram in the max-norm. Your continuous guidance and support was an important factor for finishing this dissertation. I also like to thank Decanato for their help and support, you took care of all the official paper works for attending conferences, salaries, contracts, letters, always helped with a bright smile.

I am specially thankful to my friends Randolf Schärfig, Nihat Engin Toklu, Dmitry Anisimov and Artiom Kovnatsky for their valuable suggestions and comments during all phases of my stay in Lugano. Let it be life or studies, I always felt strong and motivated because of you. Discussions on programming, algorithms, data structures were very fruitful and they contributed to essential part of my dissertation. I am thankful to Alexander Förster for giving me very valuable tips on C++ coding. I am grateful to Anadi Mishra, Suchismita Bannerjee, and Abhishek Bhattacharjee for always being there for me.

From the bottom of my heart, I thank my parents, sister Papri, and my wife Ruchira for their endless love and support.

Sandeep Kumar Dey
June 2015

Contents

Figures

Tables

Chapter 1

Introduction

"A person who never made a mistake never tried anything new."

Albert Einstein

The Voronoi diagram [5, 7, 61] is a powerful geometric object that encodes proximity information for a given set of simple geometric objects, called sites. These sites are typically points, line-segments or curves. Voronoi diagrams can also be defined for complex geometric objects, where a complex geometric object is a combination of basic geometric objects.

The *nearest neighbor Voronoi diagram* of a given set S of n sites in the plane is a subdivision of the plane into regions, called Voronoi regions, such that all points within the region of a site $s \in S$, are closer to s than to any other site in S. Formally, the Voronoi region of a site s is given by:

$$\mathit{reg(s)} = \{x \in \mathbb{R}^2 \mid \forall t \in S \setminus \{s\},\ d(x,s) \leq d(x,t)\}, \tag{1.1}$$

where $d(x,s)$ denotes the distance between a point x and site s under a given metric. The site s is called the *owner* of the Voronoi region *reg(s)*. The partitioning of the plane derived by the union of all such regions formed by the sites in S, together with their bounding edges and vertices, defines the nearest neighbor Voronoi diagram of S.

Among many variations of the Voronoi diagram, the simplest and the most popular one is the Voronoi diagram of points. Figure 1.1 (a) illustrates the Voronoi diagram of five points p_1, p_2, p_3, p_4, and p_5. The grey shaded region is the Voronoi region of point p_4, which means that all the points in the shaded

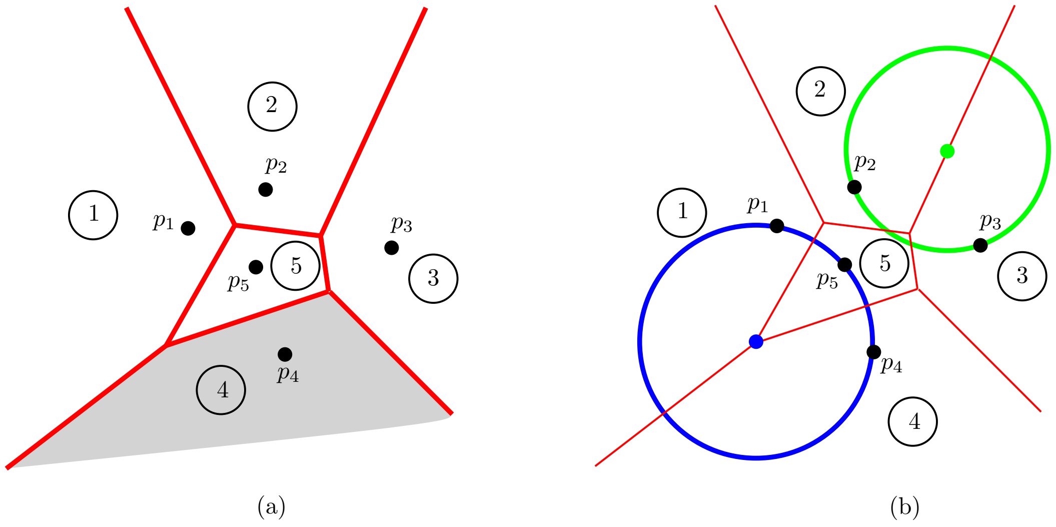
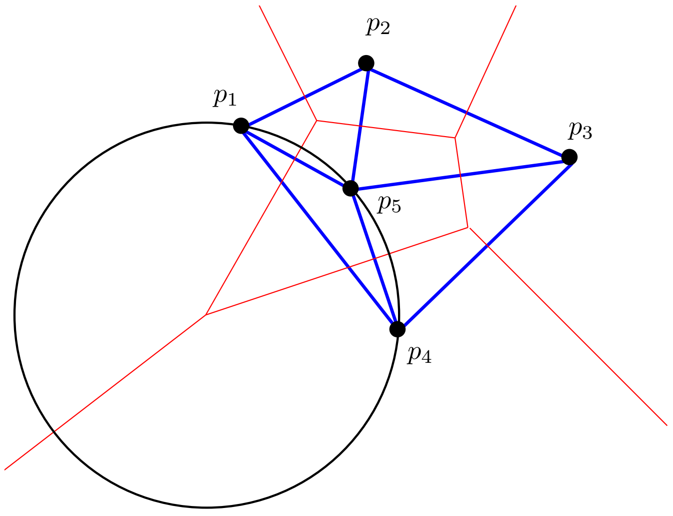


Figure 1.1. (a) Example of nearest neighbor Voronoi diagram (in red) of five points p_1, p_2, p_3, p_4, p_5 in the Euclidean metric, (b) showing the empty circles on the Voronoi diagram of points.

region are closer to the point p_4 than to other input points in the example.

In the nearest neighbor Voronoi diagram every input site has a non-empty Voronoi region. The boundary separating two neighboring Voronoi regions is called a *Voronoi edge* and it is equidistant from the corresponding owners. The locus of points equidistant from two sites is called the *bisector* of the sites, thus, Voronoi edges are portions of the bisector of the neighboring owner sites. By the definition of the Voronoi edge, any circle with center on a Voronoi edge and passing through (touching at a single point) the two corresponding owner sites, must be empty (for example, the green circle in Figure 1.1 (b)), that is it does not contain other sites in its interior. Voronoi edges meet at a *Voronoi vertex*. Any circle with center on a Voronoi vertex and passing through the corresponding owner sites must be empty (for example, the blue circle in Figure 1.1 (b)).

Another important geometric structure related to the Voronoi diagram is its dual, known as the *Delaunay triangulation*. Voronoi [5, 7] was the first to consider the dual structure of a Voronoi diagram, where he mentioned that any two points of the given point set are connected if their Voronoi regions have a common boundary. Later, Delaunay [5, 7] obtained the same by defining that two points are connected if they lie on a circle whose interior is empty. After his



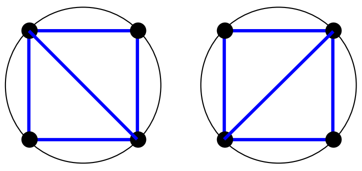


Figure 1.2. Example of a Delaunay triangulation of points: (a) empty circle in black, Delaunay triangulation in blue, (b) Two Delaunay triangulations for four co-circular points.

name, the dual structure of a Voronoi diagram is called Delaunay tessellation or Delaunay triangulation. The graph structure of the Delaunay tessellation is called the *Delaunay graph*. The Delaunay graph can be defined for any set of simple geometrical sites, where the sites are the nodes, and any two nodes are connected by an edge if they share a Voronoi edge. For simplicity we explain the concept of a Delaunay graph using point sites in general position[1], where the Delaunay graph is simply a Delaunay triangulation. Let P be a set of points in the general position. Then, three points of P give rise to a Delaunay triangle if and only if their circumcircle does not contain any other point of P in its interior [5, 7, 61] (for example see the circle in the Figure 1.2 (a)). The triangulation formed from points of P following the definition of a Delaunay triangle, is known as the Delaunay triangulation of P (see the triangulation in blue in the Figure 1.2 (a)). If more than three points are on the same circle (e.g. the vertices of a square), then the Delaunay graph is no longer a triangulation, nevertheless, different Delaunay triangulations are possible (see Figure 1.2 (b): each of the two possible triangulations that split the square into two triangles satisfies the definition of a Delaunay triangle).

[1]No three points are on the same line that is collinear and no four points are on circle that is co-circular.

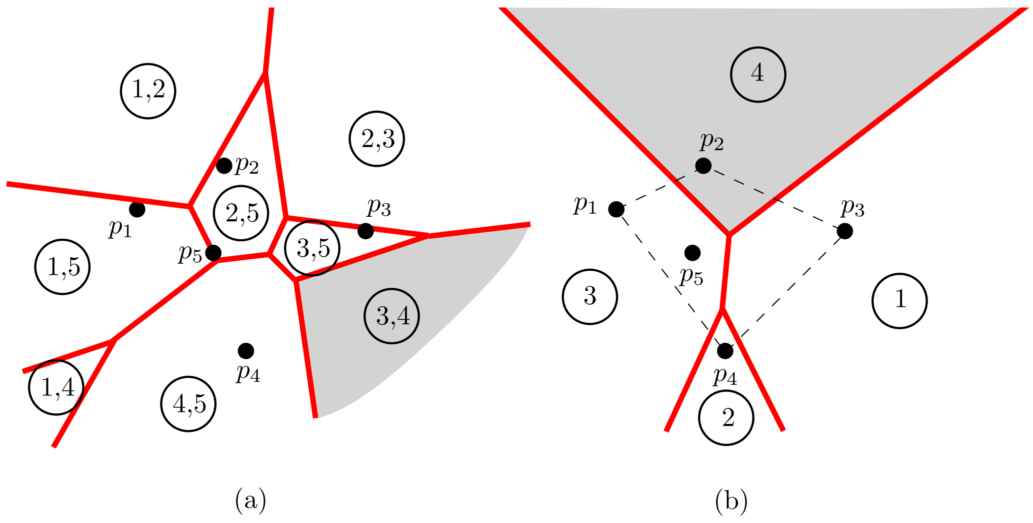


Figure 1.3. Examples of Voronoi diagram (in red) of five points p_1, p_2, p_3, p_4, p_5 in the Euclidean metric: (a) second order, and (b) farthest.

We will now discuss briefly a more general variant of the nearest neighbor Voronoi diagram. The *higher order Voronoi diagram* is a generalization of the nearest neighbor Voronoi diagram. The Voronoi region of a k-order Voronoi diagram ($1 \leq k < n$) is the locus of points closer to a set $H \subseteq S$ of k sites than to any other site $t \in S \setminus H$. The k-order Voronoi region of $H \subset S$, $|H| = k$:

$$kreg(H) = \{x \in \mathbb{R}^2 \mid \forall s \in H,\ \forall t \in S \setminus H,\ d(x,s) \leq d(x,t)\} \qquad (1.2)$$

The partitioning of the plane derived by the union of all such regions formed by the subsets of S of size k, together with their bounding edges and vertices, defines the k-order Voronoi diagram of S. Figure 1.3 (a) illustrates the 2-order Voronoi diagram of points. Every region in the example has two owners. For example, the shaded region has owners, p_3 and p_4, which means that all the points in the shaded region are closer to either p_3 or p_4 than to any other input points.

For $k = n - 1$, the higher order Voronoi diagram yields the *farthest site Voronoi diagram*. The farthest Voronoi region of a site $s \in S$, is the locus of points which are farther from s than from any other site in S. It corresponds to the order-(n—1) Voronoi region of $S \setminus \{s\}$.

$$freg(s) = kreg(S \setminus \{s\}) \qquad (1.3)$$

The farthest Voronoi region of a site s can be defined equivalently as:

$$\mathit{freg(s)} = \{x \in \mathbb{R}^2 \mid \forall t \in S \setminus \{s\},\ d(x,s) \geq d(x,t)\} \tag{1.4}$$

For point sites the farthest Voronoi diagram has been well studied. In the farthest point Voronoi diagram, the Voronoi regions are only contributed by the points on the *convex hull*[1] of the input set of points. Figure 1.3(b) illustrates example of farthest Voronoi diagram of points. The shaded region is the farthest Voronoi region of the point p_4, that means all the points in the shaded region are farther to p_4 than any other input point in the given example. Observe that the point p_5 does not have any farthest Voronoi region as it is not a part of the convex hull of the input points in the example (shown by black dashed lines in Figure 1.3(b)).

Interestingly, for line-segments the farthest Voronoi diagram illustrates different properties from its counterpart for points [6]. For example, the farthest Voronoi regions are not defined by the convex hull properties, and a line-segment can have disconnected farthest Voronoi regions. Figure 1.4 illustrates an example of farthest Voronoi diagram of line-segments [6]. The two disconnected shaded regions in the figure are Voronoi regions of line-segment s_5. We discuss more on structural properties and construction algorithms of farthest line-segment Voronoi diagram in Chapter 2.

Voronoi diagrams can be defined for various metrics for various simple sites, and in different dimensions. In the general L_p metric, distance between two points $a = (x_a, y_a)$ and $b = (x_b, y_b)$ is defined as $\sqrt[p]{\mid x_b - x_a \mid^p + \mid y_b - y_a \mid^p}$. For $p = 2$, we have, $\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$, which is the Euclidean (L_2) distance between a and b. This dissertation focuses on 2-D Voronoi diagrams of line-segments in the L_∞ metric. The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is the maximum between the horizontal and the vertical distance between p and q, i.e., $d(p,q) = \max\{d_x(p,q), d_y(p,q)\}$, where $d_x(p,q) = \mid x_p - x_q \mid$ and $d_y(p,q) = \mid y_p - y_q \mid$. Figure 1.5 (a), (b), and (c) illustrates the nearest neighbor, 2-order, and the farthest Voronoi diagram of points in the max-norm. The Voronoi diagram in the L_1 (Manhattan distance) metric is equivalent to the Voronoi diagram in the L_∞ metric under a 45 degree rotation.

[1]The convex hull of a point set $\mathscr{P}$ in 2D is the smallest convex set $\mathscr{C}$ containing $\mathscr{P}$. Convex set $\mathscr{C}$: $\forall p_i, p_j \in \mathscr{C}$ the line-segment $\overline{p_i p_j}$ is also in $\mathscr{C}$.

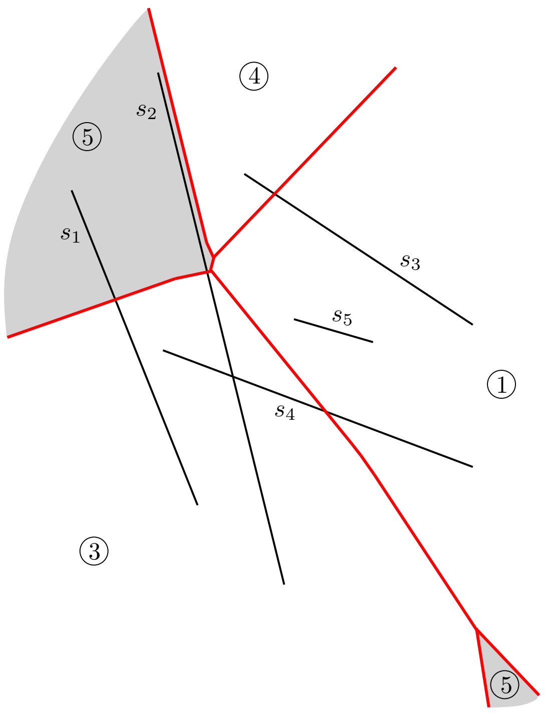


Figure 1.4. Example of a farthest line-segment Voronoi diagram in the Euclidean plane.

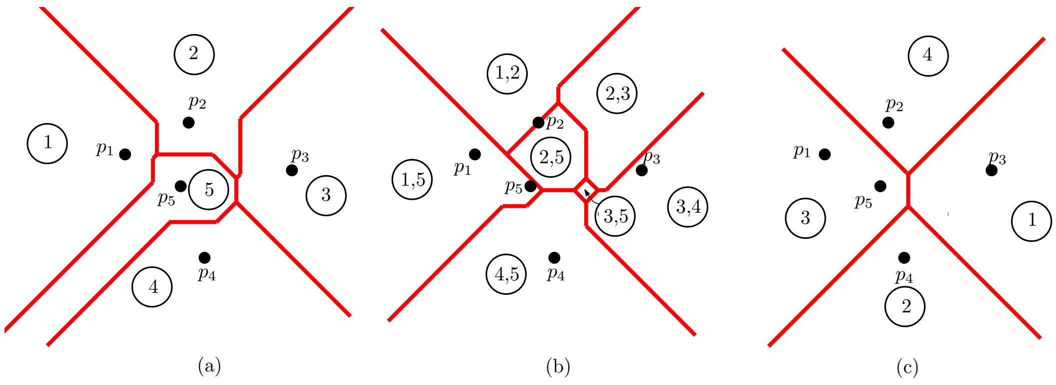
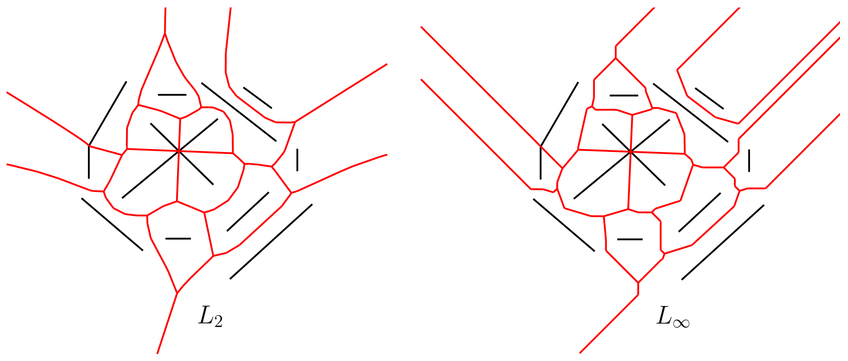


Figure 1.5. Examples of Voronoi diagram (in red) of five points p_1, p_2, p_3, p_4, p_5 in the L_∞ metric: (a) nearest neighbor, (b) second order, and (c) farthest.



$d_2(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ $d_\infty(p,q) = \max(|p_x - q_x|, |p_y - q_y|)$

Figure 1.6. Examples of Voronoi diagram of line-segments: (a) L_2, (b) L_∞

This dissertation has the following three directions:

1. Algorithmic and combinatorial properties of the farthest line-segment Voronoi diagram in the L_p metric, $1 \le p \le \infty$.

2. Implementation issues of the L_∞ Voronoi diagram of line-segments in CGAL (Computational Geometry Algorithms Library).

3. Applications of the L_∞ Voronoi diagrams of line-segments in VLSI pattern analysis. We use the software in item 2 to address the pritability issues of VLSI patterns.

1.1 Motivation for working in the L_∞ metric

A major portion of this dissertation involves the segment Voronoi diagram in the L_∞ metric. The L_∞ metric has several advantages, especially when the input contains many axis-parallel line-segments. Advantages are as follows [69]:

1. The L_∞ Voronoi diagram of line-segments consists solely of *straight line-segments*, whereas the L_2 diagram can also have *parabolic arcs* (see Figure 1.6).

2. If the coordinates of the endpoints of the input line-segments (sites) are rational, then the coordinates of vertices of the L_∞ diagram are also rational. In contrast, in the L_2 diagram, the coordinates of vertices can be algebraic numbers of higher degree and square roots could be required

to denote exactly the coordinates of vertices in the L_2 diagram, even with rational input.

3. The *degree* of an algorithm [52] is a complexity measure capturing its potential for robust implementation. An algorithm has degree d if its test computations involve the evaluation of multivariate polynomials of arithmetic degree at most d. The degree captures the precision to which arithmetic calculations need to be executed, for a robust implementation of the algorithm. Therefore algorithms of low degree are desirable. A crucial predicate for a Voronoi algorithm is the *in-circle* test, which checks whether a new input site is altering or erasing an existing vertex of the diagram. The L_2 in-circle test for arbitrary line-segments can be implemented with degree 40 [13, 14], whereas the corresponding L_∞ test only with degree 5 [69].

4. The *straight skeleton* [4] of a simple polygon is defined by shrinking the polygon by translating each of its edges at a fixed rate, keeping sharp corners at the reflex vertices. Tracing the vertices during the shrinking process gives the straight skeleton. All its edges are straight line including in the L_2 metric. It captures the shape of objects in a natural manner. A straight skeleton is more difficult to compute than a Voronoi diagram. Straight skeletons do not provide proximity information and therefore cannot be used in place of Voronoi diagrams. The straight skeleton and the Voronoi diagram under the L_∞ metric coincide when the input consists of axis-parallel line-segments.

The L_∞ distance is very well suited for applications in VLSI CAD, where the shapes are predominantly rectilinear. This is because, for rectilinear shapes the degree of in-circle predicates is 1 (that is no numerical precision issues). Moreover, the straight skeleton coincides with the Voronoi diagram of polygons having axis parallel edges. Thus the L_∞ Voronoi diagram of rectilinear polygons gives both proximity as well as shrinking and expansion information. For properties of bisectors, construction algorithms, and applications in VLSI CAD of Voronoi diagrams in the L_∞ metric [62, 63, 64, 68, 69, 88, 92].

1.2 Implementation of computational geometry algorithms in CGAL

For the last three decades there has been substantial development of efficient geometric methods and data structures. But the implementation of the simplest geometric algorithm can be a difficult task because of precision problems (replacement of theoretical model of exact arithmetic by imprecise built-in floating-point arithmetic), degenerate cases, and requirements for advanced data structures. Also advanced algorithms are often hard to understand and hard to code by average programmers. For these reasons, there is need for software libraries that provide correct and efficient implementations of geometric algorithms. CGAL [28, 29] is one such project. This project started in 1996 as a joint effort of several research groups working on computational geometry, and was partially funded by European and National Research Agencies. CGAL's current release 4.6 consists of over 600K lines of code, organized in 100 packages, and documented in nearly 4000 pages user and reference manual. The goal of the CGAL Open Source Project is to provide easy access to efficient and reliable geometric algorithms in the form of a C++ library.

In this dissertation we describe an implementation of the nearest neighbor line-segment Voronoi diagram in the L_∞ metric in CGAL (see Chapter 4). We focus on the implementation of the Voronoi diagram in the L_∞ metric targeting applications in VLSI CAD. An implementation of the L_∞ line-segment Voronoi diagram is desirable (see section 1.1), but, as far as we know, there was none freely available. The nearest neighbor Voronoi diagram of line-segments is a well studied geometric structure, but its efficient implementation is not straight forward due to precision problems and degenerate cases. Instead of building such an algorithm from scratch, we decided to develop it in the CGAL framework, on top of the existing L_2 line-segment Voronoi diagram of CGAL [44]. The existing CGAL implementation is the first generic and efficient implementation of the line-segment Voronoi diagram based on the exact computation paradigm, done by Karavelas [44]. The implementation is based on a randomized incremental construction algorithm. The expected running time of the algorithm is $O((n+m)\log^2 n)$, where n is the size of the input set and $m = O(n^2)$ is the number of points of intersection of the (open) segments in the input set. The implementation actually computes the Delaunay graph and also provides the drawing functions to draw the Voronoi diagram.

1.3 Applications of Voronoi diagrams in VLSI CAD

Voronoi diagrams have practical importance in various areas. In VLSI CAD Voronoi diagrams can model a variety of problems related to proximity of VLSI shapes. A concrete example addressed by generalized Voronoi diagram is computing the *critical area* to predict the *yield* of a VLSI chip [68, 69].

The critical area is a measure that reflects the probability of defects in a VLSI layout during the manufacturing process. The defects can arise due to three major failure mechanisms of VLSI circuits, namely, *shorts* (two design shapes come closer violating a design rule), *breaks*(open) and *via blocks*(open, a continuous shape is broken resulting in incomplete printing).

There had been different methods to estimate the critical area, such as *Grid based approach*, *Monte Carlo approach* and others mentioned in [69], but all these approaches have approximation in their estimation and are slow. Papadopoulou and Lee gave deterministic and fast estimation of critical area for shorts using generalizations of Voronoi diagrams [69]. For example, the critical area of shorts can be computed accurately assuming square defects in $O(n \log n)$ time [69] using 2-order L_∞ Voronoi diagram of layout polygons. Papadopoulou also described critical area extraction for opens and via blocks in a VLSI layout [62, 63, 64]. The Voronoi based methods can compute the critical area integral accurately and fast compared to other methods.

In the application part of this dissertation we focus on analyzing the patterns of VLSI layouts to find probable locations of faults using the line-segment Voronoi diagram. We came to know about the pattern analysis problem through personal communication with Dr. Maria Gabrani of IBM Research Zurich. Pattern analysis in VLSI layouts is an important area of research. Currently the tools in this area faces three main challenges [1, 83, 85]: (1) Severe timing challenges, as they may take from hours to days to analyze a single layout, (2) Reliability, that is to find all the fault locations in the given layout, and (3) A fully automated tool. We describe all the related terminology in this domain and our solution based on Voronoi diagram of line-segments in Chapter 5.

1.4 Contributions

In this section we summarize the contributions of this dissertation. This dissertation focuses on three directions: First, we study the farthest Voronoi diagram in the general L_p metric, $1 \leq p \leq \infty$; In particular, we study algorithms and combinatorial properties for this diagram. Second, is the implementation of the nearest neighbor Voronoi diagram of line-segments in the L_∞ metric in CGAL. Third, is the application of the line-segment Voronoi diagram addressing some problems in VLSI pattern analysis. Following is a summary of contributions of this dissertation:

1. Algorithmic and combinatorial issues of the farthest Voronoi diagrams of line-segments:

    - We introduce the farthest line-segment hull and its Gaussian map for the general L_p metric, $1 \leq p \leq \infty$. The farthest line-segment hull is a closed polygonal curve that characterizes the faces of the farthest line-segment Voronoi diagram similarly to the way an ordinary convex hull characterizes the regions of the farthest-point Voronoi diagram.
    - We present algorithms to construct the farthest line-segment hull by adapting standard convex hull construction algorithms to compute the farthest line-segment hull. We give a simple $O(n \log n)$ divide and conquer algorithm,where n is number of input line-segments. Our approach unifies the construction algorithms of farthest Voronoi diagram for points and line-segments.
    - In the L_∞ metric, provided the farthest line-segment hull, the farthest Voronoi diagram can be constructed in $O(h)$ time, where h is the size of the farthest line-segment hull.
    - We give improved structural bounds for the farthest line-segment Voronoi diagram in the Euclidean metric. We prove that the total number of faces of the farthest line-segment Voronoi diagram of n arbitrary line-segments is at most $6n$—6 which improves the previous bound $8n + 4$, and we also showed the corresponding lower bound equal to $5n - 6$ which improves the previous bound $4n - 4$.
    - The farthest Voronoi diagram in the L_∞ metric has at most $n + 8$ faces and this is tight. For non-crossing line-segments this number is 8. A single input line-segment can have at most 5 faces, and this is tight.

2. Implementing the line-segment Voronoi diagram in the L_∞ metric in CGAL:

    - We describe the implementation issues derived by our effort to make the L_∞ Voronoi diagram of line-segments available in CGAL. In particular, we describe the difference in the predicates between the L_2 metric and the L_∞ metric used in the randomized incremental construction algorithm for the line-segment Voronoi diagram.
    - We give a case analysis of the in-circle predicate for points and line-segments in the L_∞ metric.
    - We present an implementation of the line-segment Voronoi diagram in the L_∞ metric based on the randomized incremental construction of the line-segment Voronoi diagram in the Euclidean metric.
    - Approximately $17K$ lines of code is written for the implementation. The code is accepted and integrated to the CGAL library, and will be a part of the upcoming version CGAL 4.7. The new version CGAL 4.7 will be released in september 2015.

3. Application of the line-segment Voronoi diagram in VLSI pattern analysis:

    - We present our work in VLSI pattern analysis to detect problematic locations in VLSI layouts using the line-segment Voronoi diagram. The data for the experiments is provided by IBM Research, Zurich.
        - We show our experimental results on small size patterns (10 VLSI shapes). Our method predicts better fault locations in the patterns than the existing internal methods of IBM.
        - We define five different types of locations in the Layout based on neighborhood information provided by the Voronoi diagram of the design shapes. We also give a scoring method to prioritize these locations. The score of a location indicate the probability of fault at that location.
        - We show our results on bigger portion of layout. We find all the problematic locations in the layout in a minute's time.

      We provide a fast and reliable automatic tool for identifying problematic patterns in a VLSI layout.
    - We also discuss the future utility of our tool. Our tool can generate a wide range of different locations in the layout. We have a scoring method for these locations. We require an improved scoring method,

to support the need of current tools in the area of VLSI pattern analysis.

1.5 Dissertation overview and list of publications

This dissertation is organized as follows:

In Chapter 2, we review the literature on Voronoi diagram of line-segments. We also discuss the implementation issues mainly related to nearest neighbor Voronoi diagram of line-segments.

In Chapter 3, we discuss our study on farthest line-segment Voronoi diagrams. We start with the simpler L_∞ metric, describe its structural and combinatorial properties. Then, we discuss our study for the general L_2 metric, which is also valid in the L_p metric, $1 < p < \infty$. We discuss the structural properties, combinatorial properties, and construction algorithms.

In Chapter 4, we describe our work on the implementation of the line-segment Voronoi diagram in CGAL. We first give the overview of CGAL, then we review briefly the existing L_2 implementation in CGAL. Finally, we discuss interesting issues which arise during our implementation of the L_∞ line-segment Voronoi diagram.

In Chapter 5, we describe our work in the application domain. We first give the basic terminology used in the VLSI domain to address the problems of pattern analysis of VLSI layouts, and then we describe our line-segment Voronoi based solutions for the pattern analysis problem, which are finally validated by our experimental results.

We summarize the dissertation and discuss the future working directions from this dissertation in Chapter 6.
Now we list our publications associated with this thesis.

Chapter 3 is based on the following paper:

- E. Papadopoulou and S. K. Dey. On the Farthest Line-Segment Voronoi Diagram. International Journal of Computational Geometry Applications 23(6): pages 443-460, 2013.

This publication is based on the following conference papers:

- E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. International Symposium on Algorithms and Computation, pages 187-196, 2012.

  First appeared as a short abstract:
  E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. European Workshop on Computational Geometry, pages 237-240, 2012.
- S. K. Dey and E. Papadopoulou. The $L_\infty(L_1)$ farthest line-segment Voronoi diagram. In 9th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD), pages 49-55. IEEE, 2012.

Chapter 4 is based on our code submitted to CGAL. We also published one paper listed below describing our implementation:

- P. Cheilaris, S. K. Dey, E. Papadopoulou: CGAL package is accepted and is in the final stages of integration with the latest version CGAL 4.7, user and reference manual can be found at: `http://compgeom.inf.usi.ch/doc_output/Segment_Delaunay_graph_Linf_2/`

- P. Cheilaris, S. K. Dey, M. Gabrani, E. Papadopoulou: Implementing the L_∞ Segment Voronoi Diagram in CGAL and Applying in VLSI Pattern Analysis. International Congress on Mathematical Software (ICMS), pages198-205, 2014.(*also used in Chapter 5*)

- S. K. Dey, P. Cheilaris, and E. Papadopoulou. Implementing the L_∞ segment Voronoi diagram in CGAL. USI Technical Report Series in Informatics, 2015.

Chapter 5 is based on the following paper:

- S. K. Dey, P. Cheilaris, N. Casati, M. Gabrani, E. Papadopoulou. Topology and context-based pattern extraction using line-segment Voronoi diagram, SPIE Advanced Lithography, Design-Process-Technology Co-optimization for Manufacturability IX, 2015.

  This work received Luigi Franco Cerrina Memorial best student paper award at SPIE 2015.

- S. K. Dey, P. Cheilaris, M. Gabrani, E. Papadopoulou. Topology and context-based pattern extraction using line-segment Voronoi diagram, Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3), 2015 *(to be submitted)*.
  This paper is in the process of being expanded with more experiments.

Chapter 2

Related work

"Study the past if you would define the future."

Confucius

Mathematicians Voronoi and Dirichlet were the first to formally introduce the concept of Voronoi diagrams [5, 7]. The concept became popular and found its usage in many application domains after Shamos and Hoey presented their work on *Closest-point problems* in [76]. For detailed information on algorithms and properties of different variants of Voronoi diagrams we refer to the surveys on Voronoi diagrams in [5, 7, 61].

In this Chapter, we discuss algorithmic and combinatorial results on Voronoi diagrams: in particular, nearest neighbor Voronoi diagrams in Section 2.1, higher order Voronoi diagrams in Section 2.2, farthest site Voronoi diagrams in Section 2.3, and the Voronoi diagrams in the max-norm in Section 2.4. We also discuss the implementation issues for Voronoi diagrams in Section 2.5.

2.1 Nearest neighbor Voronoi diagram

The nearest neighbor Voronoi diagram for a given set S of n simple sites in the plane, is a partitioning of the plane into regions (see Equation 1.1), such that each point within the Voronoi region of site $s \in S$, is closer to s than any other site in S (see Figure 1.1 (a) for an example of Voronoi diagram of points). For n point sites in the plane, the structural complexity of the nearest neighbor Voronoi diagram is linear in the number of sites, that is, it has $O(n)$ number of vertices, edges, and faces. The interior of the circle centered on a Voronoi edge of two points and passing through them is always empty. Also, the interior of the circle centered at a Voronoi vertex and passing through the three point sites associated with the Voronoi vertex is always empty. These empty circle properties are important as they are used in the construction algorithm of the nearest neighbor Voronoi diagram.

There are several algorithms to construct a Voronoi diagram, such as incremental approaches, divide and conquer and plane sweep. Construction of Voronoi diagrams by incremental insertion is a very natural idea. Green and Sibson [35] first gave such an algorithm that takes $O(n^2)$ time. The incremental insertion process was described using the dual structure, the Delaunay triangulation. The advantage over a direct construction of the Voronoi diagram is that there is no need to construct and store the Voronoi vertices that appear in the intermediate diagrams and not in the final diagram. The randomized version of the incremental insertion approach takes $O(n \log n)$ expected time [5, 36]. The main idea of the incremental approach is to locate a face in the existing Voronoi diagram during the insertion of a new site, such that the new site lies partially or completely within that face, and then update the diagram locally. During the insertion of a new site, we may need to do local updates due to *conflicts*[1] [45].

The divide and conquer approach to compute the Voronoi diagram was presented by Shamos and Hoey [76]. It is a deterministic worst case optimal algorithm for Voronoi diagram construction that takes $O(n \log n)$ time and $O(n)$ space.

The plane sweep approach also provides a way of computing the Voronoi diagram of n points in the plane in $O(n \log n)$ time and $O(n)$ space. The main idea is to sweep over the plane by an axis parallel line called *sweep line*. During the sweep process, the information about the Voronoi diagram computed so far does not change as the sweep line moves forward, except at certain special points called *events*. The difficulty is to discover new Voronoi vertices in time. This is because by the time the sweep line reaches a new point site, it has been already intersecting the Voronoi edges of this new point site for quite some time. To handle this difficulty, the sweep line itself is considered as another additional site, and the Voronoi diagram of all the points to the left of the sweep line and

[1]Let s_n be a new site to be inserted in the existing Voronoi diagram. Also let p be a Voronoi vertex or a point on a Voronoi edge of the current diagram. Then p is said to be in conflict with s_n, if the Voronoi disk (a disk tangent to the associated sites) centered at p includes or intersects s_n. If p is a Voronoi vertex, we call it a *vertex conflict*, otherwise an *edge conflict*.

the sweep line itself are maintained. The boundary of the bisector of the sweep line and the points to its left are called *wavefront*. The sweep line status, which is the wavefront, can be maintained in a balanced binary tree, so that any operation on it will take $O(\log n)$ time, where n is the number of input point sites. The events are maintained in a priority search tree, so that operations on it will also take $O(\log n)$ time in the worst case. Thus, the sweep line algorithm in total takes $O(n \log n)$ time to construct the Voronoi diagram of n points. Fortune [30] was the first to give a plane sweep approach to compute Voronoi diagrams efficiently.

The nearest neighbor Voronoi diagram of line-segments is also well studied (see example in Figure 2.1 (a), shaded region is $V(s_5)$). Most of the earlier algorithms assumed that the line-segments are either disjoint or they are allowed to intersect only at endpoints. Drysdale and Lee [51] presented an algorithm to compute the nearest neighbor segment Voronoi diagram in $O(n \log^2 n)$ time, where n is the number of input segments. Kirkpatrick [46], Lee [50], and Yap [93] showed divide and conquer worse case optimal $O(n \log n)$ algorithms for this problem. Fortune [30] described a worst case optimal algorithm, using the sweep-line paradigm. Boissonnat et al. [11] and Klein et al. [48] presented $O(n \log n)$ randomized incremental algorithms for computing the Voronoi diagram of line-segments. Karavelas [44] gave a robust and efficient implementation of a randomized incremental algorithm for constructing the Voronoi diagram of line-segments.

2.2 Higher order Voronoi diagram

The order-k Voronoi diagram, $1 \leq k \leq n-1$, of a set S of n sites is a partitioning of the plane into regions (see Equation 1.2), such that each point within an order-k Voronoi region has the same set of k nearest sites.

For point sites in the plane, the order-k Voronoi diagram has been studied extensively [5, 18, 50]. Its structural complexity has been shown to be $O(k(n-k))$ [50]. The first algorithm to compute the order-k Voronoi diagram for points was given by Lee [50]. Lee's iterative algorithm for computing the higher order Voronoi diagram for points in the Euclidean metric, which computes the diagram iteratively from order 1 to order $n-1$, takes $O(k^2 n \log n)$ time. A sweepline algorithm was given by Rosenberger [75], that also takes $O(k^2 n \log n)$ time. Edelsbrunner et al [18] presented the first algorithm that constructs the k-order diagram without constructing the lower order diagrams in $O(n^2 \log n + k(n-k)\log^2 n)$ or $O(n^2 + k(n-k)\log^2 n)$ time depending on the data structures used. There are also many randomized algorithms available, an output sensitive randomized algorithm given by Mulmuley [58] that takes expected $O(k^2 n \log n)$ time, an on-line randomized incremental algorithm of Aurenhammer et al. [8] that takes expected $O(k^2 n \log n + nk \log^3 n)$, a randomized divide and conquer by Clarkson [22] that takes expected $O(kn^{1+\epsilon}), \epsilon > 0$ time and many others [2, 17, 74].

Recently, Papadopoulou and Zavershynskyi [72] have analyzed the structural properties of the order-k Voronoi diagram of line-segments (see example in Figure 2.1 (b) for order-2 diagram, where segment pair (s_1, s_2) have two disconnected regions shown in grey shade). It is shown that a single order-k Voronoi region may disconnect to $\Omega(n)$ disconnected faces in the worst case. Despite disconnected regions, the overall structural complexity of the Voronoi diagram of n disjoint line-segments remains $O(k(n-k))$.

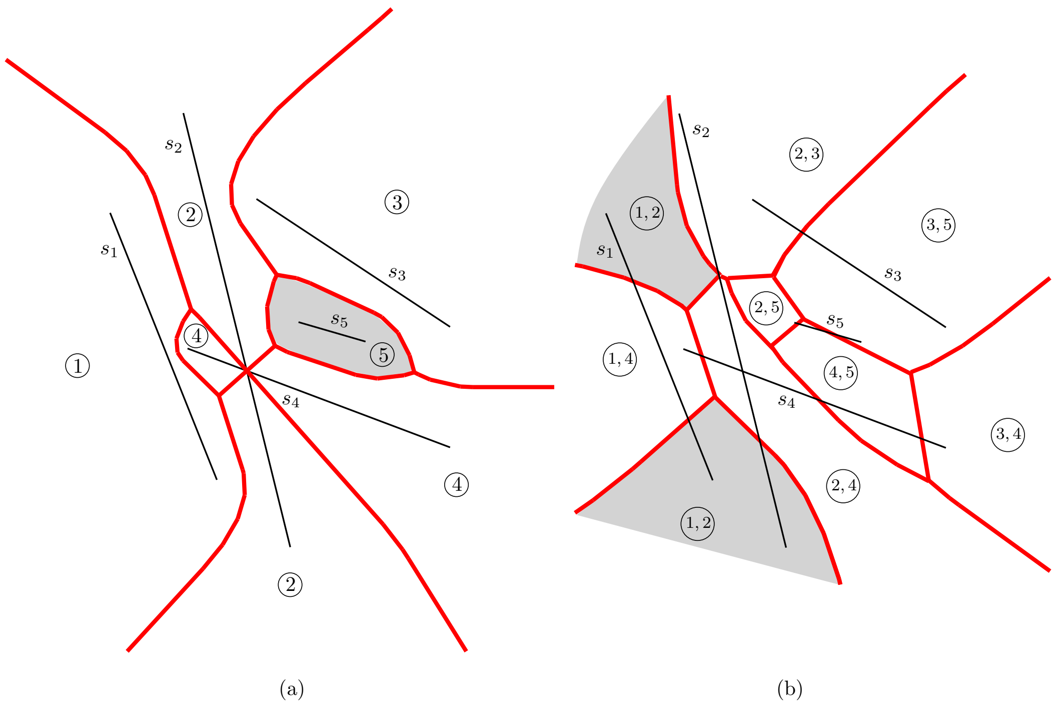


Figure 2.1. Examples of line-segment Voronoi diagrams (shown in red) in the Euclidean plane: (a) nearest neighbor, (b) 2-order.

The order-k Voronoi diagram of line-segments can be constructed in $O(k^2 n \log n)$ time by adapting any iterative approach to compute higher order Voronoi diagrams [72]. Papadopoulou and Zavershynskyi [73] has presented sweepline algortihm to compute the higher order Voronoi diagram of line-segments in $O(k^2 n \log n)$ time.

Recently, Zavershynskyi et al [10] presented a randomized iterative algorithm for the construction of the higher order Voronoi diagram of line-segments in expected $O(k^2 n \log n)$ time.

2.3 Farthest site Voronoi diagram

The farthest site Voronoi diagram of S is a subdivision of the plane into regions (see Equations 1.3 and 1.4) such that the region of a site $s \in S$, is the locus of points farther away from s than from any other site.

For simple sites, such as points and line-segments, the Voronoi regions are unbounded and the Voronoi diagram is a tree structure with linear structural complexity on the size of the input sites. In the farthest site Voronoi diagram, a circle passing through a site s with its center at any of the associated Voronoi vertices of s will enclose all the other input sites.

The farthest point Voronoi diagram can be computed using several algorithms given in $O(n \log n)$ time [5, 49, 76]. One of the popular algorithm is divide and conquer. In the divide and conquer approach, first the convex hull of points is computed, since only the points on the convex hull contribute the farthest Voronoi regions. Then, the convex hull is partitioned into two chains (e.g left and right chains considering the dividing vertical line passing through the point with median x coordinate of the input points, or upper and lower chains divided by a horizontal line passing through the point with median y coordinate of the input points). There will be $O(\log n)$ partition steps. And in each partition step, the merging can be done in linear time following Kirkpatrick's algorithm [46].

A randomized construction is provided by Chew [21, 23] which computes the diagram in expected linear time. The farthest point Voronoi diagram for points on a convex hull can be computed by a deterministic linear time algorithm provided by Aggarwal et al [3].

The farthest Voronoi diagram for line-segments had not received attention in the literature for a long time. The farthest line segment Voronoi diagram was only recently considered in [6]. Aurenhammer et al [6] described the structural and combinatorial properties. The farthest Voronoi diagram of segments shows surprisingly different properties from both the farthest Voronoi diagram of points and the nearest neighbor Voronoi diagram of segments. For an example, in the farthest Voronoi diagram of line-segments the regions are not characterized by convex hull properties as in the case of points [6]. Furthermore, the Voronoi regions in the farthest line segment Voronoi diagram may be disconnected as opposed to the Voronoi regions in the farthest Voronoi diagram of points. In particular, a single line segment can have $\Theta(n)$ disconnected faces [6] (see Figure 2.2). Nevertheless, the number of edges and vertices of the farthest segment Voronoi diagram remains $O(n)$ [6] (Aurenhammer el al [6] showed the upper bound and the lower bound on the number of faces to be $8n+8$ and $4n-4$ respectively), regardless of the crossing properties of the input segments. In contrast, the nearest neighbor Voronoi diagram of segments has $O(n^2)$ vertices and edges in the worst case, where each crossing constitutes a vertex.

The construction algorithm of farthest line-segment Voronoi diagram presented in [6] takes $O(n \log n)$ time using divide and conquer approach paired with a plane sweep algorithm.

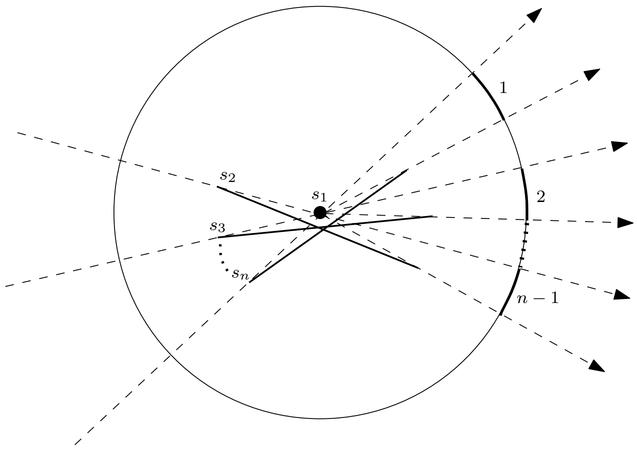


Figure 2.2. Directions indicating towards faces (there are $n-1$ disconnected faces) of s_1 [6].

The farthest-polygon Voronoi diagram, was addressed by Cheong et al [20]. Cheong et al provided a $O(n \log^3 n)$ divide-and-conquer algorithm to construct the farthest-polygon Voronoi diagram, where n is the total complexity of the disjoint polygonal sites.

Abstract Voronoi diagram is a high-level framework for Voronoi diagrams suggested by Klein [47]. The definition of Voronoi diagram in the abstract framework is independent of the concrete geometry, metric space or shapes of the sites. In the abstract framework the Voronoi diagrams are defined using bisecting curves. An abstract framework on the farthest-site Voronoi diagram (which does not include the case of intersecting line-segments) was given by Mehlhorn et al [55]. Mehlhorn et al [55] provided a randomized algorithm to construct the farthest site Voronoi diagram in expected $O(n \log n)$ time.

2.4 L_∞ Voronoi diagrams

We have already discussed in the Introduction, that Voronoi diagrams in the piecewise linear L_∞ metric are computationally simpler than their Euclidean counterparts, due to the lower degree of geometric predicates in the L_∞ metric. A feature that needs to be handled carefully in L_∞ Voronoi diagrams is the structure of the bisector. In case of two points along the same horizontal or vertical line, the bisector consists of a line segment and two unbounded regions (bisectors are two dimensional). Such unbounded regions can be assigned entirely to one of the points or it can be split among the points. The splitting of equidistant regions should be handled consistently (e.g. splitting can be done by the angular bisector) to avoid any discrepancy in the construction algorithms of L_∞ Voronoi diagrams. A detailed study on the bisectors in L_∞ can be found in [69].

The Voronoi diagrams of points in the L_∞ metric can be computed by any of the existing algorithms for the Euclidean plane (see Section 2.1). The higher order Voronoi diagrams of points in the L_∞ metric is also well studied. The structural complexity of the k-order Voronoi diagram in the L_∞ metric is $O(min\{k(n-k), (n-k)^2\})$ [53]. Liu et al [53] gave an output sensitive algorithm to compute the L_∞ k-order Voronoi diagram in $O((n+m)\log n)$ time, where m is the structural complexity of the k-order Voronoi diagram.

Papadopoulou et al[69] adapted the classical plane sweep algorithm to construct Voronoi diagram of line-segments in L_∞ metric in $O(n \log n)$ time. Recall that the degree of an algorithm [52] is d if its test computations involve the evaluation of multivariate polynomials of arithmetic degree at most d. The plane sweep algorithm to construct the Voronoi diagram of line-segments in the L_∞ metric has degree 7 [69]. A crucial predicate for a Voronoi algorithm is the in-circle test (see Section 2.5), which checks whether a new input site is in conflict with existing vertex of the diagram. The in-circle test in the L_∞ metric for line-segments has degree 5 [69].

The Voronoi region of a line-segment in the nearest neighbor Voronoi diagram can be subdivided in to finer regions by the Voronoi diagram of its neighbor, and thus the 2-order Voronoi diagram can be computed. Papadopoulou et al[64, 69] showed applications of the L_∞ 2-order Voronoi diagram in VLSI CAD (see Section 1.3).

Papadopoulou introduced the Hausdorff[1] Voronoi diagram of rectangles in the L_∞ metric in [62]. The Hausdorff Voronoi diagram of n non-crossing rectangles can be computed by a plane sweep algorithm [62] in $O((n+K)\log n)$ time, where K is the number of interacting rectangles (as defined in [62]) during the plane sweep. The Hausdorff Voronoi diagram of line-segments finds applications in VLSI design automation [62, 63, 71].

2.5 Implementation of Voronoi diagrams

The initial algorithms for the construction of nearest neighbor Voronoi diagrams were written often assuming a general position of the input point sites, that is, no three points are collinear and no more than three points are co-circular. Another assumption was the ability to compute exactly the basic geometric predicates like the in-circle test. These two assumptions lead to two different problems at the implementation. One is the problem of special or degenerate cases which is an algorithmic problem, and the other is the problem of numerical precision of floating point arithmetic.

[1]The (directed) Hausdorff distance from a set A to a set B is $h(A,B) = max_{a\in A} min_{b\in B}\{d(a,b)\}$. The (undirected) Hausdorff distance between A and B is $d_h(A,B) = max\{h(A,B), h(B,A)\}$.

In this Section we give some basic concepts that are important for the robust and efficient implementation of Voronoi diagrams, and will mention some available software for computing Voronoi diagrams of points and line-segments in the Euclidean metric.

2.5.1 Degree of in-circle test

The basic predicate in any algorithm for computing Voronoi diagrams is the *in-circle test*. Let V_{pqr} be the Voronoi vertex of three sites p, q, and r through which a circle C_{pqr} passes with no sites in its interior. Then, the in-circle test is defined as a test to determine whether a fourth site is inside (or intersects), outside (no intersection), or on the circumference of C_{pqr}.

The robustness of the implementation of any algorithm to compute a Voronoi diagram depends at least on the degree of the in-circle test. The in-circle test can be described in the following form of a determinant whose sign determines the position of a point (x, y) with respect to the circle passing through the three points (x_1, y_1), (x_2, y_2), and (x_3, y_3).

$$\begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = \begin{cases} \text{equals } 0, & \text{on circle} \\ \text{less than } 0, & \text{outside circle} \\ \text{greater than } 0, & \text{inside circle} \end{cases}$$

The degree of the above computation is 4, that is for point sites the in-circle predicate in the Euclidean metric can be answered with degree 4.

In the Euclidean metric, the in-circle test for the Voronoi diagram of line-segments, can be answered correctly with degree 40 [13]. Karavelas and Kamarianakis et al.[39, 40] gave a detailed case analysis of in-circle predicates for line-segments with improved degree in cases for axis-aligned line-segments. The in-circle test for line-segments in the L_∞ metric, can be answered with degree 5 [69], and for the axis parallel line-segments, can be answered with degree 1. See section 4.3.6 for detailed analysis of degree of the in-circle test for points and line-segments in the L_∞ metric.

2.5.2 Robust geometric algorithms

A geometric software using floating point arithmetic can often fall in to traps of infinite loops, or give incorrect outputs. Robustness means the software should be able to carry out computations exactly or with in the specified error bounds and not fall in infinite loops or crash due to degenerate cases. The exact computation [94] of geometric predicates is important for any implementation to have a guaranteed correct output. The implementation of algorithms in most fields may live with numerical approaches like rounding off, or if rounding off is accumulating undesirable error, then improving the results using standard numerical precision techniques. In contrast, the numerical precision techniques may not work for implementation of algorithms in computational geometry. This is because the queries of geometric algorithm requires exact answer, for an example, the in-circle predicate for points wants to know exactly if the query point is inside, outside, or on the circle, and the answer is exactly one of the three choices, an approximate answer in this case may just give a wrong output. The problems of numerical stability for Voronoi diagrams are addressed in [31, 81].

Exact computation with arbitrary precision is time expensive and thus can be very slow. For efficiency, many software either choose finite precision with guaranteed error bounds and give approximate results, or filtering techniques which allow the tools to use computations with arbitrary precision less often and give exact results.

The exactness of predicates in the implementations can be achieved using exact multiple precision integer, rational and floating point numbers. GMP [34], MPFR [32], and LEDA [56] are widely used libraries to achieve multiple precision and hence the exact computation of a given design.

Another approach for having numerically robust geometric algorithms is the *topology-oriented approach* [81]. This approach primarily describes the basic part of the algorithm in terms of topological and combinatorial computation. The combinatorial and topological computation is never contaminated with numerical errors and thus guarantees robustness of the algorithm. This approach often gives non deterministic output. A combination of numerical computation with this approach may lead to the exact output. However, this approach does not guarantee a deterministic exact output.

2.5.3 Software for computing Voronoi diagrams

There are several software available to compute Voronoi diagrams for simple sites such as points, line-segments, curves, polygons. We list some of them with their features.

Matlab [54]
Matlab is a commercial software with many mathematical computation capabilities. It can compute the Voronoi diagram of points. The implementation is based on *Qhull* (quick hull [9] - a standard method to compute the convex hull). It handles round-off errors from floating point arithmetic. It does not use the exact computation paradigm.

LEDA [56]
LEDA is a software that provides various algorithms and data structures. LEDA use the exact computation paradigm for accurate results. It provides nearest and farthest Voronoi diagram of points.

BGL [78]
Boost Graph Library (BGL) is an open source C++ library that has a package named *Boost.Polygon*, which provides Voronoi diagrams of points and line-segments in the Euclidean metric. There are limitations for the type of input. The input points and end points of input line-segments must be of integer type. The input line-segments can only meet at end points. Boost uses C++ STL data structures to deal with memory management issues, and uses multiprecision computation to avoid numerical instability.

OpenVoronoi [89]
OpenVoronoi in an open source software written in C++ with Python bindings. The software computes the Voronoi diagram of points and line-segments using incremental topology-oriented design [37, 38, 82]. OpenVoronoi is currently trying to incorporate Voronoi diagram for circular-arcs.

VRONI [37]
VRONI is a commercial software widely used in industry. The implementation of VRONI is based on the topology-oriented approach introduced by Sugihara et al [82], for the computation of the Voronoi diagram of points and line-segments in the Euclidean metric. Later, VRONI was upgraded to compute the Voronoi diagram of circular-arc as well, and was called *ArcVroni* [38].

CGAL [84]
CGAL provides Voronoi diagram of points and line-segments in the Euclidean metric through a randomized incremental design, developed by Karavelas [41, 42, 44]. The package is carefully written to work with exact arithmetic. The use of exact arithmetic is time expensive thus, it can be very slow. CGAL uses filtering techniques (more discussions in Chapter 4) to make the implementation efficient. CGAL also provides the Voronoi diagram of disks, the higher order Voronoi diagram of points, Voronoi diagram of spheres and some other variants as well. We will discuss more on CGAL basics and the Voronoi package for points and line-segments in Chapter 4.

Chapter 3

Farthest line-segment Voronoi diagram

"Work without love is slavery."

Mother Teresa

In this chapter, we discuss our study on the farthest Voronoi diagram of line segments. The content of this chapter is taken from [67]:

E. Papadopoulou and S. K. Dey. On the Farthest Line-Segment Voronoi Diagram. International Journal of Computational Geometry Applications (IJCGA) 23(6): pages 443-460, 2013.

The farthest-site Voronoi diagram of a set S of n line segments is a subdivision of the plane into regions such that the region of a line-segment s is the locus of points farther away from s than from any other line-segment in S. We define the farthest Voronoi region of a line-segment s_i as:

$$\textit{freg(s)} = \{x \in \mathbb{R}^2 \mid \forall t \in S \setminus \{s\},\ d(x,s) \geq d(x,t)\}, \tag{3.1}$$

where $d(x,s)$ denotes the distance between a point x and site s under a given metric. The partitioning of the plane derived by the union of all such regions formed by the sites in S, together with their bounding edges and vertices, defines the farthest line-segment Voronoi diagram of S.

We study the combinatorial properties of the farthest line-segment Voronoi diagram in the L_p metric, $1 \leq p \leq \infty$. We introduce the *farthest line-segment hull* and its *Gaussian map*. The farthest line-segment hull is a closed polygonal curve that characterizes the faces of the farthest line-segment Voronoi diagram similarly to the way an ordinary convex hull characterizes the regions of the farthest-point Voronoi diagram. The Gaussian map reflects the cyclic order of the farthest line segment hull (detailed definition in Section 3.2). We study construction algorithms for the farthest line-segment hull. In particular, we adapt the standard convex hull algorithms to construct the farthest line-segment hull. We also give improved combinatorial bounds on the number of faces of the farthest line-segment Voronoi diagram.

The farthest line-segment Voronoi diagram finds applications in computing the smallest disk that overlaps all line-segments in a given set. For example, line-segments may represent wires in a VLSI layer or in a different type of network, while disks represent defects. In the VLSI *critical area extraction* problem [64], random manufacturing defects cause *open faults* when they overlap entire sets of wires or contacts on a layer under consideration. Thus, the farthest line-segment Voronoi diagram can be used for computing the *Probability of Fail* of a VLSI layer in the presence of random manufacturing defects. It is also necessary in defining and computing the *Hausdorff Voronoi diagram*[1] of clusters of line segments, which finds similar applications in VLSI design automation [64].

This chapter is organized as follows. We start our study with the simpler L_∞ metric. In Section 3.1 we describe our investigation on the farthest line-segment Voronoi diagram in the L_∞ metric, which appears as a ISVD conference paper [25]. In Section 3.2 we describe the farthest line-segment Voronoi diagram in the L_2 metric, which naturally extends to L_p, $(1 < p < \infty)$ without changes. The content of this work first appears as a ISAAC conference paper [65] and then as a IJCGA journal paper [67].

3.1 Farthest line-segment Voronoi diagram in L_∞

We first investigated the structural properties of the L_∞ farthest Voronoi diagram for line segments. We coded a demo program in GNUplot to realize the examples of farthest Voronoi diagrams for segments in L_∞. It was a grid based naive program but it helped us to visualize the diagram and investigate its properties. Due to limited methods and graphics capability of GNUplot we again coded in C and OPENGL a GPU demo to get clearer visuals (see Figure

[1]The (directed) Hausdorff distance from a set A to a set B is $h(A,B) = max_{a \in A} min_{b \in B}\{d(a,b)\}$. The (undirected) Hausdorff distance between A and B is $d_h(A,B) = max\{h(A,B), h(B,A)\}$.

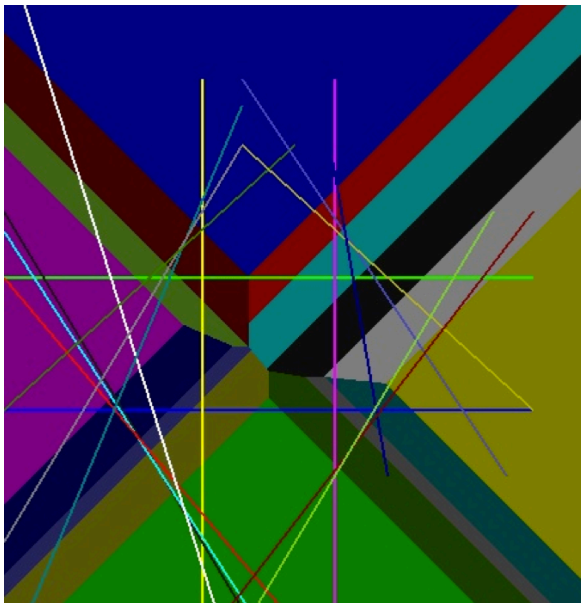


Figure 3.1. Farthest Voronoi diagram of segments in L_∞

3.1) of the diagram for greater number of input segments. This demo helped to identify various properties of the diagram.

The farthest Voronoi region of line segments in the L_∞ metric is given by Equation 3.1. $d(x,s)$ in Equation 3.1 denotes the distance between a point x and a line segment s, which is the minimum L_∞ distance between x and any point on s, that is, $d(x,s) = min\{d(x,q), \forall q \in s\}$. The collection of all such regions together with their bounding edges and vertices, defines the L_∞ farthest line-segment Voronoi diagram.

A Voronoi edge bounding regions $freg(s_i)$, $freg(s_j)$ is portion of the bisector $b(s_i,s_j)$, which is the locus of points equidistant from s_i and s_j. In L_∞, $b(s_i,s_j)$ consists of a constant number of pieces of straight lines, where each piece is portion of an elementary bisector between the endpoints and the open line-segment portions of s_i and s_j. The L_∞ bisector of two non-parallel lines $b(l_1,l_2)$ of slopes b_1, b_2 consists of two straight line branches of slopes as given in [69]. If b_1, b_2 are both positive (resp. negative) then one branch is always a line with slope -1 (resp. slope $+1$). When two line-segments or their endpoints are aligned along the same vertical or horizontal line their bisector consists of a line segment and two unbounded regions. The equidistant region can be assigned arbitrarily (or lexicographically) to one of the points and consider the region boundary only as their bisector. For more information on L_∞ line-segment bisectors, see [69]. Note that the endpoints and the open line-segment portion of an arbitrary line-segment s can be separated by ±1-slope lines of opposite sign of the slope of s.

A maximally connected subset of a region in *FVD(S)* is called a *face*. A halfplane bounded by an axis-parallel line is called an *axis-parallel halfplane*. The common intersection of two axis parallel halfplanes with perpendicular bounding lines is called a *quadrant*. The following lemma and its corollary is a simple adaptation of Lemma 1 and Corollary 2 of [6] for the L_∞ metric.

Lemma 1. *All faces of FVD(S) are unbounded in one of the eight possible directions that are implied by rays of slope* $\pm1, 0, \infty$.

Proof. Let t be a point in an arbitrary face f of *FVD(S)* belonging to the region of segment s_i. Then there is a square disk $D(t)$ centered at t, touching s_i, that is also intersecting (or touching) all segments $s_j \in S \setminus \{s_i\}$. If $D(t)$ touches s_i with a vertical (resp. horizontal) side let R be a horizontal (resp. vertical) ray starting at t directed away from s_i. If $D(t)$ touches s_i with a corner at a single point p, let R be the ±1-slope ray through the incident diagonal of $D(t)$, starting at t and directed away from p. Since for any point y along R, $D(t) \subset D(y)$, and $D(y)$ touches s_i in exactly the same way as $D(t)$, $D(y)$ must continue to intersect all other segments in S and thus, R must be entirely contained in $reg(s_i)$ and in particular f. Thus, face f must be unbounded along the direction of R. □

Corollary 1. *The interior of $reg(s_i)$ is non-empty if and only if there exists an axis-parallel halfplane or a quadrant L which touches s_i and intersects (or touches) all other segments in S. If L is an axis-parallel halfplane then $reg(s_i)$ is unbounded in the direction in L normal to the bounding line of L. If L is a quadrant touching s_i by its corner then $reg(s_i)$ is unbounded in the ±1 direction away from the corner of L.*

It is clear by Corollary 1 that FVD(S) must always contain four faces, denoted as *north, south, east* and *west*, such that each face is unbounded in one of the four axis-parallel directions respectively. Each one of these faces is induced by a halfplane bounded by an axis-parallel line as defined below.

Definition 1. *Let $l_i, i = n, s, e, w$ be four axis parallel* bounding lines *of S defined as follows: Let l_n (resp. l_s) be the horizontal line passing through the bottommost upper-endpoint (resp. the topmost lower-endpoint) of all segments in S. Let l_e*

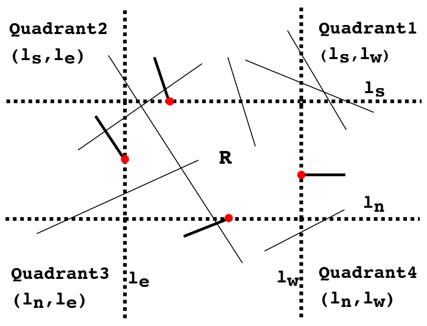


Figure 3.2. The axis parallel bounding lines

(resp. l_w) be the vertical line passing through the leftmost right-endpoint (resp. the rightmost left-endpoint) of all segments in S.

Clearly the north, south, east, and west face of *FVD(S)* is induced by line l_n, l_s, l_e, l_w respectively; the L_∞ distance within each face simplifies to the vertical or horizontal distance from the respective bounding line. Figure 3.2 illustrates the bounding lines. The bounding lines partition the plane into four quadrants, labeled $1-4$, in counterclockwise order as follows (see Figure 3.2): Quadrant 1 is formed by (l_s, l_w) and faces *north-east*, Quadrant 2 is formed by (l_s, l_e), Quadrant 3 by (l_n, l_e), and Quadrant 4 by (l_n, l_w) facing *south-east*. The closed rectangular regions induced by the four bounding lines is denoted by R.

It is clear, by Corollary 1, that no segment s that lies partially or entirely in the closed rectangular region R can have a non-empty Voronoi region in *FVD(S)*. Thus, these segments can be immediately discarded. Among the remaining segments, it is clear, by definition, that no such segment can have endpoints in the four quadrants. The quadrants are thus, either empty of the remaining segments, or there are segments that straddle them entirely.

Definition 2. *Let $E_i, i = 1, 2$, denote the upper envelope of the set of line segments straddling Quadrant i. Let $E_j, j = 3, 4$ denote the lower envelope of the segments straddling quadrant j. In case no segments straddle quadrant i, $i = 1, \ldots, 4$, let $E_i = O_i$, where O_i is the corner point (origin) of Quadrant i.*

We now define the *farthest line-segment hull* of S, a closed polygonal curve that characterizes the regions and the unbounded bisectors of *FVD(S)*. Figure 3.3 illustrates *FH(S)* for an arbitrary set of segments in its standard form.

Definition 3. *The* farthest line-segment hull *of S (in short farthest hull, denoted FH(S)) is the closed polygonal curve obtained by $E_1, l_{s'}, E_2, l_{e'}, E_3, l_{n'}, E_4, l_{w'}$, where $l_{i'}$ denotes the segment along bounding line l_i between its two incident envelopes.*

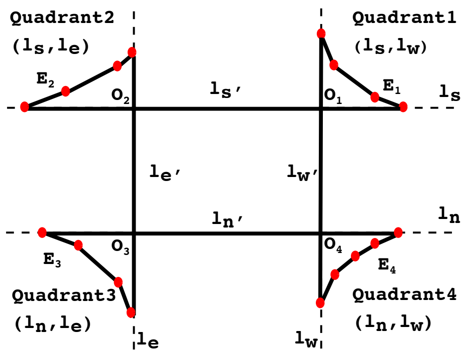


Figure 3.3. The L_∞ farthest-hull of an arbitrary set of segments

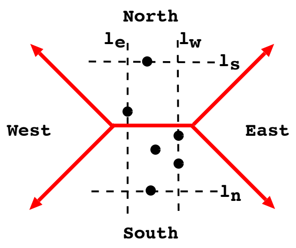


Figure 3.4. Farthest point Voronoi diagram in L_∞

For a set of points or a set of axis-parallel segments, the farthest-hull simplifies to the rectangle R of the bounding lines. In the case of points, R coincides with the minimum enclosing rectangle of the given set of points, which is well known to characterize the L_∞ farthest point Voronoi diagram (see Figure 3.4). For a set of axis parallel segments, the farthest-hull simplifies to the rectangle R induced by the bounding lines, however, the placement of the bounding lines need not always be standard, i.e., l_n may lie above l_s or l_e to the right of l_w as in Figure 3.6. Figure 3.5 illustrates similar non standard situations. The farthest hull is defined in the same way in all cases.

By Corollary 1 and the definition of the farthest hull we conclude.

Lemma 2. *A line segment s_i in S has a non-empty Voronoi region in FVD(S) if and only if it appears on the farthest hull, either explicitly as a segment on some envelope or implicitly by defining a bounding line. Voronoi faces are circularly ordered following the order of the farthest hull.*

Let e_i, e_j be two edges of the farthest hull corresponding to segments s_i, s_j. Let l_i, l_j denote the lines through s_i and s_j respectively and let $H(e_i), H(e_j)$ de-

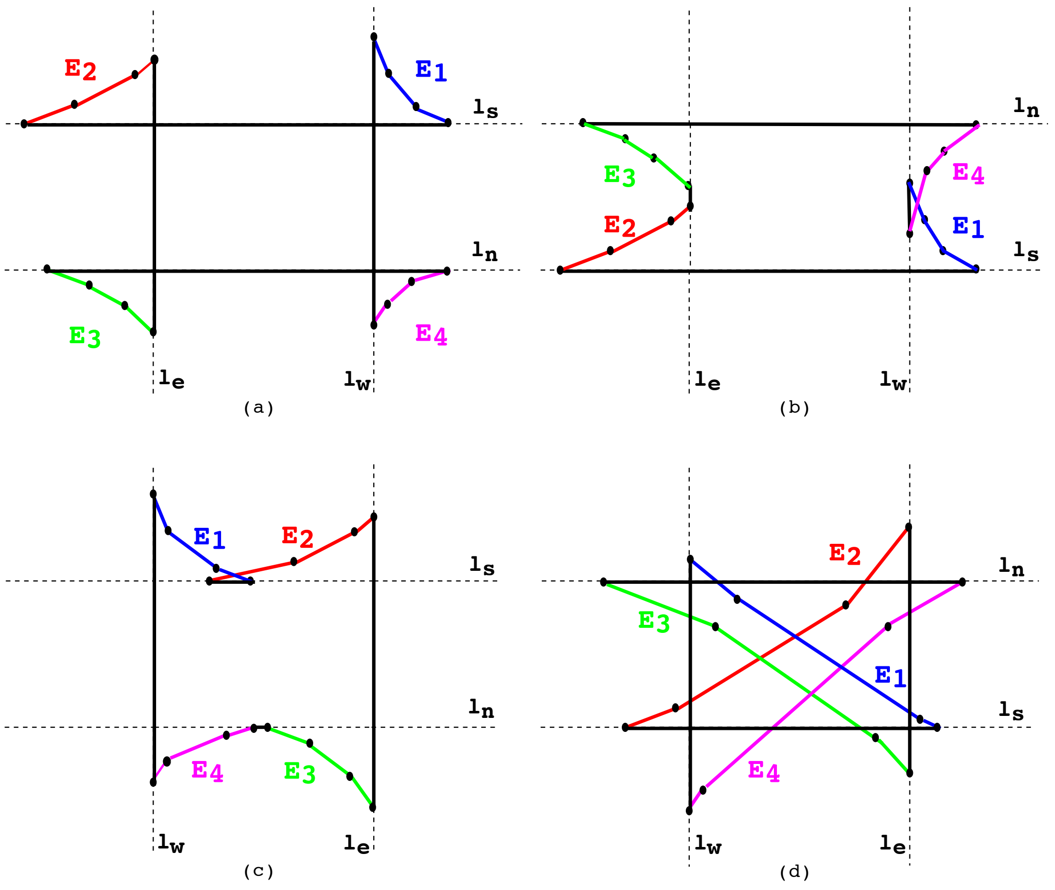


Figure 3.5. The L_∞ farthest hull for different positions of l_n, l_s, l_e and l_w. (a) Standard position (b) l_n lies above l_s (c) l_e lies to the right of l_w (d) l_n lies above l_s, as well as l_e lies to the right of l_w

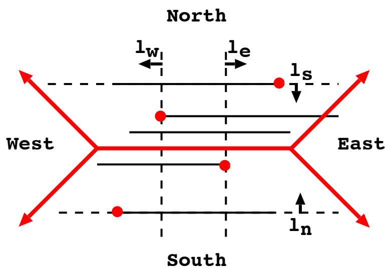


Figure 3.6. FVD(S) of axis parallel segments, where l_e is right of l_w.

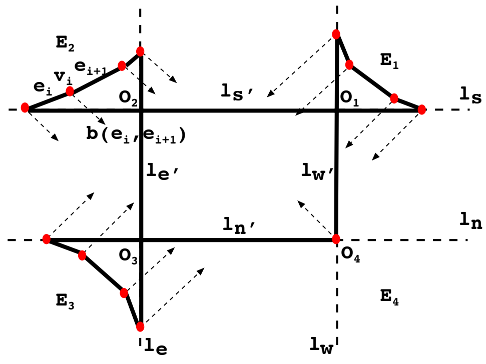


Figure 3.7. The unbounded bisectors corresponding to vertices on the farthest-hull.

note the corresponding halfplane on the side of l_i, l_j containing R. The *unbounded bisector* of e_i, e_j, denoted $b'(e_i, e_j)$, is the portion of $b(l_i, l_j)$ that is relevant to *FVD(S)*, that is, $b'(e_i, e_j)$ is the ray of $b(l_i, l_j)$ in $H(e_i) \cap H(e_j)$. In Figure 3.7, unbounded bisectors of consecutive farthest hull edges are indicated by arrows.

Lemma 3. *There is a 1-1 correspondence between the unbounded Voronoi edges of FVD(S) and the vertices of the L_∞ farthest hull. All unbounded Voronoi edges are rays of slope ± 1 partitioned into four groups, one group for each envelope of the farthest hull, each group being a set of parallel rays.*

Proof. Let $v_1, v_2, \ldots, v_h$ denote the list of vertices along *FH(S)* in, say, counterclockwise order, starting at $E_1 \cap l_w$. Each vertex v_i corresponds to an unbounded bisector $b'(e_i, e_{i+1})$, where e_i, e_{i+1} are the farthest hull edges incident to v_i (see Figure 3.7). By definition of the farthest hull, the quadrant L induced by any vertex v on the farthest hull that contains R must touch the two segments that explicitly or implicitly define v and it must intersect all other segments in S. Thus, every vertex of *FH(S)* corresponds to an unbounded Voronoi edge. Conversely, for any unbounded Voronoi edge of FVD(S), portion of bisector $b(s_i, s_j)$, there is a quadrant L that touches s_i, s_j and intersects all other segments in S. In all cases the corner point of L corresponds to a vertex of the farthest hull (by the definition of the farthest hull). □

As shown in [6] for the Euclidean case, the graph structure of *FVD(S)* is connected and corresponds to a tree. This property clearly remains valid in L_∞ / L_1 following the same arguments. By the above discussion we conclude.

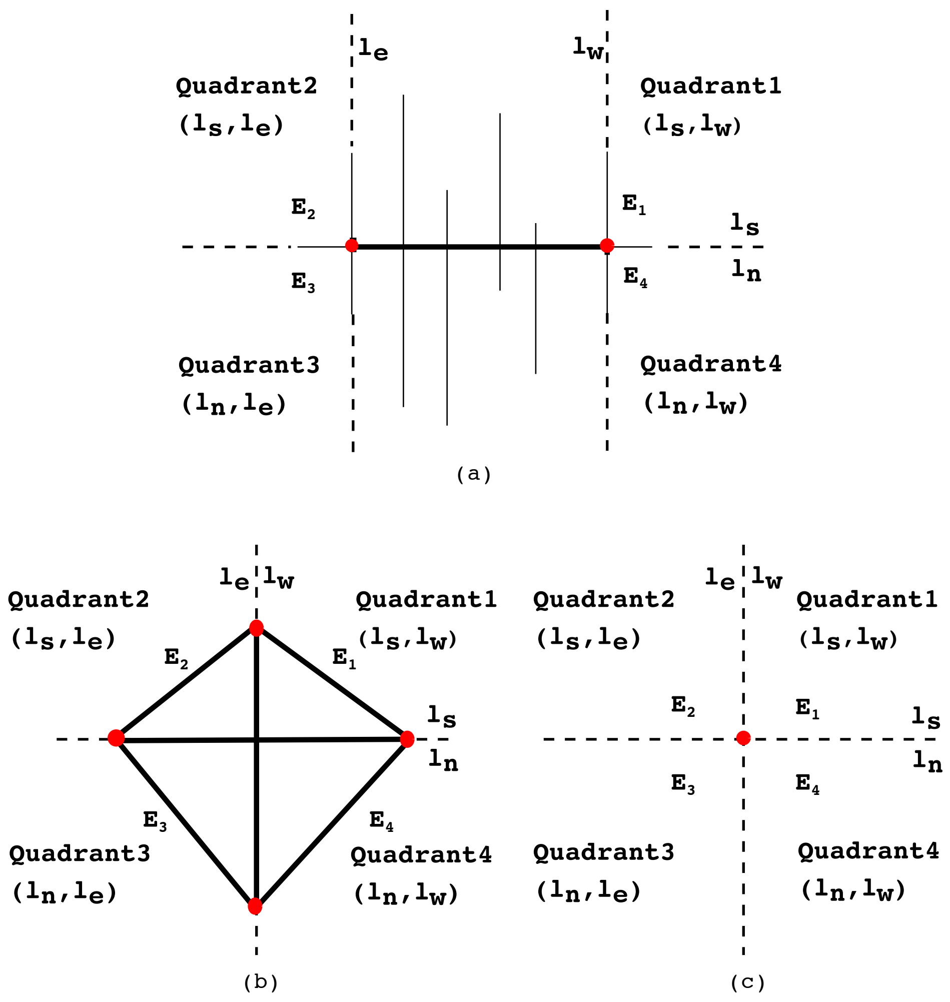


Figure 3.8. Examples showing degenerate farthest hulls. (a) l_n and l_s coincide (farthest hull degenerates to a line-segment). (b) l_n and l_s coincide as well as l_e and l_w coincide, and the quadrants are straddled. (c) l_n and l_s coincide as well as l_e and l_w coincide (farthest hull degenerates to a point)

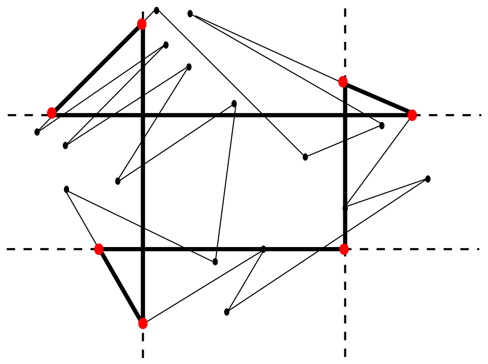


Figure 3.9. Constant size farthest-hull for a simple polygon

Theorem 1. *The L_∞ farthest segment Voronoi diagram consists of exactly one unbounded face for each edge of the farthest hull and has size $O(h)$, where h is the size of the farthest hull that can vary from $O(1)$ to $O(n)$.*

Remarks. The L_∞ *FVD(S)* always consists of four faces, facing north, south, east, and west, each induced by one bounding line, l_n, l_s, l_e, l_w respectively. In addition, for any non-trivial envelope, it contains one face for each segment of the envelope, always bounded by parallel slope-± 1 rays. A segment may contribute to at most two different envelopes and thus, it may appear twice on the farthest hull. It may also contribute to both a bounding line and its incident envelope or induce a number of the bounding lines. Degenerate instances of the *farthest-hull* may arise, when parallel bounding lines coincide (see e.g Figure 3.8). For non-crossing segments the envelopes can clearly consist of at most one segment each, thus, the L_∞ *FVD(S)* has $O(1)$ size. Figure 3.9 illustrates the farthest hull of non-crossing segments forming a simple polygon.

Corollary 2. *The region of a segment in FVD(S) may consist of a constant number of disjoint faces. The maximum number of disjoint faces for a single segment is five.*

Proof. A segment may contribute an edge to at most two envelopes and it may induce at most four bounding lines. However, any two edges of the farthest hull that are adjacent produce neighboring regions in *FVD(S)*, which, if they are induced by the same segment, correspond to a single face. By definition, a segment may contribute to two envelopes only if it contributes their first or last edge. Thus, the maximum number of disjoint faces for a single segment is five, at most four for the bounding lines, plus one for a single potential envelope edge that is not incident to bounding lines. □

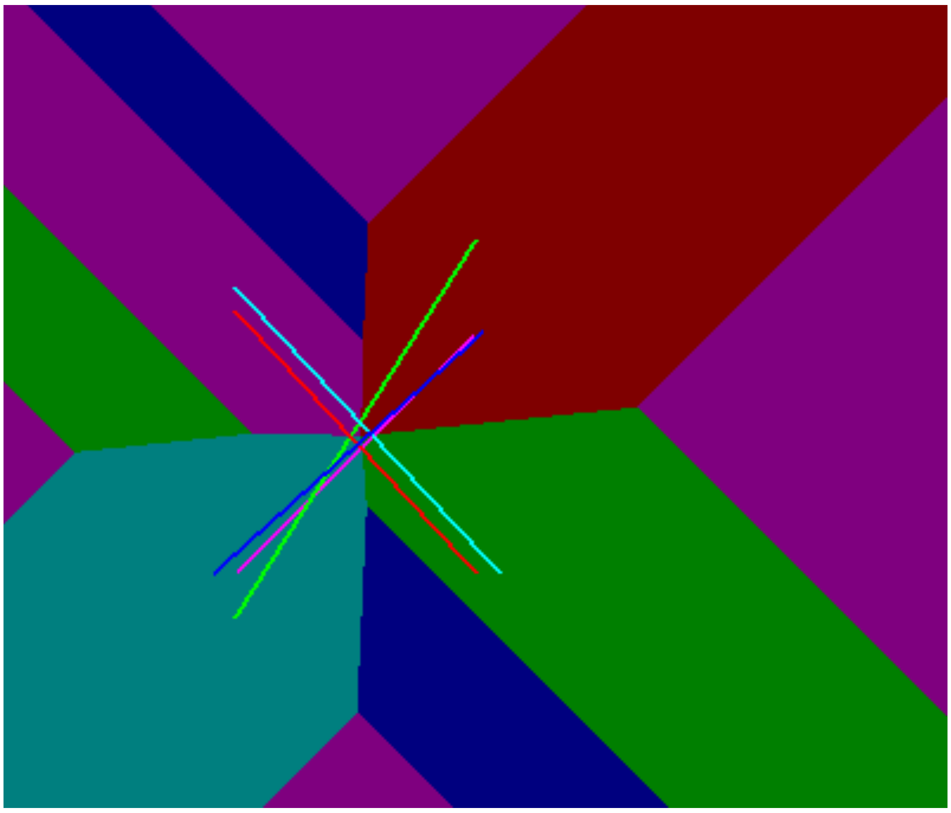


Figure 3.10. Example showing a segment (in purple) whose farthest Voronoi region consists of five disconnected faces.

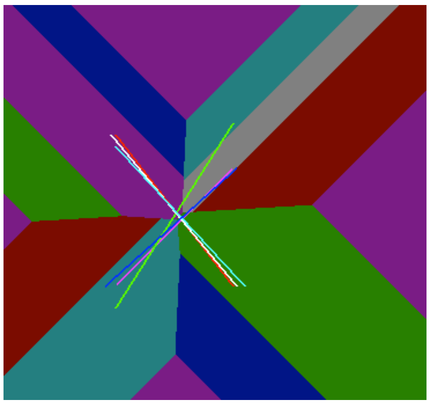


Figure 3.11. Example of $n = 6$ segments producing $n + 8$ disjoint faces.

Figure 3.10 illustrates an example of a segment having exactly five disjoint faces, where four of those five faces are the standard faces of the bounding lines.

This segment induces all four bounding lines plus it contributes one edge to an envelope that is not adjacent to any bounding line.

Theorem 2. *The L_∞ farthest Voronoi diagram of n arbitrary line segments may have at most $n + 8$ faces and this is tight. For non-crossing segments this number is eight.*

Proof. The total number of faces of the diagram is four (for the bounding lines) plus the total number of edges along the envelopes of the farthest hull. A segment may contribute an edge to at most two envelopes. However, only the first and the last edge of a pair of envelopes may be attributed to the same segment (by definition of an envelope). Thus, the total number of edges along the envelopes is at most $n + 4$ and the total number of faces in the diagram is at most $n + 8$. This bound is tight as shown in Figure 3.11, where four line segments produce eight disjoint faces, each one contributing to exactly two non-adjacent envelope edges. The segment in purple induces all four bounding lines plus one envelope edge that is not incident to any bounding line, thus, it produces 5 disjoint faces. Any other segment can be placed to contribute exactly one face. □

In the presence of multiple segments or multiple segment endpoints aligned along the same axis-parallel line, there may be a number of segments inducing a bounding line. Let $S(l_i)$ be the set of segments inducing l_i, where $i = n, s, e, w$. Then $reg(l_i)$ is equidistant from all segments in $S(l_i)$. In this case, a convention regarding the ownership of the region may be adapted. For example, we can

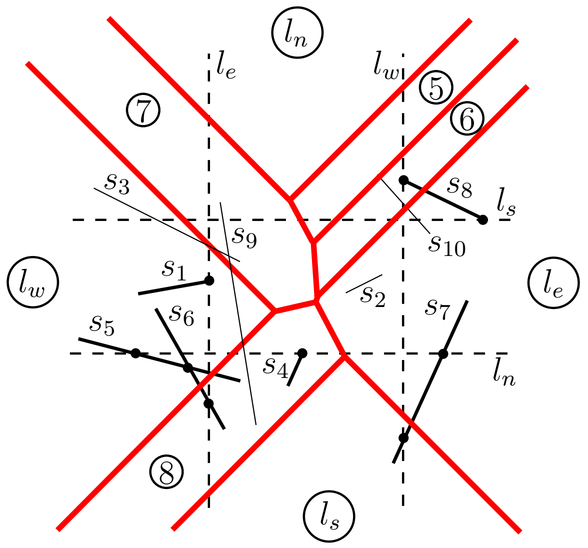


Figure 3.12. Farthest Voronoi diagram of segments in L_∞.

assign $reg(l_i)$ to one segment in $S(l_i)$ arbitrarily or to one segment according to a lexicographic order. Alternatively, we can attribute $reg(l_i)$ to l_i and characterize it as a region common to all segments in $S(l_i)$. The choice of convention depends on the needs of the application. Our choice is the latter convention as it avoids any artificial partitioning of equidistant regions.

Construction algorithm:
Following [6], the algorithm to construct *FVD(S)* (for example see Figure 3.12) proceeds in two steps:

1. Construct the farthest hull *FH(S)*. Given *FH(S)*, construct the circular list C of all unbounded bisectors

2. Given C, construct *FVD(S)*.

Step 1 takes $O(n \log h)$ time, where h is the size of the farthest hull. In particular, the bounding axis parallel lines l_n, l_s, l_e, and l_w can be constructed in $O(n)$ time. The envelopes of the segments straddling the four quadrants can be constructed by any convex hull type algorithm, thus, in $O(n \log h)$ time following [16]. The circular list C of all unbounded bisectors is derived in additional $O(h)$ time. Step 2 can be constructed in $O(h)$ time as a simplification of the algorithm in [6] for the L_∞ metric.

Algorithm 1 CONSTRUCT-FVD(S)

```
INPUT: A set S of $n$ arbitrary line segments.
OUTPUT: $FVD(S)$.
CONSTRUCT-FVD(S)
1. Compute the farthest-hull of S, *FH(S)*.
2. From *FH(S)* construct the circular list $C$ of unbounded
   bisectors.
3. Identify the four pairs of intersecting neighboring
   bisectors, compute their intersection point, and insert it
   in the intersection set $V$.
4. While (C is not empty)
      4.a. Report the intersection point (Voronoi vertex) in $V$
           with maximum weight. Let $b'_1, b'_2$ be the two
           bisectors in $C$ inducing the intersection
      4.b. Compute a new bisector $b'_3$ as defined by the
           two non-common edges of the farthest hull
           inducing of $b'_1$ and $b'_2$.
      4.c. Compute the intersection points of the new bisector
           with its two neighboring bisectors, choose the one
           of maximum weight and insert it in $V$.
      4.d. Delete $b_1, b_2$ from $C$ and insert $b_3$ in their
           position.
   end While
```

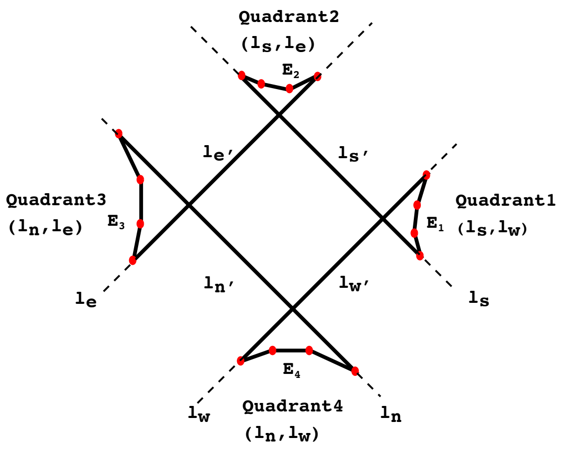


Figure 3.13. Farthest hull in the L_1 metric

The algorithm in [6] is based on the observation that vertices of *FVD(S)* can be discovered in order of decreasing *weight*, where the weight of any point v along *FVD(S)* is the radius of the disk (a square in L_∞) centered at v passing through the line segments whose bisector induces v (see also [70]). Intuitively, the tree structure of *FVD(S)*, denoted $T(S)$, can be regarded as a rooted tree, rooted at the point of minimum weight $v_{\min}$ (in L_∞ v_{min} may be an entire axis-parallel segment), where $v_{\min}$ is the locus of centers of the minimum size disk intersecting all segments in S. Along any path of the rooted $T(S)$ (except the root in L_∞) the weight of nodes is strictly increasing. Thus, we can compute $T(S)$ in a bottom up fashion, always proceeding in order of decreasing weight, until the node of minimum weight is reached.

In the L_∞ metric the implementation is particularly simple: There are only four pairs of neighboring bisectors that may intersect; all other unbounded bisectors are parallel rays of slope ± 1. As a result, the intersection of maximum weight can be determined in $O(1)$ time. We simply maintain a set V of size four of all possible intersections. Given a node of maximum weight v let $b'(e_i, e_k)$ and $b'(e_k, e_j)$ be the neighboring bisectors inducing v. Once v is selected, delete $b'(e_i, e_k)$ and $b'(e_k, e_j)$ from C and substitute them with the new bisector $b'(e_i, e_j)$; compute the intersections of $b'(e_i, e_j)$ with its two neighboring bisectors in C, choose the one of larger weight, and insert it in V. The algorithm is summarized in Algorithm 1. Its asymptotic complexity is clearly optimal in the worst case.

Theorem 3. *Given a set of arbitrary segments S, the farthest hull of S can be con-*

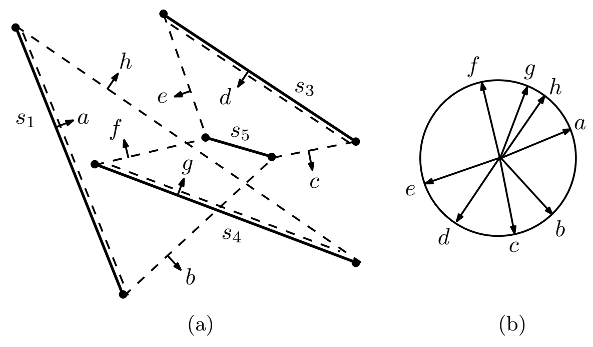


Figure 3.14. The farthest line-segment hull of segments in Figure 1.4 and its Gaussian map.

structed in $O(n \log h)$ time. Given the farthest hull, the L_∞ (L_1) farthest segment Voronoi diagram of S can be constructed in additional $O(h)$ time, where h is the number of edges on the farthest hull. h can vary from $O(1)$ to $O(n)$.

It is well known that the L_1 metric is equivalent to L_∞ under 45° rotation. Figure 3.13 illustrates the farthest hull of S in the L_1 metric. Unbounded bisectors in the L_1 version of $FVD(S)$ are axis-parallel rays. The construction is equivalent.

3.2 The farthest line-segment Voronoi diagram in L_p $(1 < p < \infty)$

We study the structural properties, define the farthest hull and its properties in Subsection 3.2.1. We describe the improved combinatorial bounds in Subsection 3.2.2, and the construction algorithms for the farthest line-segment hull in the L_2 metric (which also holds true in any L_p metric, $1 < p < \infty$) in Subsection 3.2.3.

3.2.1 Defining the farthest-hull and its Gaussian map

The *farthest hull* is a closed polygonal curve that encodes the faces and the unbounded bisectors of *FVD(S)* while it maintains their cyclic order. The cyclic order of the farthest hull is reflected by its *Gaussian map*. In the following, we define these concepts.

Definition 4. *A line ℓ through the endpoint p of a line-segment $s \in S$ is called a* supporting line *of S if an open halfplane induced by ℓ, denoted $H(\ell)$, intersects all segments in S, except s (and possibly except additional segments incident to p). Point p is said to* admit a supporting line *and it defines a vertex on the farthest hull. The unit normal of ℓ, pointing away from $H(\ell)$, is called the* unit vector *of ℓ and is denoted $\nu(\ell)$. If the line ℓ through s is supporting, i.e., $H(\ell)$ intersects all segments in $S \setminus \{s\}$, line-segment s is said to* admit a supporting line *and it defines a segment on the farthest hull of unit vector $\nu(s) = \nu(\ell)$, which is called a* hull segment.

A single line-segment s may result in two hull segments of two opposite unit vectors. This is the case when the line through s intersects all segments in S.

Definition 5. *The line-segment $\overline{pq}$ joining the endpoints p, q, of two line segments $s_i, s_j \in S$ is called a* supporting segment *if the open halfplane induced by the line ℓ through $\overline{pq}$, at opposite side of ℓ than s_i, s_j, denoted by $H(\overline{pq})$, intersects all segments in S, except s_i, s_j (and possibly except additional segments incident to p, q). The unit normal of $\overline{pq}$ pointing away from $H(\overline{pq})$ is called the* unit vector *of $\overline{pq}$, $\nu(\overline{pq})$.*

Aurenhammer et al.[6] showed that a segment s_i has a non-empty farthest Voronoi region (unbounded in direction ϕ) if and only if there exists an open halfplane (normal to ϕ) which intersects all segments in S but s_i. Thus, $freg(s_j) \neq \emptyset$ if and only if s_i, or an endpoint of s_i, admits a supporting line. The unbounded bisectors of *FVD(S)* correspond exactly to the supporting segments of S. Let C denote the circular list of unit vectors of all the supporting and hull segments of S following their angular order. The angular ordering of C implies an ordering in the set of supporting and hull segments of S that corresponds exactly to the cyclic ordering of the faces of *FVD(S)* at infinity. Figure 3.14(b) illustrates the ordering of C for the set of segments whose farthest Voronoi diagram appears in Figure 1.4.

Theorem 4. *Let S be a set of line-segments in the plane. The sequence of the hull segments and the supporting segments of S, ordered according to the angular order of their unit vectors, forms a closed, possibly self-intersecting, polygonal curve.*

We call the polygonal curve, subject of Theorem 4, the *farthest line-segment hull* of S (for brevity, the *farthest hull*), and denote it by f-hull(S). Figure 3.15 illustrates the farthest hull for the Voronoi diagram of Figure 1.4. Its ordering is revealed by the angular order of unit vectors in Figure 3.14(b).

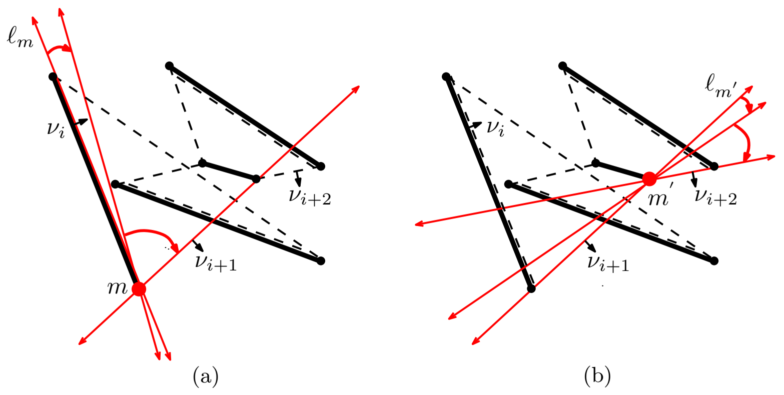


Figure 3.15. Proof of Theorem 4.

Proof. Consider the circular list C of the unit vectors of S. By Defs. 4 and 5, unit vectors are unique, i.e., no two different supporting lines or supporting segments of S can have the same unit vector. Let $\nu_i = \nu(e_i)$, $\nu_{i+1} = \nu(e_{i+1})$, $\nu_{i+2} = \nu(e_{i+2})$ be three consecutive unit vectors in C in say clockwise order, where e_i, e_{i+1}, e_{i+2} are their corresponding supporting segments or hull segments (see Figure 3.15). Consider the line through e_i. By the definition of a supporting line or a supporting segment, there must be an endpoint m of e_i, such that if we rotate the line ℓ_m through e_i, infinitesimally clockwise around m, ℓ_m will become a supporting line through m (see Figure 3.15 (a)). Vertex m is chosen among the endpoints of e_i according to whether e_i is a hull segment or a supporting segment. As we rotate ℓ_m clockwise around m it must remain a supporting line of S, leaving the segment s_m incident to m lying entirely in the complement halfplane of $H(\ell_m)$, until $\nu(\ell_m) = \nu_{i+1}$. At that time, since unit vectors are unique, ℓ_m must be the line through e_{i+1}, and thus, e_{i+1} must be incident to m (by Defs. 4, 5). However, if we continue to rotate ℓ_m clockwise past ν_{i+1}, ℓ_m stops being a supporting line. Thus, e_{i+2} cannot be incident to m. Note that e_{i+2} must be incident to the other endpoint of e_{i+1}, denoted as m' in Figure 3.15(b).

We showed that by the argument above, any two consecutive edges on f-hull(S) must be incident to the same vertex while no three consecutive ones can. Thus, f-hull(S) must form a closed polygonal curve. The vertices of the polygonal curve may repeat, that is, multiple vertices of f-hull(S) may correspond to the same point in S. Two different hull edges may correspond to the same segment. □

Remarks. The vertices of the farthest hull are exactly the endpoints of S that admit a supporting line. The edges are of two types: supporting segments and hull segments. Recall that two different hull segments may correspond to a single segment in S, and that several hull vertices may correspond to the same segment endpoint. A supporting line of S is exactly a supporting line of the farthest hull. Any maximal chain of supporting segments between two consecutive hull segments must be convex. If the line-segments in S degenerate to points, the farthest hull corresponds exactly to the convex hull of S.

Consider the Gaussian map, for short Gmap, of the farthest hull of S onto the unit circle, denoted as Gmap(S). This is a mapping of the farthest hull onto the unit circle K_o, such that every edge e is mapped to a point on K_o as obtained by its unit vector $v(e)$, and every vertex is mapped to one or more arcs as delimited by the unit vectors of the incident edges (see Figure 3.14). The Gaussian map can be viewed as a cyclic sequence of vertices of the farthest hull, where each vertex is represented as an arc of K_o. Recall that multiple vertices of the farthest hull, and thus, several arcs of the Gmap, may correspond to the same endpoint in S. Each point along an arc of the Gmap reveals the unit vector of a supporting line. The Gaussian map provides an encoding of all the supporting lines of the farthest hull. It also provides an encoding of the unbounded bisectors, the regions of the farthest line-segment Voronoi diagram, and the directions along which every face of the diagram is unbounded. It can be readily obtained by the circular list of unit vectors C.

Corollary 3. *FVD(S) has exactly one unbounded bisector (unbounded Voronoi edge) for every supporting segment of f-hull(S), which is unbounded in the direction opposite to its unit vector. Unbounded bisectors in FVD(S) are cyclically ordered following exactly the cyclic ordering of Gmap(S).*

By Corollary 3, there is a 1-1 correspondence between the faces of *FVD(S)* and *maximal arcs* along the Gmap between pairs of consecutive unit vectors of supporting segments. There are two types of maximal arcs along the Gmap: *segment arcs*, which consist of a segment unit vector and its two incident arcs of the segment endpoints, and *single-vertex* arcs, which are single arcs bounded by the unit vectors of the two incident supporting segments. A segment arc corresponds to one hull segment while a a single-vertex arc corresponds to one occurrence of a hull vertex between two supporting segments.

Using the Gmap we can easily define the *upper* (resp., *lower*) farthest hull similarly to an ordinary upper (resp., lower) convex hull. In particular, the portion of the Gaussian map above (resp., below) the horizontal diameter of the unit circle is referred to as the *upper* (resp., *lower*) Gmap. We can define the upper (resp., lower) farthest hull as the portion that corresponds to the upper

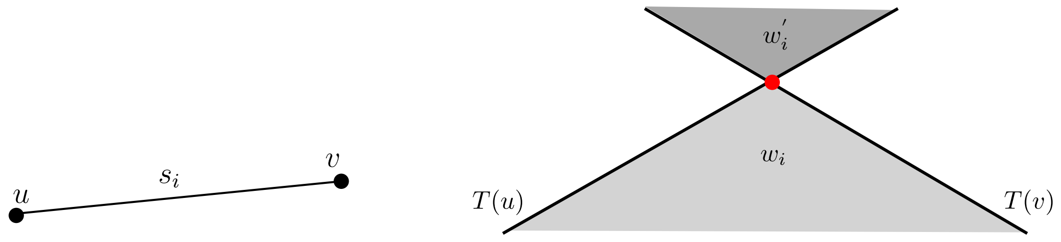


Figure 3.16. Example of a line-segment s_i and its dual. The lightly shaded region is the lower wedge, and the dark shaded region is the upper wedge. The red point in the dual plane is the supporting line of the line-segment s_i.

(resp., lower) Gmap, which is similar to an ordinary upper (resp., lower) convex hull.

In Definitions 4, 5, we used the unit vectors of supporting lines and segments pointing away from the farthest hull in order to simulate a convex hull. Alternatively, we could use the unit vectors in their exact opposite direction and derive an equivalent map on the unit circle in mirror direction, which reflects exactly the faces of *FVD(S)* at infinity, their unbounded directions, and the unbounded bisectors of *FVD(S)*.

3.2.2 Improved combinatorial bounds

In this subsection, we give tighter upper and lower bounds on the number of faces of the farthest line-segment Voronoi diagram for arbitrary line-segments. To start our approach for computing the combinatorial bounds on the number of faces of *FVD(S)*, we use the standard point-line duality transformation T [6], that maps a point $p = (a, b)$ in the primal plane to a line $T(p) : y = ax - b$ in the dual plane, and vice versa. A line-segment $s_i = uv$ is sent into the wedges w_i and w_i' that lie below and above respectively both lines $T(u)$ and $T(v)$, referred to as the lower and upper wedge respectively (see Figure 3.16). Note that wedges w_i and w_i' must contain the vertical ray that emanates from their apex and is directed to $-\infty$ and to ∞ respectively. Define E to be the boundary of the union of the lower wedges $w_1, \dots, w_n$ (see Figure 3.17) and E' to be the boundary of the union of the upper wedges $w_1', \dots, w_n'$. Then as shown by Aurenhammer et al. [6], the faces of *FVD(S)*, which are unbounded in directions 0 to π in cyclic order, correspond to the edges of E in x-order [6]. Respectively for the edges of E' and the Voronoi faces unbounded in directions π to 2π.

Let a *start-vertex* and an *end-vertex* respectively stand for the right and the left endpoint of a line-segment. In terms of a lower wedge, a start-vertex is the left ray of the wedge and an end-vertex is the right ray. Let an *interval* $[a_i, a_{i+1}]$ denote the portion of the lower Gmap between two consecutive (but not adjacent) occurrences of arcs for segment $s_a = (a', a)$, where a, a' denote the start-vertex and end-vertex of s_a respectively. Interval $[a_i, a_{i+1}]$ is assumed to be *nontrivial* i.e., it contains at least one arc in addition to a, a'. The following lemma is easy to derive using the duality transformation.

Lemma 4. *Let $[a_i, a_{i+1}]$ be a nontrivial* interval *of segment $s_a = (a', a)$ on lower Gmap(S). We have the following properties:*

1. *The vertex following a_i (resp., preceding a_{i+1}) in $[a_i, a_{i+1}]$ must be a start-vertex (resp., an end-vertex).*
2. *If a_i is a start-vertex (resp., a_{i+1} is end-vertex), no other start-vertex (resp., end-vertex) in the interval $[a_i, a_{i+1}]$ can appear before a_i or past a_{i+1} on the lower Gmap, and no end-vertex (resp., start-vertex) in $[a_i, a_{i+1}]$ can appear before a_i (resp., past a_{i+1}) on the lower Gmap.*

To count the number of vertex re-appearances along the lower Gmap we use the following charging scheme for a nontrivial interval $[a_i, a_{i+1}]$: If a_i is a start-vertex, let u be the vertex immediately following a_i in $[a_i, a_{i+1}]$; the appearance of a_{i+1} is charged to u, which must be a start-vertex, by Lemma 4. If a_{i+1} is an end-vertex, let u be the vertex in $[a_i, a_{i+1}]$ immediately preceding a_{i+1}; the appearance of a_i is charged to u, which must be an end-vertex, by Lemma 4.

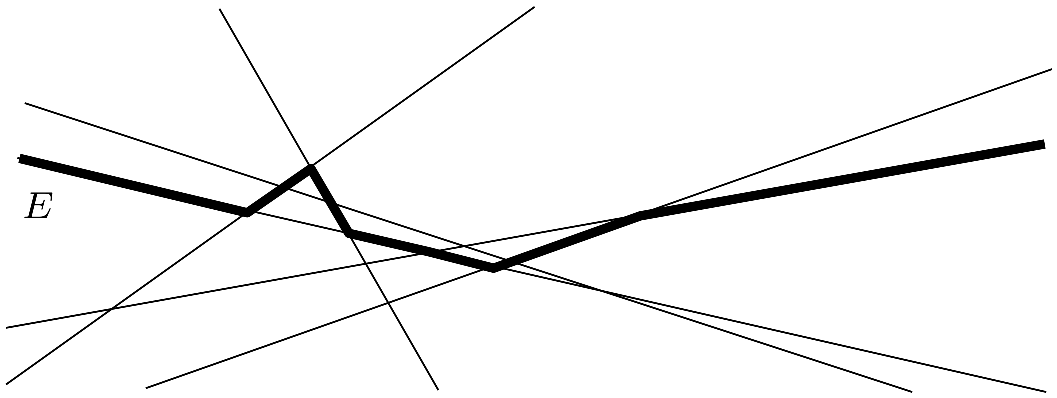


Figure 3.17. Example of union of dual wedges [6].

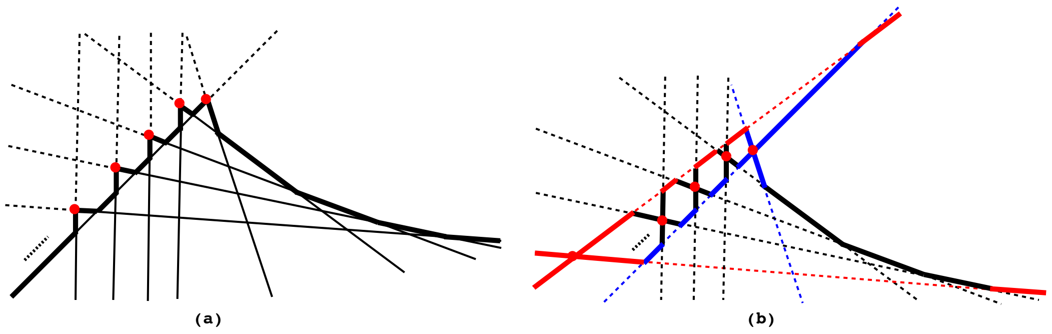


Figure 3.18. (a)Lower Gmap of $3n-2$ arcs. (b)Gmap of $5n-6$ arcs.

Lemma 5. *The re-appearance along the lower Gmap of an endpoint of a segment s_a is charged to a unique segment endpoint u (a start-vertex or an end-vertex of the lower Gmap) such that no other re-appearance of a segment endpoint on the lower Gmap can be charged to u.*

Proof. Let $[a_i, a_{i+1}]$ be a nontrivial interval of the lower Gmap and let u be the endpoint that has been charged the re-appearance of a_{i+1} (or a_i) as described in the above charging scheme. By Lemma 4, all occurrences of u must be in $[a_i, a_{i+1}]$. Suppose (for contradiction) that u can be charged by the reappearance of some other vertex c. Assuming that u is a start-vertex, c must also be a start-vertex, and an interval $[cu...c]$ must exist such that $cu \in [a_i, a_{i+1}]$. Thus, $[cu...c] \in [a_i, a_{i+1}]$. But then u could not appear outside $[cu...c]$, contradicting the fact that u has been charged the reappearance of a_{i+1}. Similarly for an end-vertex u. □

By Corollary 3, the number of faces of *FVD(S)* equals the number of supporting segments of the farthest hull, which equals the number of *maximal arcs* along the Gmap between consecutive pairs of unit vectors of supporting segments.

Lemma 6. *The number of maximal arcs along the lower (resp., upper) Gmap, and thus, the number of faces of FVD(S) unbounded in directions 0 to π (resp., π to 2π), is at most 3n-2. This bound is tight.*

Proof. Consider the sequence of all occurrences of a single segment $s_a = (a', a)$ on the lower Gmap. It is a sequence of the form

$$\ldots a \ldots aa' \ldots a' \ldots \quad \text{or} \quad \ldots a \ldots a \ldots a' \ldots a' \ldots$$

The number of maximal arcs involving s_a is exactly one plus the number of (nontrivial) intervals involving the endpoints of s_a. Summing over all segments, the total number of maximal arcs on the lower Gmap is at most n plus the total number of vertices that may get charged due to a nontrivial interval (i.e., the reappearance of a vertex). By Lemma 5, a vertex can be charged at most once and there are 2n vertices in total. However, the first and last vertices along the lower Gmap cannot be charged at all, thus, in total $3n-2$. Similarly for the upper Gmap.

Figure 3.18a illustrates an example, in dual space, of n line-segments (lower wedges) whose lower Gmap (boundary of the wedge union) consists of $3n-2$ maximal arcs. There are exactly n hull segments plus $2n-2$ charges for vertex (wedge) reappearances. □

Theorem 5. *The total number of faces of the farthest line-segment Voronoi diagram of a set S of n arbitrary line-segments is at most $6n-6$. A corresponding lower bound is $5n-6$.*

Proof. The upper bound is derived by Lemma 6 and Corollary 3. A lower bound of $5n-6$ faces can be derived by the example, in dual space, illustrated in Figure 3.18b. A set n line-segments are depicted as lower and upper wedges using the point-line duality. There are $2n$ hull segments, $2(n-2)+1$ charges for vertex reappearances on lower wedges, and $n-1$ charges on upper wedges. Thus, a total of $5n-4$, minus the two common elements of the upper and lower Gmap. □

Theorem 5 improves the $8n+4$ upper bound and the $4n-4$ lower bound on the number of faces of the farthest line-segment Voronoi diagram given by Aurenhammer et al.[6], based on results of Edelsbrunner et al.[27]. For disjoint segments the corresponding bound is $2n-2$, which is tight.[20]

3.2.3 Algorithms for the farthest line-segment hull

Using the Gmap, we can adapt most standard techniques to compute a convex hull with the ability to compute the farthest hull, within the same time complexity. For example, we can adapt Chan's output sensitive approach and compute the farthest hull in output sensitive $O(n \log h)$ time, where h is the size of the hull. As a result, the farthest line-segment Voronoi diagram can be computed in output sensitive $O(n \log h)$ time. Recall that h equals the number of faces on the farthest line-segment Voronoi diagram. Our goal is to unify techniques for the construction of the farthest-point and farthest line-segment Voronoi diagram.

In this section we give algorithmic details for a divide and conquer technique to compute the farthest hull.

Divide and conquer construction of the farthest hull

We list properties that lead to a linear-time merging scheme for the farthest hulls of any two sets of segments, L, R, $L \cap R \neq \emptyset$, and their Gaussian maps. The merging scheme gives an $O(n \log n)$ divide-and-conquer approach and to an $O(n \log n)$ two-stage incremental construction. Recall that the farthest hulls of L and R may have $\Theta(|L| + |R|)$ supporting segments between them.

For farthest-hull edge $e \in L$, let the R-vertex of e be the point v in R such that $v(e)$ falls along the arc of v in Gmap(R). Respectively for e in R. A supporting line, an edge or a vertex of the farthest hull of L or R and their corresponding unit vector or arc are called *valid* if they remain in the farthest hull of $L \cup R$.

Lemma 7. *A a farthest hull edge e in f-hull(L) of unit vector $v(e)$ remains valid in f-hull(L $\cup$ R) if and only if its R-vertex lies in the halfplane $H(e)$.*

Proof. Let e be an edge of f-hull(L) and let q be its R-vertex. Suppose e remains valid in f-hull$(L \cup R)$. Then the line $\ell(q)$, parallel to e, passing through q, must be supporting to f-hull(R). Consider $H(e)$. If $q \in H(e)$ then ℓ_q must be contained entirely in $H(e)$. Since $\ell(q)$ is supporting to f-hull(R), $H(e)$ must intersect all segments in R. Since $H(e)$ must also intersect all segments in L, except those inducing e, e must be valid. If $q \notin H(e)$, then ℓ_q lies entirely outside $H(e)$ and thus, the segment incident to q cannot not intersect $H(e)$, that is, e must be invalid. □

Corollary 4. *If $v(e)$ in Gmap(L) is invalid, its R-vertex must be valid.*

Proof. If e is invalid, then its R-vertex $q \notin H(e)$. Thus, $H(\ell_q)$, where ℓ_q is the line parallel to e passing through q, must intersect all segments in L and all segments in R except the one incident to q. Thus, q must be valid. □

Lemma 8. *An arc in Gmap(L), i.e., a hull-vertex m of f-hull(L), remains valid in Gmap(L $\cup$ R) if and only if either an incident unit vector remains valid, or m is the L-vertex of an invalid unit vector in Gmap(R).*

Proof. A vertex incident to a valid farthest hull edge must clearly be valid. By Corollary 4, if m is the L-vertex of an invalid edge in f-hull(R) then m must be valid.

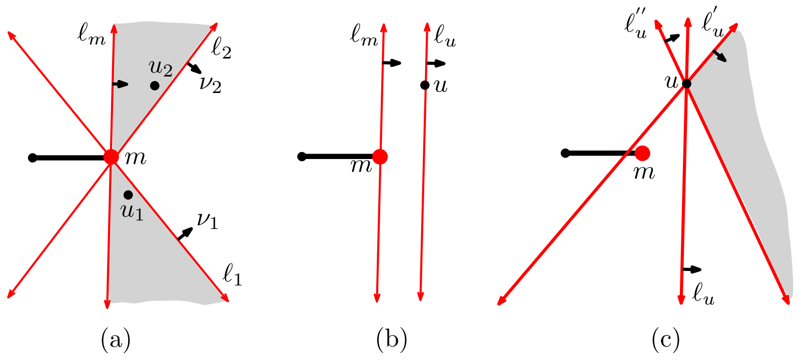


Figure 3.19. Proof of Lemma 8.

Conversely, suppose that m is valid but both incident unit vectors ν_1 and ν_2 in Gmap(L) are invalid. Since m is valid, there is a supporting line of $L \cup R$ passing through m, denoted ℓ_m, such that $\nu(\ell_m)$ is between ν_1 and ν_2 on the arc of m (see Figure 3.19(a)). Let u_1, u_2 be the R-vertices of ν_1 and ν_2 respectively. Since ν_1, ν_2 are invalid, $u_1 \notin H(\ell_1)$ and $u_2 \notin H(\ell_2)$, where ℓ_1, ℓ_2 are the corresponding supporting lines of L. Since m is valid, ℓ_m is a supporting line of $L \cup R$, thus, u_1 and u_2 must lie in $H(l_m) \cap H'(\ell_1)$ and $H(l_m) \cap H'(\ell_2)$ respectively, where $H'(l)$ is the complement of $H(l)$. But these regions, which are shown shaded in Figure 3.19(a), are disjoint, therefore, $u_1 \neq u_2$. Thus, there is a unit vector separating the arcs of u_1 and u_2 in Gmap(R), whose L-vertex is m.

Let u be the R-vertex of $\nu(\ell_m)$ in Gmap(R). Let ν'_1 and ν'_2 denote copies of unit vectors ν_1 and ν_2 in Gmap(R). Let ν be a unit vector in R bounding the arc of u between ν'_1 and ν'_2, i.e, ν has m as its L-vertex. Such a unit vector must exist in Gmap(R) by the argument above. Since ℓ_m is valid, by Lemma 7, u must be in $H(\ell_m)$. Consider the line ℓ_u parallel to l_m passing through u (see Fig 3.19(b)). It lies entirely in $H(\ell_m)$. If we rotate ℓ_u slightly clockwise and counterclockwise around u into lines ℓ'_u and ℓ''_u respectively, it is clear that m cannot be contained in both $H(\ell'_u)$ and $H(\ell''_u)$ (see Figure 3.19(c) where $H(\ell'_u) \cap H(\ell''_u)$ is shown shaded). Thus, not both $\nu(\ell'_u)$ and $\nu(\ell''_u)$ can be valid. If we continue the rotation around u we will reach the unit vectors bounding the arc of u in Gmap(R), among which one may be ν'_1 or ν'_2. For the same reason, not both bounding vectors can be valid. Since both ν'_1 or ν'_2 are valid in Gmap(R), an invalid unit vector must be between them. □

Let Gmap(L)∪Gmap(R) denote the circular list of unit vectors and arcs derived by superimposing Gmap(L) and Gmap(R). We can determine valid and

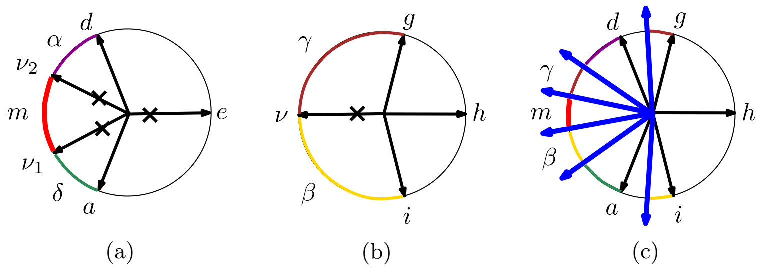


Figure 3.20. (a) Gmap(L). (b) Gmap(R). (c) Gmap($L \cup R$), merge vectors are inserted between consecutive valid vertices in different sets.

invalid unit vectors as well as valid and invalid hull vertices (arcs) using Lemmas 7 and 8. Then Gmap(L)∪Gmap(R) represents a circular list of the vertices in f-hull($L \cup R$). Recall that by Lemma 8 any invalid vector represents a valid vertex in the opposite set. For any pair of consecutive vertices, one in L and one in R, a new unit vector, referred to as a *merge vector*, needs to be inserted corresponding to the new supporting segment between the two hulls joining these two vertices. After inserting the merge vectors into Gmap(L)∪Gmap(R), all invalid vectors can be deleted, and we obtain Gmap($L \cup R$). The merging process is illustrated in Figure 3.20. Merge vectors are indicated by longer arrows. The merging process is clearly linear. We thus, conclude.

Lemma 9. *Gmap(L) and Gmap(R) can be merged into Gmap(L ∪ R) in linear time.*

The merging process implies a simple $O(n \log n)$-time divide-and-conquer algorithm to compute the farthest hull of S.

In an incremental process the merging phase is performed such that one set is a single segment. In this case, say $R = \{s\}$, the merging process can be refined as follows: Let $\nu_1(s)$ be the unit vector of s in its upper Gmap. (1) Perform binary search to locate $\nu_1(s)$ in the upper Gmap(L); (2) Move counterclockwise along the upper Gmap(L) sequentially, to test the validity of the encountered unit vectors, until either a valid start-vertex of x-coordinate smaller than the start-vertex of s is found or the beginning of upper Gmap(L) is reached; and (3) Move clockwise along the upper Gmap(L) until either a valid end-vertex of x-coordinate larger than the end-vertex of s is found or the end of the upper Gmap(L) is reached. In the process, all relevant supporting segments in the upper Gmap are identified. Similarly for the lower Gmap.

Some other standard convex-hull techniques such as Jarvis march, quick hull, and an output sensitive Chan's algorithm [16] can also be adapted to construct the farthest line-segment hull with same time complexity [67].

3.3 Summary

In this chapter we have studied combinatorial properties and construction algorithms of the farthest line-segment diagram in the general L_p metric, $1 \leq p \leq \infty$. The most interesting ones are the L_∞ (max-norm) and the L_2 metric (Euclidean). The results in the L_2 metric extends easily to the L_p metric. The contents of this chapter appear in research papers [66], [25], [65], [67].

We introduced the farthest line-segment hull and its Gaussian map in the L_2 metric. The farthest line-segment hull encodes the faces and characterizes the unbounded Voronoi edges.

We improved the combinatorial bounds on the number of faces of the diagram. For the L_2 metric, we improved the upper bound from $8n+4$ to $6n-6$ and the lower bound from $4n-4$ to $5n-6$. We also proved exact bound for the L_∞ (L_1) metric, which is $n+8$. In the L_∞ (L_1) metric, for a set of disjoint line segments this bound is constant and is equal to eight.

In the L_∞ metric, the unbounded Voronoi edges have constant number of directions, and thus given the farthest hull, the farthest Voronoi diagram can be constructed in $O(h)$ time, where h is the size of the farthest hull.

We gave a divide and conquer algorithm to construct the farthest hull in $O(n \log n)$ time.

The derivation of results on the farthest line-segment Voornoi diagram has been a joint work with major contribution from Prof. Evanthia Papadpoulou.

Chapter 4

CGAL implementation of the line-segment Voronoi diagram in the L_∞ metric

"Out of clutter, find simplicity."

Albert Einstein

The CGAL project was founded in 1996 [28] with the following goal:

" *Make the large body of geometric algorithms developed in the field of computational geometry available for industrial application* "

The current version of CGAL which is CGAL-4.6 [90] is released on April 2015 and consists of approximately $600K$ lines of $C++$ source code. It contains various geometric algorithms and data-structures such as convex hull algorithms, triangulations, Voronoi diagrams, arrangement of curves, and many more.

The major portion of the content of this chapter is published in the International Congress on Mathematical Software (ICMS) 2014 [19]. The development of the software package for the line-segment Voronoi diagram in the L_∞ metric is a joint work Dr. Panagiotis Cheilaris. Our software package is accepted and integrated to the upcoming version of the library CGAL 4.7, and will be released in September 2015.

Line-segment Voronoi diagrams encode proximity information between polygonal objects. In many applications, proximity is most naturally expressed with the Euclidean distance, but there are applications, particularly in integrated circuits design and manufacturing, for which the L_∞ distance provides a good and simpler alternative. A detailed motivation for using the L_∞ metric is given in the Section 1.1 of the Introduction of this dissertation.

In this chapter, in Section 4.1 we briefly discuss the basic terminology used in CGAL, structure and architecture of CGAL, and important fundamental concepts used in CGAL. Then in Section 4.2 we review the existing implementation of the line-segment Voronoi diagram in the Euclidean plane. Finally, in Section 4.3 we discuss our implementation of the L_∞ segment Voronoi diagram in CGAL.

4.1 CGAL preliminaries

The geometric algorithms and data structures in CGAL are implemented under the design goals of robustness, genericity, flexibility, efficiency, and ease of use [28, 29]. These design goals are fulfilled in CGAL by choosing C++ generic programming, using template classes, and function templates. The high level block diagram of C++ Standard Template Library (STL) architecture is shown in Figure 4.1. In STL the algorithms are de-coupled from the containers (data-structures) using *iterators* [80] (iterators are used to traverse data-structures), and the functionality of an algorithm is provided using *functors* [80]. CGAL follows the STL style of coding (see Figure 4.2) with some more modifications. For an example, apart from iterators, CGAL has *circulators*, which is a natural need for geometric software, where many objects are closed (you would like to circle around the object) like convex hull.

Genericity and flexibility: Genericity and flexibility are achieved by *concepts* and *models* in the generic programming paradigm. A concept is a set of requirements that must be satisfied by a class. A model of a concept is a class that fulfills those requirements. For example, the *iterator* concept enables the use of the same algorithms for different containers (data structures). In CGAL, all algorithms are passed with a special template parameter called a *traits class* [59]. A traits class represents a concept and so has a corresponding set of requirements that define the interface between the algorithm and the geometric (or numeric) primitives it uses. Any concrete class that serves as a model for this concept is a traits class model for the given algorithm or data structure.

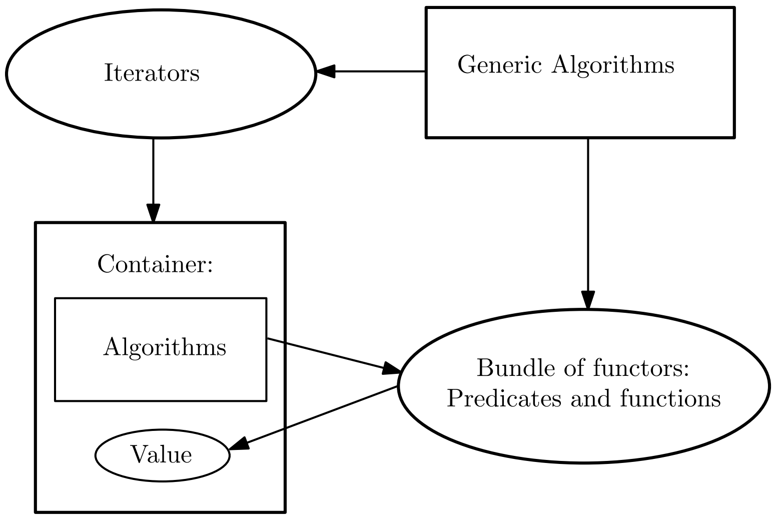


Figure 4.1. The STL design in C++.

The use of traits concept as template parameters allows for the customization of the behavior of algorithms without changing their implementations. At least one model for each traits concept is provided in CGAL. Users can also provide their own traits class satisfying all the requirements of traits concept.

Predicates and constructions: There are two types of geometric computations, one is a *predicate* which is associated with a branching decision in the algorithm and determines the flow of the algorithm, and the other is a *construction* which is associated with generating the output of the algorithm. Any approximation in the predicate computation can cause incorrect branching of the algorithm and can lead to failure of the implementation. Any approximation in the construction computation is acceptable as long as the maximum absolute error is within the resolution required by the application as well as the approximation also gives a geometrically consistent result. Passing the approximated results of construction to the predicate can cascade the error and again fall in an unacceptable state.

Exact arithmetic and floating point filters: The use of exact arithmetic to handle the problems incurred due to the use of floating point arithmetic is discussed by Yap in [94]. Geometric algorithms can be written using integer or rational number types, but the required bit length even for the simplest geometric computation can exceed the built in machine precision. This is handled in CGAL by

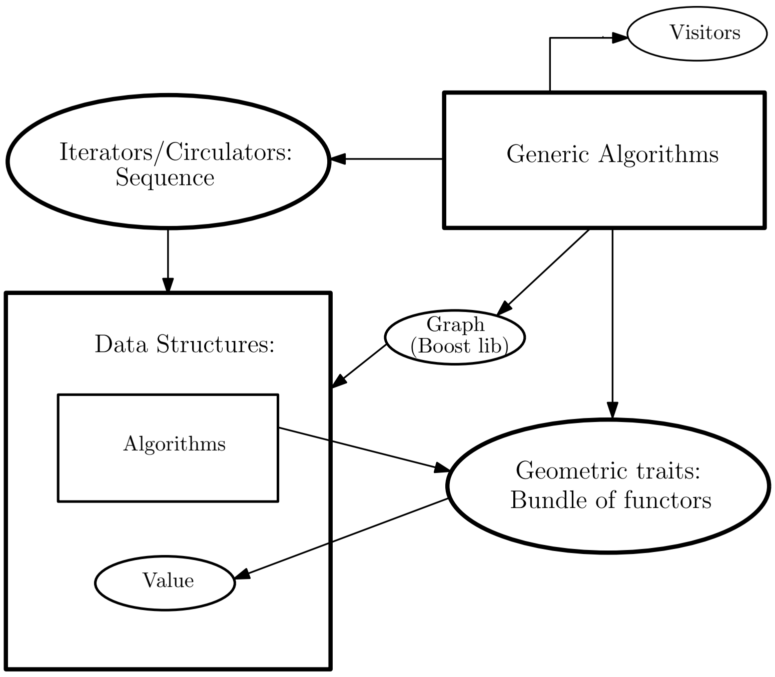


Figure 4.2. The CGAL design.

using multiple precision software packages such as *GNU Multiprecision Package* (GMP) [34], LEDA's integer and rational number types [56]. The CGAL kernel is template parametrised by number types, so there is a flexibility of using the number type depending on the application. The use of exact arithmetic in a naive way may result in slow execution time.

In many cases we do not need exact arithmetic as the floating point arithmetic can give correct results. To handle computationally intensive predicates, CGAL uses two different kinds of filtering techniques: *geometric filtering* and *arithmetic filtering*. The geometric filtering technique amounts to performing simple geometric tests exploiting the representation of data, as well as the geometric structure inherent in the problem, in order to evaluate predicates in seemingly degenerate configurations. Geometric filtering can be seen as a preprocessing step before performing arithmetic filtering. Geometric filtering can help by pruning situations in which the arithmetic filter will fail, thus decreasing the number of times one needs to evaluate a predicate using exact arithmetic. Arithmetic filtering first tries to evaluate the predicates using a fixed-precision floating-point number type (such as double), and at the same time keeps error

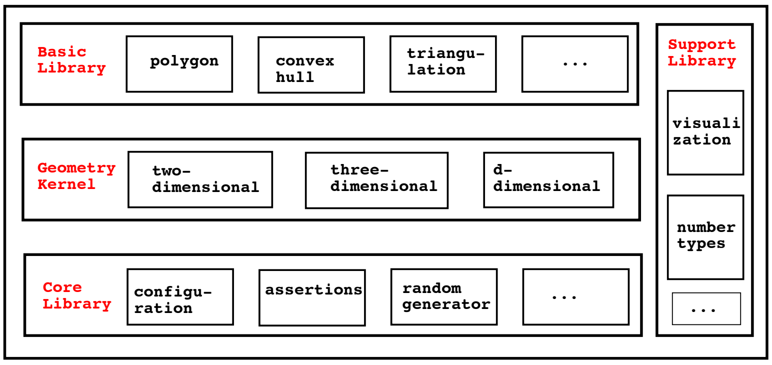


Figure 4.3. Structure of CGAL [29].

bounds on the numerical errors of the computations performed. If the numerical errors are too big and do not permit to evaluate the predicate, it switches to an exact number type, and repeats the evaluation of the predicate. This way CGAL keeps the speed of a multiple of floating point arithmetic and degrades to exact arithmetic only when it is necessary. There are three types of floating point filters.

1. *Static filters* [31], where the error bounds are computed at compile time and need specific information on the input data. Static filters are very efficient as they only require comparison with the already computed error bound at run time.

2. *Semi dynamic filters* [56], which bound the size of the operands at run time, and some factors are computed at compile time.

3. *Fully dynamic filters* [56], where the error bound is computed fully at runtime. Interval arithmetic [12] can also be used as a fully dynamic filter, first all computations are done using interval arithmetic, then the computation is repeated in the second step only if the computed intervals do not allow reliable comparisons.

Overview of the CGAL structure: The library is structured into three layers (see Figure 4.3) [29]: the core library with basic non-geometric functionalities, the geometry kernel with basic geometric objects and operations on the geometric objects, and the basic library with more complicated algorithms and data structures. There is also a support library that deals with visualization, number types, streams, and STL extensions. In CGAL we can choose the underlying number types and arithmetic. We can choose implementations with

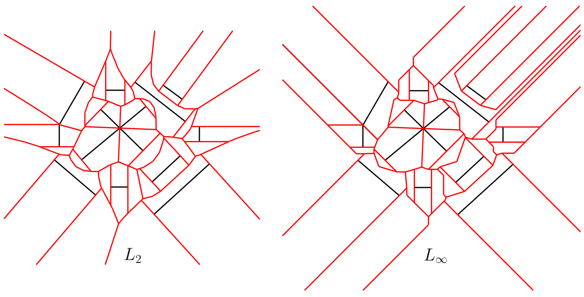


Figure 4.4. Segment Voronoi diagram under L_2 and L_∞ metric, with distinct sites for interiors of segments and their endpoints.

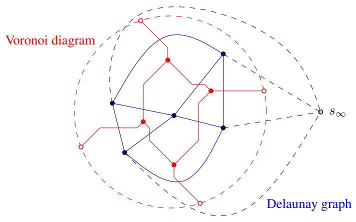


Figure 4.5. L_∞ Voronoi diagram and Delaunay graph of five point sites (black) with additional site s_∞ at infinity (infinite edges: dashed; Voronoi vertices: red, finite ones filled and infinite ones unfilled).

fast but occasionally inexact arithmetic or implementations guaranteeing exact computation and exact results. Since there are more applications in 2-D, 3-D, as compared to general dimension, CGAL provides separate kernel representations for 2-D, 3-D, and general dimension objects.

4.2 Reviewing L_2 segment Voronoi diagram in CGAL

The *2D segment Delaunay graph package* provides a randomized incremental construction of the L_2 segment Voronoi diagram. The input S is a set of points and segments (supports intersecting line-segments). The package supports intersections of segments by computing internally the *arrangement* $\mathscr{A}(S)$ of the input sites (segments and points). Internally, each “closed” input segment AB is converted to three sites, namely its two endpoints A, B and the “open” part of the segment (AB). For example, for input $S = \{AB, CD\}$, where the two segments cross at their interior at point E, the diagram is computed for the arrangement $\mathscr{A}(S) = \{A, B, C, D, E, (AE), (EB), (CE), (ED)\}$. This increases the combinatorial complexity of the diagram by a constant factor, Computing the Voronoi diagram over the arrangement guarantees all bisectors in the Voronoi diagram to be one-dimensional and all Voronoi cells to be simply connected. As a result, the resulting diagram is an *abstract* Voronoi diagram [47], which can be efficiently computed. The expected cost of inserting n sites is $O((n+m)\log^2 n)$, where m is the number of intersections of the n input segments and points.

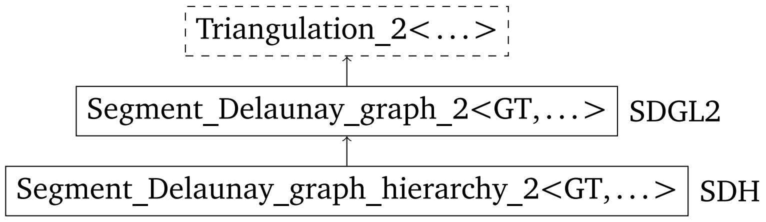

Triangulation_2<...>
Segment_Delaunay_graph_2<GT,...>
SDGL2
Segment_Delaunay_graph_hierarchy_2<GT,...>
SDH

Figure 4.6. Previous algorithm classes for SDG L_2 package.

The dual graph of the segment Voronoi diagram is also a plane graph and is called the *segment Delaunay graph*. Faces of the Voronoi diagram correspond to vertices (or sites) in the Delaunay graph. Vertices of the Voronoi diagram correspond to faces in the Delaunay graph. Edges of the Voronoi diagram correspond to edges in the Delaunay graph. It is typical to always include an additional site s_∞ at infinity, as it simplifies the construction algorithms [48]. See Figure 4.5. As its name suggests, the Segment Delaunay graph package of CGAL computes in fact the *segment Delaunay graph* (SDG) under the L_2 distance. The package also provides drawing functions that can convert each edge of the Delaunay graph to the corresponding dual edge of the Voronoi diagram and this is how it is possible to draw the Voronoi diagram.

The SDG L_2 package contains two algorithm template classes (see Figure 4.6) to construct the SDG.

1. The *segment Delaunay graph* class Segment_Delaunay_graph_2 (abbreviation: SDGL2) is derived from a triangulation class (from the 2D Triangulation package of CGAL). Among other things, it contains the functionality to maintain and update the arrangement of the input sites. It also contains functions to construct duals of edges of the SDG, i.e., edges of the Voronoi diagram.

2. The *segment Delaunay (graph) hierarchy* class Segment_Delaunay_graph_hierarchy_2 (abbreviation: SDH) is derived from the SDG class. It builds a hierarchy of SDGs and uses it to achieve faster insertion of a new site in the segment Delaunay graph. This is an implementation with better worst-case complexity than the SDG class (for details, see [24, 44]).

Both template classes have a mandatory template argument (denoted by GT in Figure 4.6), that must be instantiated with a geometric traits class, which contains geometric predicates related to the L_2 diagram (like the in-circle test).

1. The geometric traits class, which provides the basic geometric objects, geometric predicates and constructions involved in the algorithm.

2. The segment Delaunay graph data structure, which stores the planar subdivision generated by the algorithm. The *Triangulation_data_structure_2* in CGAL is the default value for the corresponding template parameter.

The predicates required for the computation of the segment Voronoi diagram can be computationally intensive [13, 39, 40], depending on the input. This package uses both geometric and arithmetic filtering for fast and robust computation. The requirements of this traits class are elaborated in the CGAL SegmentDelaunayGraphTraits_2 concept [43]. There are four different geometric traits implementations, (a) supporting intersections or not and (b) using a user-supplied filtering kernel or not:

Segment_Delaunay_graph_traits_2,

Segment_Delaunay_graph_traits_without_intersections_2,

Segment_Delaunay_graph_filtered_traits_2, and

Segment_Delaunay_graph_filtered_traits_without_intersections_2.

4.3 L_∞ segment Voronoi diagram in CGAL

The use of traits classes as a template parameter allows the customization of the behavior of algorithms without changing implementations. We have written the traits classes that define basic predicates and basic constructions needed for the construction of the segment Voronoi diagram in L_∞. Although algebraically the predicates in the traits class are simpler in L_∞ than in L_2, they are non-trivial as there are many cases, which need a careful analysis before implementation.

4.3.1 Design and relation with SDG L_2

In this section, we explain how we implement the segment Delaunay graph under the L_∞ distance (SDG L_∞) in CGAL, trying to reuse as much code as possible from the existing SDG L_2 package of CGAL. Ideally, we would like the situation to be as follows: We only write a geometric traits class containing the L_∞-related predicates (and constructions) and supply it as the GT template argument of the SDG algorithm template classes of Figure 4.6. In any case, the most significant part of the algorithm, like the maintenance of the arrangement of input sites and the high-level incremental construction of the Delaunay graph is the same under both the L_2 and the L_∞ distance. Unfortunately, since the SDG L_2 algorithm classes were not designed with provision for other distances, there is some L_2-specific code in them, the most significant being the code for drawing dual edges for the Voronoi diagram. Fortunately, these hard-coded L_2-specific functions in the algorithms are few; most of the functionality is indeed in the L_2 geometric traits class. To be more specific, the segment Delaunay graph class (SDGL2) contains the L_2-specific code, whereas the segment Delaunay hierarchy class (SDH) does not contain any L_2-specific code, except the fact that it is derived from SDGL2 (see Figure 4.6).

We make the following design decisions related to the existing SDG L_2 implementation.

- We keep the same interface for users of the SDG L_2 package, so that existing user code does not have to be changed.
- We change the existing SDG L_2 code as little as possible.
- Our changes preserve the efficiency of the SDG L_2 algorithms.

Therefore, we implement a few local changes in the code of the SDG L_2 CGAL package. These changes are mostly in the SDGL2 class and are explained later in this section. These changes allow us to implement Segment_Delaunay_graph_Linf_2 (abbreviation: SDGLinf) as a class derived from SDGL2 (see Figure 4.7).

The existing hierarchy class SDH is hard-coded to use only instances of SDGL2 at its levels, we alter SDH so that it has an additional optional template parameter SDGLx (with default value SDGL2), which is the segment Delaunay graph class that is used in every level of the hierarchy (and from which SDH is derived).

In Figure 4.7, the altered classes SDGL2 and SDH are shown with gray, together with the new class SDGLinf. Since SDGLx is an optional parameter with default value SDGL2, there is no change for old user code of the L_2 segment Delaunay hierarchy. By setting SDGLx to SDGLinf in the SDH template, we obtain the segment Delaunay hierarchy under the L_∞ distance, for which we also

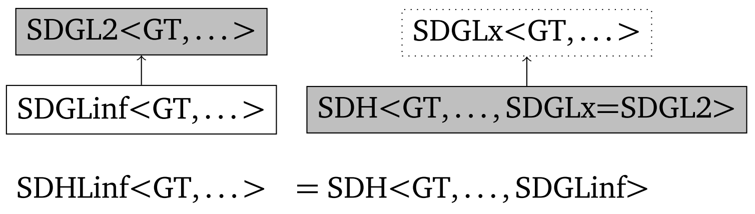


Figure 4.7. New algorithm classes for SDG L_2 and L_∞ packages.

create an alias template class Segment_Delaunay_graph_hierarchy_Linf_2 (abbreviation: SDHLinf) (see Figure 4.7), for easy access to the user.

A user of the SDG L_∞ package has access to two template algorithm classes

Segment_Delaunay_graph_Linf_2<GT,...> and
Segment_Delaunay_graph_hierarchy_Linf_2<GT,...>,

where the GT template argument should be instantiated with one of the following L_∞ geometric traits classes:

Segment_Delaunay_graph_Linf_traits_2,
Segment_Delaunay_graph_Linf_traits_without_intersections_2,
Segment_Delaunay_graph_Linf_filtered_traits_2, and
Segment_Delaunay_graph_Linf_filtered_traits_without_intersections_2,

which are analogous to the corresponding L_2 geometric traits classes.

Apart from the library classes, we also provide a GUI demo, examples, and an ipelet for the L_∞ segment Voronoi diagram. Our package is currently under review for inclusion in the CGAL library.

4.3.2 1-Dimensionalization of L_∞ bisectors

One important difference in the L_∞ setting (in comparison to the L_2 setting) is that in some special non-general position cases the L_∞ bisector between two sites can be bi-dimensional. The choice of considering an input segment as three objects (two end points and an open segment) excludes bi-dimensional bisectors in the L_2 setting, but not always in the L_∞ setting. Since the incremental construction algorithm [48] expects uni-dimensional bisectors, we resort to a

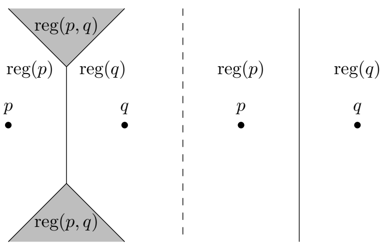


Figure 4.8. The L_∞ bisector between two points with the same y coordinate and its 1-dimensionalization.

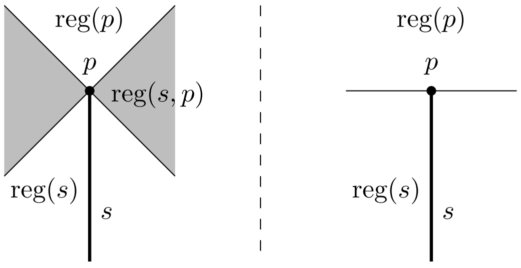


Figure 4.9. The L_∞ bisector between a vertical segment and one of its endpoints and its 1-dimensionalization.

1-*dimensionalization* of these bisectors, by assigning portions of bi-dimensional regions of a bisector to the two sites of the bisector. This way it is also easier to draw the Voronoi diagram. We remark that this simplification of the diagram is acceptable in the VLSI applications, where the L_∞ diagram is employed [69].

If two different points p, q share one coordinate, then their L_∞ bisector is bi-dimensional, as shown in Figure 4.8. In this special case, we 1-dimensionalize the bisector, by taking instead the Euclidean bisector of the two points.

Similarly, the L_∞ bisector between the interior of an axis-parallel segment and one of its endpoints is also bi-dimensional, as shown in Figure 4.9. We 1-dimensionalize this bisector by taking instead the line passing through the endpoint that is perpendicular to the segment, which is also the Euclidean bisector.

4.3.3 The L_∞ parabola and SDGLinf

The L_∞ parabola is the geometric locus of points equidistant under the L_∞ distance from a line ℓ (the directrix) and a (focus) point $p \notin \ell$. In contrast with the standard L_2 parabola, the L_∞ parabola consists of a constant number of linear segments and rays [69].

In the SDGL2 package the input sites are only segments and points, not lines. Therefore, only bounded parabolic arcs appear as edges of the L_2 segment Voronoi diagram and never a complete parabola. See Figure 4.10. On the other hand, unbounded L_∞ parabolic arcs can survive in the corresponding L_∞ diagram. Even complete L_∞ parabolas can survive (see Figure 4.10).

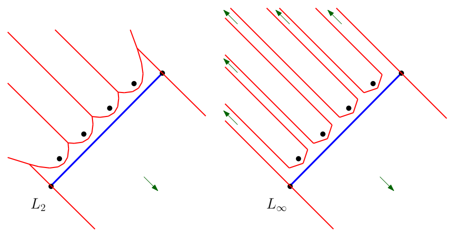


Figure 4.10. Only bounded parabolic arcs survive in the L_2 diagram, whereas even complete L_∞ parabolas can survive in the L_∞ diagram. Arrows point to distinct infinite edges of the diagrams.

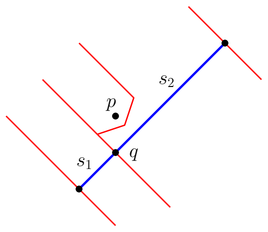


Figure 4.11. The bisector that passes through q touches the parabolic arc at the parabolic arc's portion which is parallel to this bisector.

The existing SDGL2 code is not ready to support the peculiarities of the L_∞ parabolas. For example, the Voronoi region of any segment is expected to have 0, 1, or 2 infinite edges (these are edges with the infinite site s_∞). While this is true in the L_2 setting, it is not true in the L_∞ setting, where the aforementioned number of infinite edges is unbounded in general. For example, in the L_∞ diagram of Figure 4.10, there are six distinct infinite edges neighboring with the region of the open segment.

Several problems may occur when a new point site q is inserted in the interior of an existing segment s. We remark that this operation is needed when, for example, a newly inserted segment crosses an existing segment. The algorithm checks the neighbors of s in the segment Delaunay graph, splits the site of s to two sites s_1 and s_2 and adds the site q to the diagram. In the L_∞ setting this has to be done more carefully than in the L_2 setting. For example, when the site q shares a coordinate with a point p for which there is an L_∞ parabolic arc in the diagram, we have to be careful, because the bisector that passes through q might touch a portion of the L_∞ parabolic arc that is parallel to this bisector (see Figure 4.11). Our solution is to derive SDGLinf from SDGL2 and override some SDGL2 member functions in SDGLinf, in particular the ones that insert a point in the interior of a segment.

4.3.4 Changes in the existing SDGL2 class

Here, we discuss some minimally intrusive changes in the existing SDG L_2 code, so that we can build the SDGLinf class on top of it.

Functions drawing L_2 Voronoi edges are hard-coded in the existing SDG L_2 algorithm class. For the L_∞ design, we decide to keep the algorithm class separate from drawing L_∞ Voronoi edges, and we include the L_∞ Voronoi edge construction functions in the L_∞ geometric traits. We could move the L_2 constructions to the L_2 geometric traits, but we do not do this, because we do not want to change the specification (and documentation) of the L_2 geometric traits (remember our design goals). Instead, we implement the algorithm class to check if the geometric traits contain construction functions for drawing and only then use them, otherwise (if traits are not found) use the hard-coded ones. The check was first implemented with template metaprogramming, using the *Substitution failure is not an error* (SFINAE) principle [80]. Currently the choice of bisector constructions is done by a *tag* (Tag_has_bisector_constructions that can be set to either Tag_true or Tag_false of CGAL) in the geometric trait class. The tag is set to Tag_false for the L_2 traits, and need to set as Tag_true for the L_∞ traits.

Moreover, the existing code has the types (linear segments, rays, lines, and parabolic arcs) of the Voronoi edges hard-coded in the SDGL2 class. Since the L_∞ Voronoi edges are polygonal chains, we also change the code to work with any type of edge.

In the existing L_2 code, when there are two points in the diagram and a third one is inserted, the resulting Delaunay graph construction is based on the *orientation* test for three points p, q, r (i.e., whether the three points make a left, a right turn, or they are collinear), which is very specific to the L_2 case. To make the code work for both L_2 and L_∞, we substitute the orientation test with a call to the vertex conflict predicate from the corresponding L_2 or L_∞ geometric traits class.

4.3.5 The L_∞ geometric traits

In this section, we discuss some issues related to the L_∞ geometric traits. Like in L_2, the traits contain predicates resolving whether a new site conflicts with an existing Voronoi vertex (vertex conflict) or an existing edge (edge conflict, which can be none, partial or complete). These are the predicates needed by any (randomized) incremental construction algorithm [48]. There are also special conflict-like predicates used when a new point site is inserted in the interior of a segment. Remember that, in addition to the predicates, our L_∞ traits also contain functions for constructing L_∞ bisectors that are conditionally selected by the algorithms.

Voronoi edge conflict: Determining if a Voronoi edge is in conflict due to the insertion of a new site is different in L_∞. Let p, q, and r be three consecutive sites on the convex hull in clockwise order. Consider two circles in L_2, C_{pq} (square in L_∞, D_{pq}) touching sites p and q, and C_{qr} (square in L_∞, D_{qr}) touching sites q and r, with their centers at infinity. Also assume C_{pq} and C_{qr} do not contain any other sites. Now we insert a new site t, such that $t \in C_{pq} \cap C_{qr}$. In L_2, insertion of t will force q out of the convex hull. However, in L_∞ this is not always true. If $D_{pq} \cap D_{qr}$ is a bounded rectangle and $t \in D_{pq} \cap D_{qr}$, then q remains on the L_∞ convex hull. Thus the conflict in Voronoi edges between sites p and q and between sites q and r due to insertion of t should be handled differently in L_2 and L_∞. Let us take other simple examples as shown in Figure 4.12, where we show different cases of edge conflicts due to insertion of a point in the Voronoi diagram of a segment site. Figure 4.12 (a) shows that the finite edges between a axis parallel segment site and its end points are in conflict which is similar to L_2 case, Figure 4.12 (b) shows that none of the finite edges between a non-axis parallel segment site and its end points are in conflict but the infinite edge [1] associated with the interior of the segment site is in conflict, Figure 4.12 (c) is same as (b) but the L_∞ parabola is asymmetrical, Figure 4.12 (d) shows that one finite edge is in conflict and the other is not, and Figure 4.12 (e) shows that both the finite edges between a non-axis parallel segment site and its end points are in conflict which is absolutely opposite to case (b) and (c). Depending on the configuration of the sites, we evaluated all possible differences from L_2 metric.

Voronoi vertex conflict: The vertex conflict predicates are also known as in-circle tests (detailed discussion in subsection 4.3.6). The in-circle test in L_2 is analog to an "in-square" test in L_∞. A new site is tested for containment in the minimum shape (circle or axis-parallel square) that touches the sites asso-

[1]An *infinite Voronoi edge* is an edge between the region of a site and the region of the site at infinity.

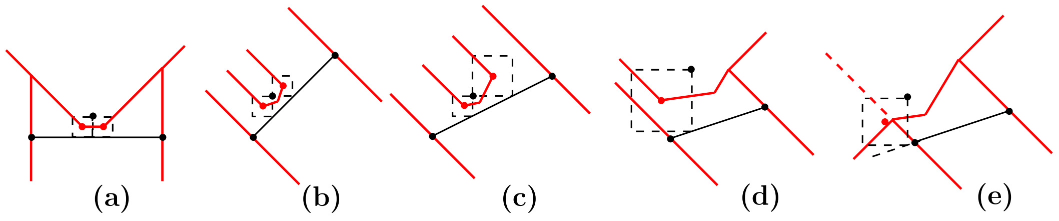


Figure 4.12. Examples showing different L_∞ parabolas which can have different cases of edge conflicts.

ciated with an existing Voronoi vertex. The computation of an in-square type of predicate is numerically simpler than the in-circle predicate. For example, in L_2 the circle that touches three non-collinear points is unique and its center corresponds to the Voronoi vertex. In L_∞, however, the analog axis-parallel square might not be unique (see Figure 4.13). Again our 1-dimensionalization comes to the rescue, since we can define the Voronoi vertex to be the intersection of L_∞ bisectors of these three points and then the square becomes unique. Determining if a Voronoi vertex is in conflict due to insertion of a new site is done differently in L_∞. We determine the Voronoi vertex by computing the intersection between the two suitable bisectors. The suitable bisector is determined by the configuration of sites in question. For example, let p, q and r be three sites such that p is an end point of q, and r is a point site which is not the same as p and is also not an end point of q. Then we choose the bisector between p and q and the bisector between p and r as suitable bisectors to determine the Voronoi vertex v_{pqr} of p, q and r. Now let t be the new site to be inserted. Then we compare distances $d_{v_{pqr},p}$ (distance between v_{pqr} and p) and $d_{v_{pqr},t}$ (distance between v_{pqr} and t) to answer if the Voronoi vertex v_{pqr} is in conflict or not. In L_2, they determine the sign of simplified algebraic expressions. Our initial technique of using intersection of L_∞ bisectors is inefficient compared to having simple arithmetic expressions whose sign would answer the predicates correctly. There are too many cases that should be carefully taken care of in order derive simple expressions for L_∞ in-circle predicates.

4.3.6 The in-circle predicate in the L_∞ metric

The basic predicate in any algorithm for computing the line-segment Voronoi diagram [14, 44, 93] is the Incircle test [13, 39, 40]. Let p, q, and r be the three sites defining a circle $\mathscr{C}$. Then, the Incircle test is defined as a test to determine whether a fourth site is inside $\mathscr{C}$, or outside $\mathscr{C}$, or on the $\mathscr{C}$. In case of the L_∞

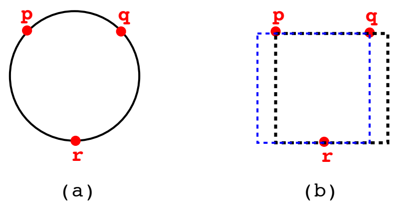


Figure 4.13. (a) p, q, and r define a unique circle, (b) p, q, and r may not define a unique square.

metric the test is done with respect to a square $\mathscr{D}$ defined by p, q, and r.

We consider the square $\mathscr{D}$ defined by the three sites p, q, and r, such that when we walk on $\mathscr{D}$, the sites p, q, and r appear in the counter-clockwise direction. The square $\mathscr{D}$ is bounded by the Voronoi vertex of p, q, and r, $V(p, q, r)$. For a query site t, the in-circle predicate *Incircle(p, q, r, t)* determines the relative position of t with respect to $\mathscr{D}$. The in-circle predicate can return three answers, positive when t is outside the $\mathscr{D}$, negative when t is the inside $\mathscr{D}$, and zero when t is on the $\mathscr{D}$. The in-circle predicate has four arguments, as the first three arguments under circular rotation do not contribute to new configurations, there can be in total eight possible distinct configurations namely, *PPPX, PPSX, PSSX, SSSX* [40], where P stands for point, S stands for line-segment and X stands for either P or S. The in-circle predicates for the Voronoi diagram of line-segments in the Euclidean metric are well studied in [13, 39, 40]. We will give the degree for in-circle predicates for the L_∞ Voronoi diagram of general and axis aligned line-segments separately for seven of the eight configurations. The degree for the *SSSS* configuration is already proved to be 5 [69].

Let s_i, $i = 1, 2, 3$ be the three sites defining a square $\mathscr{D}$. Let (X_1, Y_1) and (X_2, Y_2) denotes the bottom-left corner and the top-right corner of $\mathscr{D}$ respectively. Although the three sites may define more than one square but the test is performed for the one corresponding to a given Voronoi vertex. Let $p = (p_x, p_y)$ and s denotes the query point and the query line-segment respectively. Let e_1 and e_2 denotes the end point of s and l denote the supporting line of s with line equation $ax + by + c = 0$. We need to answer the following predicates:

- Predicate 1: $\text{Incircle}(s_1, s_2, s_3, p) = X_p - X_1 > 0 \wedge X_2 - X_p > 0 \wedge Y_p - Y_1 > 0 \wedge Y_2 - Y_p > 0$
- Predicate 2: $\text{Incircle}(s_1, s_2, s_3, s) = \text{Incircle}(s_1, s_2, s_3, e_1) \vee \text{Incircle}(s_1, s_2, s_3, e_2) \vee (P_3 \wedge P_4)$

Where, $P_3 = aX_1 + bY_1 + c < 0$ and $P_4 = aX_2 + bY_2 + c > 0$ for negative slope of l and $P_3 = aX_1 + bY_2 + c < 0$ and $P_4 = aX_2 + bY_1 + c > 0$ for positive slope of l.

- Predicate 3: For axis aligned line-segments, Incircle$(s_1, s_2, s_3, s) = P_4 \wedge P_5$. For a vertical line-segment s, $P_4 = X_{e_1} - X_1 > 0 \wedge X_2 - X_{e_1} > 0$ and $P_5 = Y_{e_1} - Y_1 > 0$ if $Y_{e_1} - Y_{e_2} > 0$, otherwise, $P_5 = Y_{e_2} - Y_1 > 0$. Similarly, for a horizontal line-segment s, $P_4 = Y_{e_1} - Y_1 > 0 \wedge Y_2 - Y_{e_1} > 0$ and $P_5 = X_{e_1} - X_1 > 0$ if $X_{e_1} - X_{e_2} > 0$, otherwise, $P_5 = X_{e_2} - X_1 > 0$.

We have summarised the degree of Incircle predicates for all different configurations in Table 4.1.

	PPPP	PPPS	PPSP	PPSS	PSSP	PSSS	SSSP	SSSS
General L_2 [13]	4	8	12	24	16	32	32	40
Axis-aligned L_2 [40]	4	6	6	6	4	4	2	2
General L_∞ (this thesis)	1	2	2	3	3	4	4	5
General L_∞ (this thesis) (intersecting line-segments)	-	-	-	-	3	4	4	5
Axis-aligned L_∞ (this thesis)	1	1	1	1	1	1	1	1

Table 4.1. Maximum algebraic degree for eight type of Incircle predicates in the L_2 metric and the L_∞ metric

The PPPX case

Let p, q, and r be three points. In Fig. 4.14, black rectangles are bounded by p, q, and r, and the red dotted square is formed by extending the sides of the black rectangle depending on the configuration of p, q, and r. Vertices of the square can be directly obtained by coordinates of p, q, and r. We assume the degree of the input coordinate of a site to be 1, thus the degree of vertices of the square will also be 1.

From Predicate 1 the Incircle(p, q, r, t) will have degree 1 when t is a point, from Predicate 2 the Incircle(p, q, r, t) will have degree 2 when t is a general line-segment, and from Predicate 3 the Incircle(p, q, r, t) will have degree 1 when t is a axis aligned line-segment.

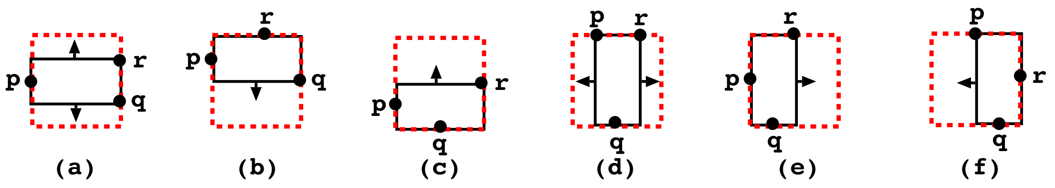


Figure 4.14. Square formed by three point sites depending on their configuration.

The PPSX case

Let p and q be two points and r be a line-segment. Let t be a query point. Then Incircle(p, q, r, t) which is a PPSP case can be reduced to simpler PPPS case Incircle(p, q, t, r) (see Fig. 4.15 for illustration). The point t must lie with in the L formed by axis parallel lines passing through p and q moving in counter clockwise direction from p to q as in Fig. 4.15 (a) and (b). Let this L be denoted by L_{pq}. For the case when t is outside the L_{pq} the Incircle(p, q, r, t) will have degree 1, we do not even need to perform Incircle(p, q, t, r). We simply need to check the position of t in the isothetic staircase formed by p, q and t (see Fig. 4.15 (c) and (d)). When t is in the middle of the staircase, Incircle(p, q, r, t) < 0, otherwise Incircle(p, q, r, t) will be positive or zero. Thus, the PPSP case will have degree atmost 2. When r is axis aligned line-segment the PPSP case is reduced to PPPP case. This is because, in the L_∞ metric a point is equivalent to a horizontal or a vertical line segment.

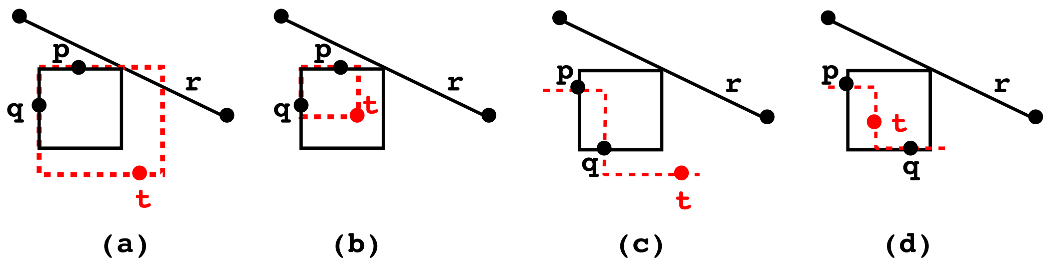


Figure 4.15. Reducing Incircle PPSP to Incircle PPPS, (a) Incircle(p, q, r, t) = Incircle(p, q, t, r) > 0, (b) Incircle(p, q, r, t) = Incircle(p, q, t, r) < 0, (c) Incircle(p, q, r, t) = Incircle(p, q, t, r) > 0, and (d) Incircle(p, q, r, t) = Incircle(p, q, t, r) < 0.

Let us now assume t is a line-segment, and we analyze the PPSS case. The

L_{pq} can be obtained by traversing from p to q isotheticaly in counter clockwise direction. Let the corner of L_{pq} be denoted by C. This C is one of the vertex of $\mathscr{D}$ the other vertex can be obtained by intersection of a horizontal line, or a vertical line, or an ortho-45 line through the corner C with the line-segment r (see Fig. 4.16). We can solve system of linear equations to obtain this other vertex. Let the known vertex c be (X_1, Y_1) (see Fig. 4.16 (b)). Let the other vertex that we will compute be (X_2, Y_2). Let the line equation of r be $ax + by + c = 0$ and we are shooting a ray of slope $+1$ from C to r. Then, we solve the following system of linear equations:

$$A\mathbf{z} = \mathbf{c}, \text{ where } A = \begin{pmatrix} a & b \\ -1 & 1 \end{pmatrix}, \mathbf{z} = \begin{pmatrix} X_2 \\ Y_2 \end{pmatrix}, \text{ and } \mathbf{c} = \begin{pmatrix} Y_1 - X_1 \\ -c \end{pmatrix}$$

Then we obtain the following formula for X_2 and Y_2:
$X_2 = \frac{N_1}{D}$, where $N_1 = -c - b(Y_1 - X_1)$, and $D = a + b$
$Y_2 = \frac{N_2}{D}$, where $N_2 = a(Y_1 - X_1) - c$

Since X_1 and Y_1 is know from input coordinates of p and q, their degree is 1, and thus the degree of X_2 and Y_2 will be 2. Now we substitute coordinates of $\mathscr{D}$ in to predicate 2, and we have the degree of the PPSS case equal to 3.

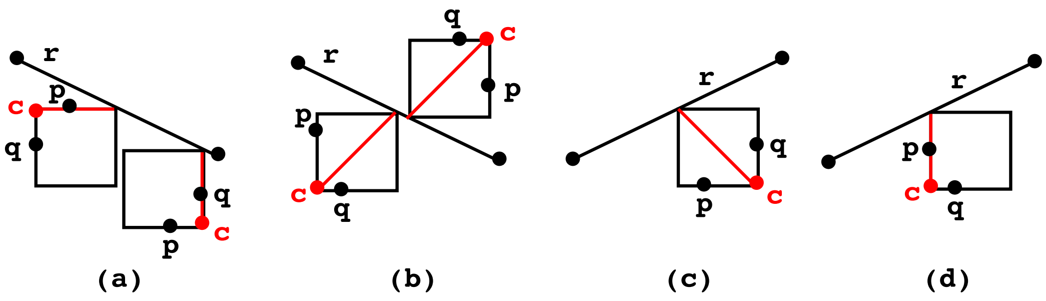


Figure 4.16. Square formed by two points and a line-segment depending on their configuration, (a) r has negative slope and two cases are shown with $c = (X_1, Y_2)$ and $c = (X_2, Y_1)$, (b) r has negative slope and two cases are shown with $c = (X_1, Y_1)$ and $c = (X_2, Y_2)$, (c) r has positive slope and a case is shown with $c = (X_2, Y_1)$, and (d) r has positive slope and a case is shown with $c = (X_1, Y_1)$

The PSSX case

Let p be a point. Let q and r be two line-segments. Let $a_1x + b_1 + c_1 = 0$ and $a_2x + b_2 + c_2 = 0$ be the line equation of line-segments q and r respectively. Depending on the configuration of the sites p, q, and r we can obtain vertices of $\mathscr{D}$ by a similar process as mentioned in the PPSS case (see Fig. 4.17 (a), (b), and (c)). There is one configuration of p, q, and r in which we cannot obtain full coordinates of any vertices of $\mathscr{D}$ directly from coordinates of p, q, and r (see Fig. 4.17 (d)). The only know value in this case is Y_2 which is basically equal to the y coordinate of the point p. For the case shown in Fig. 4.17 (d) we will obtain the the values of X_1, Y_1, and X_2 by solving the following system of linear equations:

$$A_1\mathbf{z_1} = \mathbf{c_1}$$
Where $A_1 = \begin{pmatrix} a_1 & 0 & b_1 \\ 0 & a_2 & b_2 \\ -1 & 1 & 1 \end{pmatrix}$,
$\mathbf{z_1} = \begin{pmatrix} X_1 \\ X_2 \\ Y_1 \end{pmatrix}$, and $\mathbf{c_1} = \begin{pmatrix} -c_1 \\ -c_2 \\ Y_2 \end{pmatrix}$
Then we obtain the following formula for X_1, X_2 and Y_1:
$X_1 = \frac{N_1}{D}$,
Where $N_1 = -c_1a_2 + c_1b_2 - b_1c_2 - a_2b_1Y_2$,
and $D = a_1a_2 - a_1b_2 + b_1a_2$
$X_2 = \frac{N_2}{D}$,
Where $N_2 = -a_1c_2 - b_2a_1Y_2 + c_2 + b_2Y_2$
$Y_1 = \frac{M_1}{D}$,
Where $M_1 = a_1a_2Y_2 + a_1c_2 - c_1a_2$

Since Y_2 is know from the input coordinate of p its degree is 1, and thus the degree of X_1, X_2 and Y_1 will be 3. Now we substitute coordinates of $\mathscr{D}$ in to predicate 2, and we have the degree of the PSSS case equal to 4.

The SSSX case

Let p, q, and r be three line-segments. The Incircle(p, q, r, t) when t is a line-segment is given in [69] and the degree of the SSSS case is proved to be equal to 5. The coordinates of $\mathscr{D}$ is obtained by solving system of linear equations.

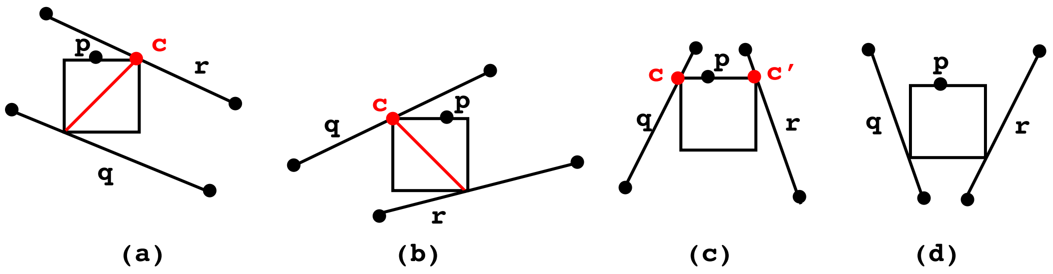


Figure 4.17. Square formed by two line-segments and a point depending on their configuration, (a) q and r both have negative slope and corner c is (X_2, Y_2), (b) q and r both have positive slope and corner c is (X_1, Y_2), (c) q have positive slope and r have negative slope, and two corners of $\mathscr{D}$ are $c = (X_1, Y_2)$ and $c' = (X_2, Y_2)$, and (d) q have negative slope and r have positive slope, and corners of $\mathscr{D}$ are not obtained directly.

Coordinates of $\mathscr{D}$ are equivalent to evaluating algabraic expression of degree 3. When t is point and we substitute the coordinates in Predicate 1, we obtain the degree of the SSSP case equal to 4.

The in-circle test for inputs having intersecting line-segments

Let input line-segments may intersect with each other. The intersecting point of two line-segment never participate in Incircle test in the L_2 metric but it can participate in forming a square with other input sites in the L_∞ metric (see Fig. 4.18). Let $a_1x + b_1y + c_1 = 0$ and $a_2x + b_2y + c_2 = 0$ be equations of two intersecting line-segments such that their intersecting point is (X_1, Y_1) (see Fig. 4.18 (a)). Then X_1 and Y_1 is given by solving the following system of linear equations:

$$A\mathbf{z} = \mathbf{c}, \text{ where } A = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix}, \mathbf{z} = \begin{pmatrix} X_1 \\ Y_1 \end{pmatrix}, \text{ and } \mathbf{c} = \begin{pmatrix} -c_1 \\ -c_2 \end{pmatrix}$$

Then we obtain the following formula for X_1 and Y_1:
$X_1 = \frac{N_1}{D}$, where $N_1 = -c_1b_2 + c_2b_1$, and $D = a_1b_2 - a_2b_1$
$Y_1 = \frac{M_1}{D}$, where $M_1 = -a_1c_2 + a_2c_1$

The square can be defined by two intersecting segments and another input site which may be a point or a segment. We will first analyze the Incircle predicate when the other site contributing in defining the square is an input point p, $p = (X_p, Y_p)$, which is the PSSX case (see Fig. 4.18 (a). The other coordinate of the square is $X_2 = X_p$ and $Y_2 = Y_1 + X_2 - X_1 = \frac{M_1+X_p+N_1}{D}$, when $(X_p - X_1) - (Y_p - Y_1) > 0$ or, $D(X_p - Y_p) + (M_1 - N_1) > 0$. We can compute the Incircle predicates by substituting X_1, Y_1, X_2, and Y_2 in Predicates 1, 2, and 3. For the SSPP case the degree will be 3 from Predicate 1, for the SSPS case when the query segment is non axis aligned the degree will be 4 from Predicate 2, and when the query segment is axis aligned the degree will be 3 from Predicate 3. Now we will consider the other input site contributing in defining the square is a line-segment r (see Fig. 4.18 (b)) with line equation $a_3x + b_3y + c_3 = 0$. The line-segment r will bound one of the other three vertex of the square (X_1, Y_2), or (X_2, Y_1), or (X_2, Y_2). When r bounds (X_1, Y_2), we have, $Y_2 = \frac{-a_3X_1-c_3}{b_3} = \frac{-a_3N_1-c_3D}{b_3D}$. Similar is the case when r bounds (X_2, Y_1). When r bounds (X_2, Y_2), we can have X_2 and Y_2 from the following system of linear equations:

$$A_1\mathbf{z_1} = \mathbf{c_1}, \text{ where } A_1 = \begin{pmatrix} a_3 & b_3 \\ 1 & -1 \end{pmatrix}, \mathbf{z_1} = \begin{pmatrix} X_2 \\ Y_2 \end{pmatrix}, \text{ and } \mathbf{c_1} = \begin{pmatrix} -c_3 \\ X_1 - Y_1 \end{pmatrix}$$

Then we obtain the following formula for X_2 and Y_2:
$X_2 = \frac{N_2}{D_1}$, where $N_2 = c_3 - b_3(X_1 - Y_1) = \frac{Dc_3 - b_3(N_1-M_1)}{D}$, and $D_1 = -a_3 - b_3$
$Y_2 = \frac{M_2}{D_1}$, where $M_2 = \frac{a_3(N_1-M_1)+Dc_3}{D}$

For the SSSP case the degree will be 4 from Predicate 1, for the SSSS case when the query segment is non axis aligned the degree will be 5 from Predicate 2, and when the query segment is axis aligned the degree will be 4 from Predicate 3. Thus the Incircle test for different configurations remains the same (see Table 4.1). For inputs having only axis aligned line-segments the degree of Incircle test in all different configurations also remains the same. This is because of the fact that the intersection point can be directly obtained from the input line-segments without any extra algebraic computations.

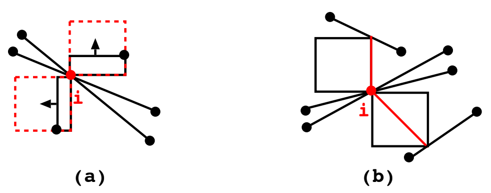


Figure 4.18. Intersecting point i may participate in forming a square with other input site.

Input	Time (seconds) for L_2	Time (seconds) for L_∞
10K points	0.33	0.13
100K points	1.49	1.30
40K segments	0.95	1.04
400K segments	8.32	9.26

Table 4.2. Comparison of runtime between L_2 and L_∞

Speed-up using better in-circle tests

One of the goals for any software is to be efficient (faster). It is a big challenge to make the code faster, and having efficient in-circle tests in code design for Voronoi diagrams, is one of the most important factor towards this goal. This motivated us to do the detailed analysis of the in-circle tests. We have run our code on typical benchmark of CGAL containing around $40K$ line-segments on a MacBook Pro 2.2 GHz Intel i7 with 4 GB RAM. Our code runs in 2.57 seconds, where as the L_2 version runs in 2.70 seconds (given time is average value of 10 runs). We compared runtimes with some more random inputs of points and segments and the results are shown in Table 4.2. Currently, our code is not fully optimized but the run time is already comparable with the L_2 version and ever faster in case of points. Nevertheless, our main goal of having a reliable software package for constructing the Voronoi diagram in the L_∞ metric is achieved.

4.4 Summary

Voronoi diagram of line segment in the L_∞ metric finds various applications in the area of VLSI CAD. The L_∞ metric is well suited for VLSI shapes which consists of line segments that are generally axis parallel or have slopes of ± 1. An open source availability of an implementation for constructing the diagram was long waited.

In this chapter, we presented an implementation of the line-segment Voronoi diagram in the L_∞ metric based on the L_2 Segment Delaunay graph package of CGAL.

The implementation involved three parts: (1) a generalization of the algorithm for existing segment Delaunay graph package to adapt it for the L_∞ segment Delaunay graph, (2) the implementation of the traits classes containing predicates and constructions for building Voronoi diagram of segments and points under the L_∞ metric, and (3) the development of GUI demo, examples and ipelet for the package.

Our goal was to provide a CGAL package. We have submitted the code to the CGAL editorial board along with user and reference manual for its inclusion as a CGAL package. Currently, the code has successfully passed the CGAL test suite for verification of the code. And our package is accepted and integrated to the upcoming version CGAL-4.7 that will be released in september 2015. The lines of code required for our implementation was approximately 17K (see Table A.1. in Appendix A).

We have also used our implementation for pattern analysis of VLSI shapes as described in Chapter 5.

The development of this software has been a joint work with Dr. Panagiotis Cheilaris and Prof. Evanthia Papdopoulou. I would like to specially thank Dr. Panagiotis Cheilaris, who also wrote some portion of the code for this software.

Chapter 5

Application of L_∞ line-segment Voronoi diagram in VLSI pattern analysis problems

"Simplicity is the ultimate sophistication."

Leonardo da Vinci

Integrated circuits (chips) are the heart of all the modern day electronic gadgets. An integrated circuit is basically an electronic circuit made on a semiconductor material. VLSI is acronym for Very Large Scale Integration, is the current level of integrated circuits, that contains millions of transistors in a single chip. The process of printing chips on a semiconductor material is called *lithography*. For printing the chips, lithographers prepare mask patterns. See examples in Figure 5.1). If we see the top view of a VLSI chip, we will see a collection of shapes closely matching the mask patterns. In the latest chips the patterns are mostly rectangular in shape. The two main features that determine proper printing of patterns are width of a pattern, and space between the neighboring patterns. A critical distance of a chip would be the minimum value of width or space. Lower is the critical distance, higher is the difficulty level of printing.

With the increase in miniaturization of current VLSI patterns (lowering of the critical distance), there is a significant rise in printability problems of such patterns, during the photolithography process, and their error-free printing challenges the chip manufacturing industry. The analysis of patterns to find faults or error-prone locations, is of prime importance to the manufacturing

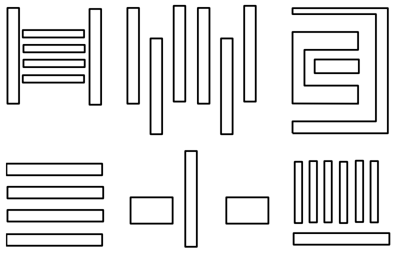


Figure 5.1. Examples of VLSI patterns.

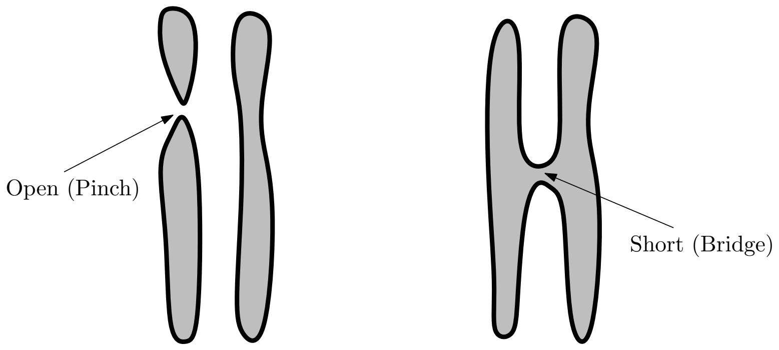


Figure 5.2. Two types of faults that occur during printing VLSI patterns.

process.

There are mainly two kind of faults that can occur during printing: a *pinch* and a *bridge* (see Figure 5.2). A pinch corresponds to an open fault and occurs due to incomplete printing of a shape or due to discontinuity in printing of a shape. A bridge corresponds to a short fault and occurs when two printed shapes are touching each other.

Patterns of interest (POI) : The analysis of a complete layout for finding faults or error-prone locations is difficult and very time-consuming. The printability of a layout is related to the clips or patterns that it contains. Therefore, pattern selection should be done in such a way that analysis of the selected set of patterns should be sufficient to assess the quality of the whole layout. In other words, it is important to identify clips, known as *patterns of interest* (POI), which are more prone to faults. A lower number of POIs (that cover sufficiently the whole layout) allows for a faster but still effective analysis of the layout. However, achieving an optimal set of POIs is a big challenge in this domain. The success of printing a POI is verified by taking several measurements (such as critical distance) on potentially critical areas of *scanning electron microscope* (SEM) images of the printed pattern. Therefore the location of measurement is very important for proper evaluation of a POI.

Hotspot identification and gauge suggestion problem: The measurement location in each pattern is called a *gauge* [86]. The gauge is generally represented by a line in the VLSI pattern around where a critical distance is measured. Therefore, gauge locations must be meaningful, i.e., the critical distance measurement around the gauge location should be the correct measurement for the pattern. Current gauge suggestion techniques are rule-based or they are done manually by VLSI designers. The suggested gauges very often miss the location of critical distance or the location of faults on the clip. The actual location of faults within the clip or POI is known as a *hotspot*. The gauges are the markers within the pattern that help to categorize a pattern as POI and also to locate hotspots within the pattern. With good gauge suggestions, the evaluation of a pattern becomes better, and there will be a possibility of achieving an optimal set of POIs.

In the literature, many variants of hotspot identification methods have been suggested, like, machine learning algorithms [26, 33], image recognition techniques [1, 60, 86], a design based approach[95], pattern matching techniques [79, 91], and topology oriented techniques[77]. For machine learning techniques, the learning time is high and there is a need for already available hotspots to be used as training sets. Image recognition techniques need to go over the entire layout for hotspot detection, which is very time consuming. Pattern matching techniques work on a predefined set of hotspot patterns, and thus, there is a limitation of detecting unseen hotspots. The design based approach has also been observed to be very time consuming as it needs to analyze the entire design. In general, the random nature of layout patterns is difficult to predict in all these methods. Topology oriented pattern extraction techniques [77] can be useful to handle random patterns, however, the existing technique [77] does not claim to be general enough to detect all the hotspots in a layout. In this chapter, we introduce a new topology oriented technique for pattern extraction, based on the geometric information of layout shapes.

The set of POI and the predicted hotspots are of prime importance to address printability problems. After obtaining such a set, designers modify the mask designs with the goal of improving the level of yield. For verification, VLSI designers have developed several models based on POI, including the *model-based optical proximity correction* (MB-OPC) [1, 83, 86]. Hotspots give feedback to the

OPC process for better yield. An optimized MB-OPC demands an optimal set of problematic patterns, but identifying such a set in a time efficient manner is very hard. Currently, reliable OPC models are mostly based on image parameters of the test patterns [1, 83, 86], however, techniques involving image parameters are computationally expensive. Another problem is the automatic identification of problematic patterns. In many cases, the expertise of lithographers and design engineers provides the problematic patterns by manual inspection of the layout. Some new automatic approaches [15, 87] use a combination of parameters. These techniques sample the full spectrum of patterns, and thus, tend to be computationally expensive.

To provide feedback to OPC models, VLSI designers have developed tools to identify optical rule violation in the simulated lithographic patterns. An Optical rule checker (ORC) [57] is a program that encodes and verifies the rules for ideal simulated lithographic patterns. Given the lithographic processing conditions, an ORC run generates markers on the violation of an ORC rule. Therefore, the problematic patterns in the layout are around the ORC markers. These sets of patterns can be feedback to OPC for better yield in manufacturing.

In Chapter 4 we have explained our work on the implementation of the line-segment Voronoi diagram in the L_∞ metric in the CGAL environment. In this chapter we discuss an application in VLSI pattern analysis, in particular, a fast automatic approach to derive a near optimal set of problematic patterns for a layout, using our code for the L_∞ line-segment Voronoi diagram. These sets of patterns can also potentially be used for calibration and verification of MB-OPC. We discuss the potential of using the L_∞ segment Voronoi diagram to identify critical locations in a VLSI pattern and verify the effectiveness, which is, if the identified locations practically corresponds to problematic locations in a VLSI layout or not.

We first find gauge locations, using the line-segment Voronoi diagram of layout shapes, and give priority to the gauges depending upon the shape and proximity information of the design polygons. The gauge locations are then used to extract windows from the design layout. We extract one window per gauge location. The windows contain patterns which are potentially problematic. Finally, we verify the usefulness of the extracted windows by comparing the coverage of the problematic patterns with respect to the ORC generated markers. We observe that the set of patterns extracted by our tool covers all the ORC generated markers for the given layout.

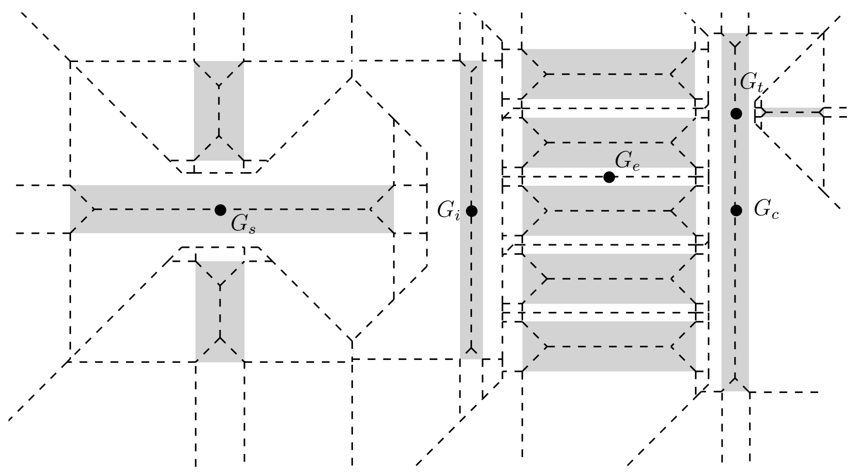


Figure 5.3. Different gauge suggestions.

The rest of this chapter is organized as follows. We describe the gauges and their scoring method in Section 5.1. We describe our method to detect potentially critical locations in the VLSI design layout using the segment Voronoi diagram and then describe our pattern selection procedure in Section 5.2. We discuss our experimental results in Section 5.3.

5.1 Gauge suggestion using the segment Voronoi diagram

We use the L_∞ segment Voronoi diagram to suggest good gauge locations, based on proximity information among the shapes of a pattern, such as the space between shapes and the extent of interaction between neighboring shapes. We suggest five types of gauges, *internal, external, sandwich, comb* and *T*, as illustrated in Figure 5.3. Figure 5.4 illustrates different gauges in a portion of a design layout. Following is the description of the five gauge types.

1. *Internal gauge* (inside a shape), G_i: This gauge lies on the center of a Voronoi edge in the interior of the polygonal shape of minimum width in the pattern (see G_i in Figure 5.3). The position of G_i is a probable location for a pinch, when printing the pattern.

2. *External gauge* (between different neighboring shapes), G_e: This gauge lies on the center of the Voronoi edge between the two shapes that are

closest in the pattern (see G_e in Figure 5.3). The position of G_e is a probable location for the formation of a bridge between the two corresponding shapes, when printing the pattern.

3. *Sandwich gauge„* G_s: This gauge lies on the center of the Voronoi edge inside a polygonal shape P_1 that is “sandwiched” between two other shapes P_2 and P_3 for which the distance between P_2 and P_3 is the minimum in the pattern (see G_s in Figure 5.3). There is a probability of a pinch happening at P_1 around G_s because of the influence of P_2 and P_3.

4. *Comb gauge,* G_c: This gauge lies on the center of the Voronoi edge inside a long polygonal shape, which is the base of the comb and it has close to it and on one side of it a number of polygonal shapes which are the teeth of the comb. We report the gauge for the configuration where the base of the comb shape is closer to the teeth in the pattern (see G_c in Figure 5.3). The position of G_c is dangerous for a pinch, when printing the pattern.

5. *T gauge,* G_t: A comb gauge at a minimum must contain one tooth and a base. We call such a minimal comb gauge a *T gauge* (see G_t in Figure 5.3). The position of G_t is a probable location for a pinch, when printing the pattern.

We introduce a scoring method for gauges, which is used for prioritizing the gauges to determine problematic patterns efficiently. The score associated with a gauge determines its affinity towards the printing problem. It indicates the potential of failure around the location of the gauge, when printing the related pattern.

5.1.1 Scoring method for gauges

For each type of gauge we defined scores as follows:

- Score of an internal gauge: Let P_i be a shape in the design layout. The score of an internal gauge is the minimum width value in P_i (see w in Figure 5.5 (a)). In case there are shapes with the same width, we break ties using the *extension* parameter. The extension parameter is the length of the Voronoi edge associated with an internal gauge. The extension parameter does not change the value of the score; it is just used to change the order of the gauge in the priority list in case of ties. The gauge associated with the longer shape that is having a greater value of associated extension parameter gets more priority and will be ranked higher in the ordered

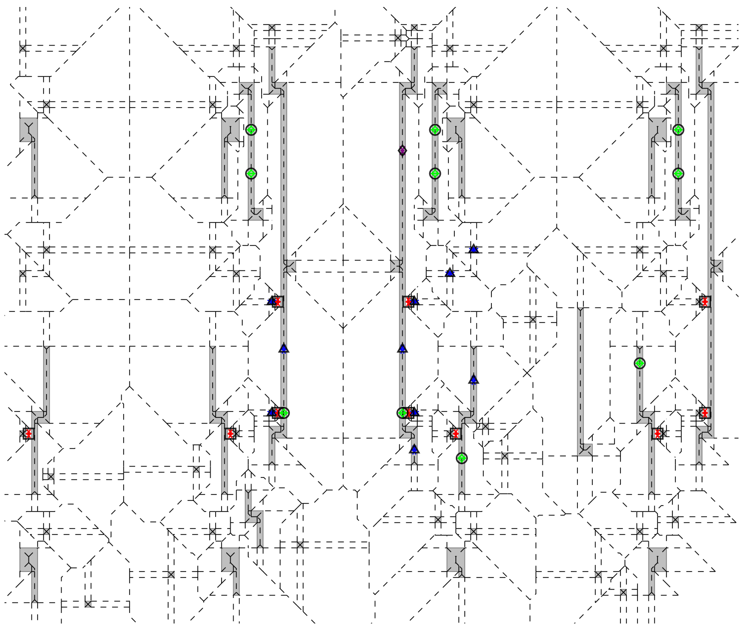


Figure 5.4. Showing different gauge locations in a portion of design layout: shapes in grey are design polygons, Voronoi diagram of design polygons is shown by dashed lines.

list of internal gauges. Lower width implies a thinner and a greater extension implies a longer shape; a thin long shape has more probability to give rise to a pinch.

- Score of an external gauge: Let P_a and P_b be two neighboring shapes in the design layout. The score of an external gauge associated with P_a and P_b is the separating distance (s in the Figure 5.5 (b)) between P_a and P_b. This is encoded by the associated Voronoi edge between P_a and P_b. When there are gauges with the same score, we break ties using the extension parameter. The gauge whose associated Voronoi edge is longer gets higher priority, as the longer edge implies more interaction between the shapes, and thus, higher probability of a bridge.

- Score of a sandwich gauge: Let P_x, P_y, and P_z be three shapes in a design layout such that P_y is sandwiched between P_x and P_z. We define the score for the sandwich gauge equal to the $0.5\times$ d (see Figure 5.5 (c)), where d is the distance between the Voronoi edges E_{xy} and E_{yz} (see Figure 5.5 (c)). If there are (P_x, P_y, P_z) triples with the same score, we use the *overlap* parameter to break ties; a sandwich gauge with a greater overlap

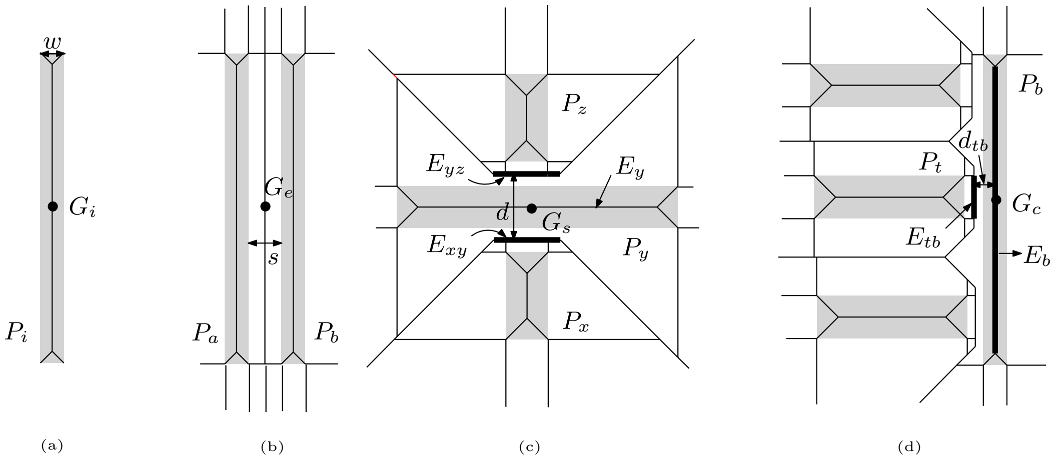


Figure 5.5. An example showing different gauges with their scoring formula: (a) internal gauge with score = w, (b) external gauge with score = s, (c) sandwich gauge with score = $0.5 \times d$, and (d) comb gauge with score = $0.75 \times d_{tb}$, the score of a T gauge is computed similarly with score = $0.80 \times d_{tb}$.

parameter gets higher priority. The overlap parameter is the measure of the length of the overlapping portion of Voronoi edges E_{xy}, E_y, and E_{yz} associated with the sandwich configuration (see Figure 5.5 (c)).

- Score of a comb gauge: For the score of comb gauges, we first define the distance d_{tb} between the tooth (P_t) and the base (P_b) as the distance between the Voronoi edges E_{tb} and E_b (as shown in Figure 5.5 (d)). The Voronoi edge E_{tb} is associated with an edge of the base and an edge of the tooth, and the Voronoi edge E_b is an edge associated with the base of the comb. The score of a comb gauge is then defined as $0.75 \times d_{tb}$. In case, we have comb gauges with same score, we break ties by the measure of overlap between E_{tb} and E_b. We give priority to the comb gauge where tooth has more overlap with the base.

- Score of a T gauge: For T gauges the score is defined as $0.80 \times d_{tb}$. The ties for T gauges are broken in the same way as for comb gauges.

The lower the score of a gauge, the higher is the probability of getting a problematic pattern around that gauge. When gauges of the same type get the same score, we break ties by the extension and overlap parameters described above. A higher value of these parameters gives a higher priority to the gauges. If two different types of gauges have the same score, we break ties using the *context* parameter, which reflects complexity of the pattern associated with the gauge. We have simply considered a hard coded order of context parameter, $C_s > C_c > C_t > C_e > C_i$, where C_i, C_e, C_s, C_c, and C_t is the context parameter of the internal, external, sandwich, comb and T gauges respectively. For example, if a sandwich gauge G_s and an internal gauge G_i have the same score, then G_s has higher priority, because $C_s > C_i$. The scoring method along with the parameters, provide us a priority-based ordered list of gauges. Figure 5.5 illustrates each type of gauge with its scoring formula.

5.2 Pattern selection based on the scoring method

We use the gauge locations as derived in Section 5.1 to extract windows and patterns from the design layout. We feed our pattern selection tool with two inputs (see Figure 5.6): (1) A design layout, (2) A set of *markers* for the design layout. A marker is a region in the design that indicates a problematic area that has high probability of faults. It is generally given in the form of rectangles. For a given design layout, we first obtain the ordered list of gauge locations according to their priority, and we traverse it to extract patterns, one pattern per gauge. Following are the steps of our Voronoi based pattern selection tool:

1. Compute all possible gauge locations in the given layout. Sort the gauges according to the scoring function.

2. For each gauge, consider a window of 5—8 pitches with the gauge location as the center of the window.

3. For each window, check if it covers any marker. Select a window only if it covers some marker that has not already been covered by a previous window; otherwise discard it. A marker is considered covered in three different ways (A,B,C) as defined below.

As we do not desire overlapping windows, we prune any windows that cover markers that have already been covered, and thus, we obtain a set of disjoint windows for a layout. In step 3 we consider a marker to be covered in three different ways: (A) when the geometric center of the marker is strictly inside the corresponding window of a gauge, in which case we say that the gauge covers the marker; (B) when the whole marker rectangle is completely inside the corresponding window of the gauge; and (C) when the marker rectangle overlaps with the gauge window. The results of our experiment for the three different methods (A), (B), and (C) of marker covering are shown in Section

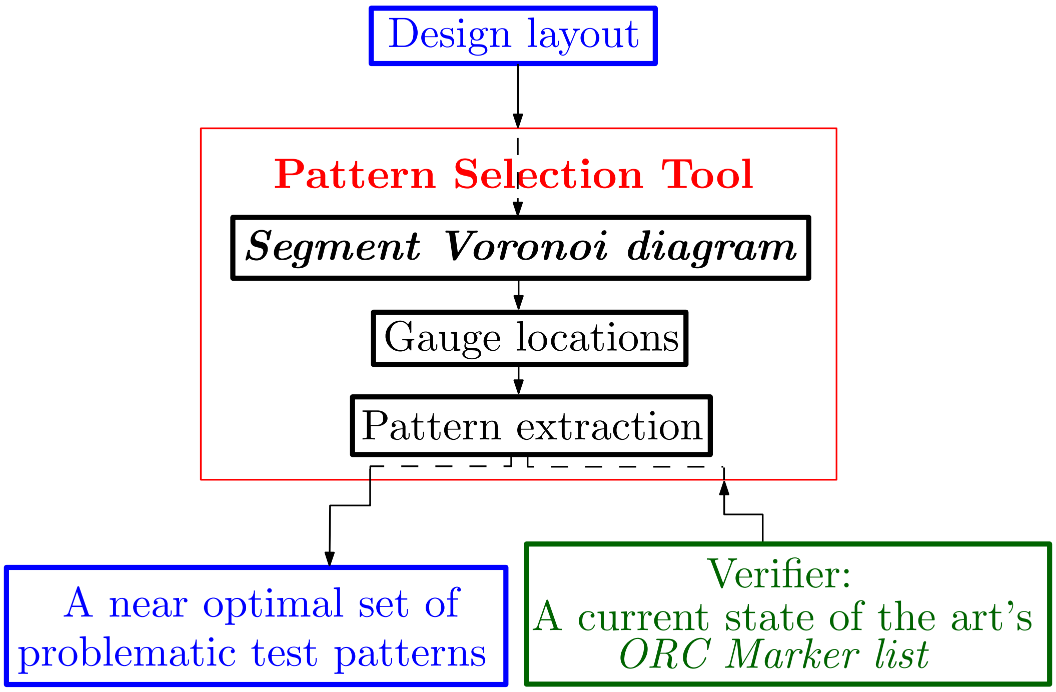


Figure 5.6. Block diagram of flow for the pattern selection using Segment Voronoi diagram.

5.3, Table 5.2, Table 5.3, and Table 5.4 respectively. Note that method (B) may miss many markers because a long or a wide marker may not completely fit inside the windows we considered.

To analyze the quality of our gauges we further investigated the number of windows required to cover all the markers. We tried to put gauge locations at close proximity into one window, by using the following heuristic. The heuristic takes a window size and goes over the layout to cover the markers, while it discards unnecessary windows. The input to our heuristic algorithm is a set of points P, derived by the set of marker centers or the set of gauge locations, and the output is a set of windows, which covers the input set of points. The description of the heuristic algorithm follows:

1. Find the bottommost point in P. Let this point be (b_x, b_y). Construct a window W with $(b_x - 1, b_y - 1)$ as its bottom left corner; remove from P all points that lie inside W.
2. If $P \neq \emptyset$, increment the window counter by 1 and repeat step 1; otherwise report the window counter as the number of windows required to cover all points in the given set.
3. Output the windows generated by step 1.

To further reduce the number of patterns, we apply this heuristic to the obtained gauge locations. The next section provides experimental results.

5.3 Experimental results

We first validated the usefulness of these gauges by experiments on patterns of small size. After getting motivating results, we then performed our experiments on bigger portion of layouts.

5.3.1 On small size patterns

We have performed experiments on a few patterns provided by IBM Zurich Research Laboratory, in order to assess the quality of gauges suggested by our code based on the L_∞ segment Voronoi diagram. These patterns were hand-picked by engineers at IBM and for most of them the existing gauge suggestion is not optimal and misses the critical location in each pattern. Each pattern is a representative of a wide set of patterns with similar behavior. We run experiments on ten patterns: A, B, C, D, E, F, G, H, I, J (Figures 5.7–5.16). For each pattern, we have a figure in which we show three images: (a) in the left, the pattern and the existing gauge suggestion; (b) in the center, the SEM image around the existing suggested gauge and the location of the critical distance measurement with a cyan arrow; (c) in the right, the pattern together with the gauge suggestions provided by our tool based on the L_∞ segment Voronoi diagram. We denote the gauge of each type with a specific symbol at its center and with colored arrows as follows: G_i: $\triangle$, blue; G_e: $\square$, red; G_s: $\circ$, green; G_c: $\diamond$, purple. In many cases (A, C, E, G, H, I), the internal and the sandwich gauge coincide. This is due to the fact that each pattern that we obtained is relatively small and with small variation.

For each pattern we measure the distance in pixels in the corresponding SEM image for each of our suggested gauges and we take the minimum. We show the comparison with the existing measurements in Table 5.1. For patterns A and E, we have essentially the same suggestion as the existing one and therefore the same measurement. For all other patterns, we have an improvement over the existing gauge. Most of the best gauges suggested are either internal or external. In particular, the external gauges suggested for patterns B, D, G, F, I, J capture interactions between different shapes that could be printed too close and improve on the existing suggestions. In pattern C, the vertical internal gauge that we suggest is more critical (the rectangle is thin along this direction) than the horizontal external existing gauge suggestion. In pattern E, we have a gauge G_s suggested by a sandwich configuration (which coincides with the G_i suggestion). In pattern H, we have a gauge G_c suggested by a comb configuration, which allows us to detect a pinch in the SEM image.

We claim the improvement in gauge suggestion for each pattern by inspecting the corresponding SEM image of the pattern. This experiment shows that the L_∞ segment Voronoi diagram can be used effectively to identify potentially critical locations of VLSI layouts.

Pattern	old measurement	type of our gauge	our measurement	improvement
A	17	internal	17	(same) 0%
B	58	external	10	83%
C	52	internal	18	65%
D	31	external	21	32%
E	16	sandwich	16	(same) 0%
F	27	external	18	33%
G	86	external	13	85%
H	35	comb	(pinch) 0	100%
I	47	external	10	79%
J	28	external	8	71%

Table 5.1. Comparison of CD measurement at different gauge locations

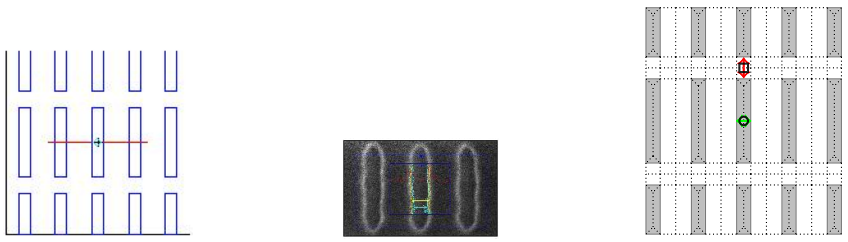


Figure 5.7. Pattern A: our suggestion: G_i (same as existing gauge) coincides with G_s (∘, green)

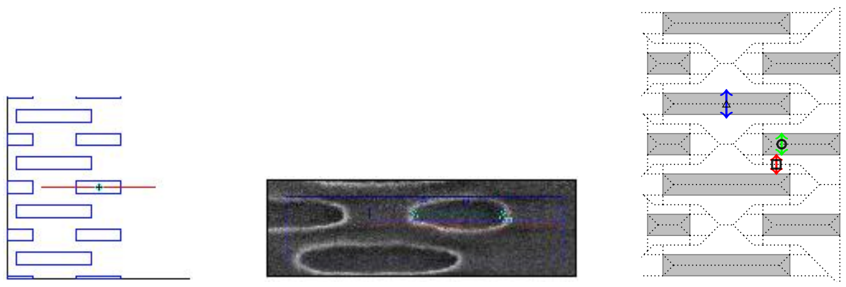


Figure 5.8. Pattern B: our suggestion: G_e (□, red) improves on existing gauge

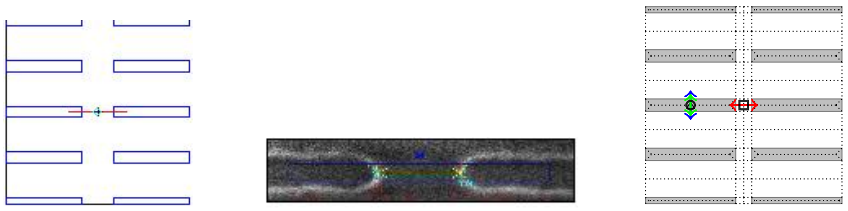


Figure 5.9. Pattern C: our suggestion: G_i coincides with G_s (∘, green), improves on existing gauge

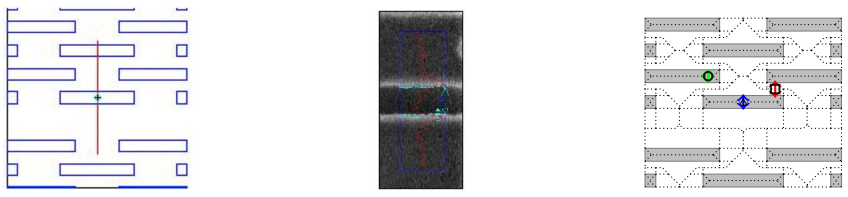


Figure 5.10. Pattern D: our suggestion: G_e ($\square$, red) improves on existing gauge

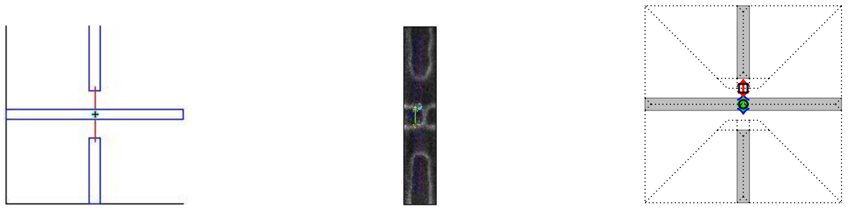


Figure 5.11. Pattern E: our suggestion: G_s ($\circ$, green, same as existing gauge) coincides with G_i.

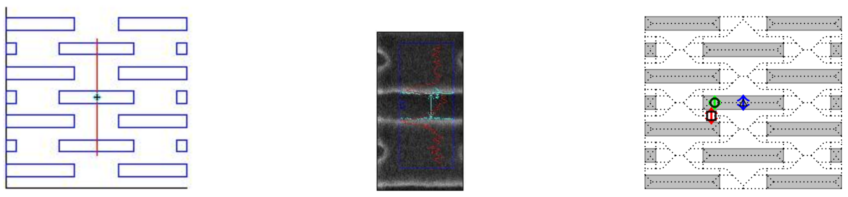


Figure 5.12. Pattern F: our suggestion: G_e ($\square$, red) improves on existing gauge

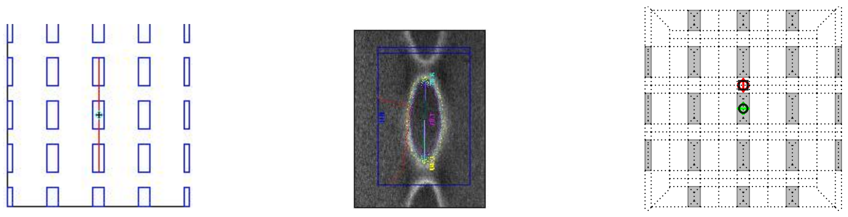


Figure 5.13. Pattern G: our suggestion: G_e ($\square$, red) improves on existing gauge

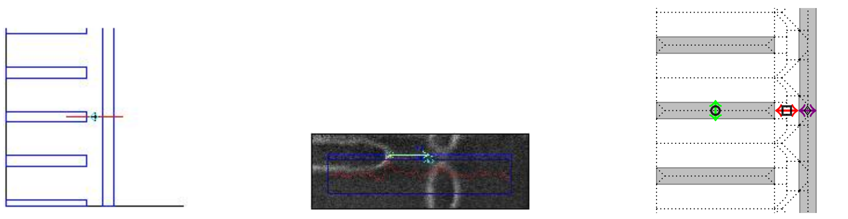


Figure 5.14. Pattern H: our suggestion: G_c ($\diamond$, purple) detects a pinch, improves on existing gauge

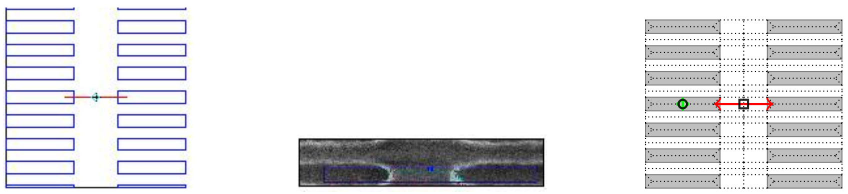


Figure 5.15. Pattern I: our suggestion: G_e (□, red) improves on existing gauge

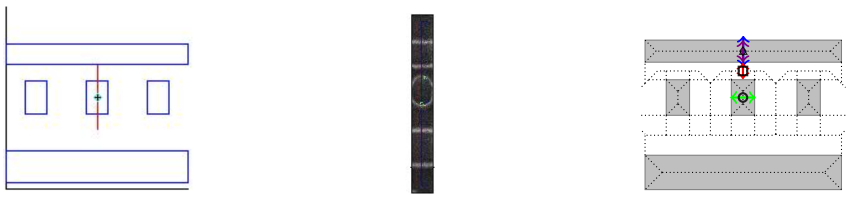


Figure 5.16. Pattern J: our suggestion: G_e (□, red) improves on existing gauge

5.3.2 On bigger portion of a layout

We have done experiments for pattern selection on portion of a 22nm random logic design layout, provided with a state of the art markers for the corresponding layout. The 22nm layout had 38584 design polygons, and 7079 markers. The experiments are executed on a MacBook Pro 2.2 GHz Intel i7 with 4 GB RAM.

Marker coverage

We check the quality of the gauges by counting the number of gauges needed to cover all the markers. We considered four window sizes (in *pitches* × *pitches*), 5 × 5, 6 × 6, 7 × 7, and 8 × 8. We observed that the smaller windows were not able to cover all the markers. As we increased the window size the covering of markers increased. Our results are summarized in Tables 5.2, 5.3, and 5.4, and graphs shown in Figures 5.17, 5.18, and 5.19.

The notation in Tables 5.2, 5.3, and 5.4 are as follows: W_s = window size = *pitches* × *pitches*, M_c = Number of markers covered, G_u = Number of gauges used, r_u = Normalized range of scores of useful gauges (normalized gauge score = $\frac{g_s}{H_s}$, where g_s is the score of a gauge, and H_s is the highest recorded score among all considered gauges), G_f = Number of gauges failed to detect any marker, r_f = Normalized range of scores of failed gauges, p_d = Probability of detection of markers by the provided gauge set = $\frac{M_c}{7079} \times 100$, p_m = Probability that markers will be missed by the given gauge set will be $1-p_d$, u_g = percentage of useful gauges those detect at least one marker (this basically evaluates the scoring function) = $\frac{G_u}{G_u+G_f} \times 100$, and T = Time taken to run the experiment on a MacBook Pro 2.2 GHz Intel i7 with 4 GB RAM.

In Table 5.2, we show results of our experiment, implementing the method (A) of marker covering. Recall that for method (A), a marker is considered covered, if the geometrical center of the marker is within some window. The gauge utilization, that is, the percentage of gauges that successfully covered some marker, increases with the increase in the window size. The larger window has greater chance to cover more marker centers. Windows in our experiments are chosen with priority, based on the priority of the gauges, which in turn depends on shapes and the proximity information of the shapes in the design. The probability of marker coverage increases with the increase in the window size. We

Table 5.2. Covering of marker centers by windows generated by gauge set

W_s (*pitches* × *pitches*)	M_c	G_u	r_u	G_f	r_f	p_d	u_g	T
5 × 5	6968	5708	[0.05 - 1.0]	1638	[0.05 - 0.1]	98.43%	77.70 %	18.208 sec
6 × 6	7070	5306	[0.05 - 0.55]	675	[0.05 - 0.1]	99.87 %	88.71%	17.666 sec
7 × 7	7079	5029	[0.05 - 0.2]	383	[0.05 - 0.1]	100%	92.92%	16.768 sec
8 × 8	7079	4767	[0.05 - 0.16]	354	[0.05 - 0.1]	100%	93.08%	16.056 sec

observe 100% marker coverage for window sizes 7 × 7 and 8 × 8. The run time of the experiment decreases with the increase in the window size. For all the different window sizes the run time is within 20 seconds.

Table 5.3 shows the result for method (B) of marker covering, in which a marker is considered covered, if the marker area is completely within some window. The marker coverage increases with the increase in window size. The gauge utilization first increases with the increase in window size (from 5 × 5 to 7 × 7), and then decreases as we further increase the window size (from 7 × 7 to 8 × 8). This is because a bigger window has potential to cover more markers but we need to check with a larger number of windows, as there are many windows which do not completely contain any marker. In this case also we observe that the probability of marker coverage increases with the increase in window size. We were not able to cover 100% of markers, mainly due to the fact that some markers were very long or wide and were not fitting in within any acceptable window size. The best case was 99.08% for the window size of 8 × 8. For all the different window sizes the run time is within 70 seconds.

Table 5.4 shows the result for method (C) of marker covering, in which a marker is considered covered, if the marker area is overlapping within some window. The probability of marker coverage increases with the increase in window size. We observe 100% marker coverage for the window size of 7 × 7 and 8 × 8. For all the different window sizes the run time is within 4 minutes. The time taken in this case is more compared to the other two cases as the predicate to determine overlap between window and design shapes takes more time that the predicate that determines if a point is inside a window or a rectangle is completely inside a window.

We observe that the window size of 7 × 7 and 8 × 8 gives fairly acceptable results in terms of marker coverage in all cases. All types of gauges have the most critical gauge score of highest priority (0.05). A clear observation from the range of scores for G_u is that, with the increase of window size, there is a

Table 5.3. Covering of marker rectangles by windows generated by gauge set

W_s (*pitches* × *pitches*)	M_c	G_u	r_u	G_f	r_f	p_d	u_g	T
5 × 5	6101	5202	[0.05 - 0.6]	2278	[0.05 - 0.1]	86.18%	69.54%	68.63sec
6 × 6	6711	5263	[0.05 - 0.55]	691	[0.05 - 0.1]	94.8%	88.39 %	61.93sec
7 × 7	6908	5126	[0.05 - 0.4]	334	[0.05 - 0.1]	97.58%	93.88%	59.21sec
8 × 8	7014	4899	[0.05 - 0.2]	370	[0.05 - 0.1]	99.08%	92.97%	57.12sec

Table 5.4. Covering (overlapping) of marker rectangles by windows generated by gauge set

W_s (*pitches* × *pitches*)	M_c	G_u	r_u	G_f	r_f	p_d	u_g	T
5 × 5	7011	5607	[0.05 - 0.6]	1627	[0.05 - 0.1]	99.03%	77.50%	3m 57s
6 × 6	7073	5180	[0.05 - 0.4]	341	[0.05 - 0.1]	99.91%	93.82%	3m 44s
7 × 7	7079	4910	[0.05 - 0.2]	319	[0.05 - 0.1]	100%	93.89%	2m 25s
8 × 8	7079	4686	[0.05 - 0.16]	339	[0.05 - 0.1]	100%	93.25%	1m 28s

decrease in the requirement of low priority gauges. In all cases the range of score for G_f is [0.05 - 0.1], which suggests that the gauge (which belongs to G_f) locations are critical and there is a possibility of finding a problematic pattern around these gauges, although in this specific layout they did not cover any ORC marker. The range of scores for G_f in all cases indicates that the scoring method captures the pattern criticality and is also able to capture more patterns that may be needed to minimize the probability of missing problematic patterns.

The variation of marker coverage with window size for all cases are shown in Figure 5.17. The variation of useful gauges with window size for all cases are shown in Figure 5.18. We compare the runtime of the experiments based on three different ways of marker covering in Figure 5.19.

Gauge distribution

We performed experiments to find out the distribution of gauges covering the markers. Results are reported in Tables 5.5 and 5.6. In Table 5.5, we give detailed results on the marker coverage by the different types of gauges. We observe that all types of gauges are useful, as there are at least some gauges of each type covering some markers. For this particular design layout, the sandwich gauge turned out to cover most of the markers. In Table 5.6, we give the distribution of gauges which fail to cover any marker. We observe that the failed gauges are of the extra or the sandwich type. The internal and T type of gauges have no failure.

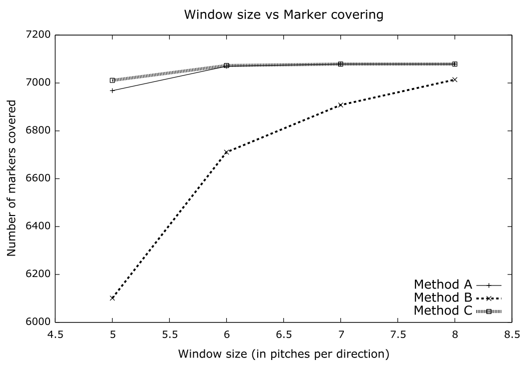


Figure 5.17. Variation of markers covered w.r.t the window size.

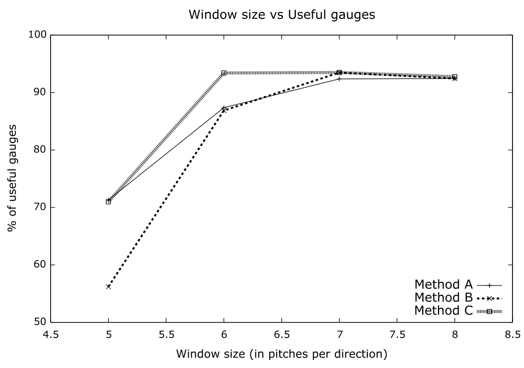
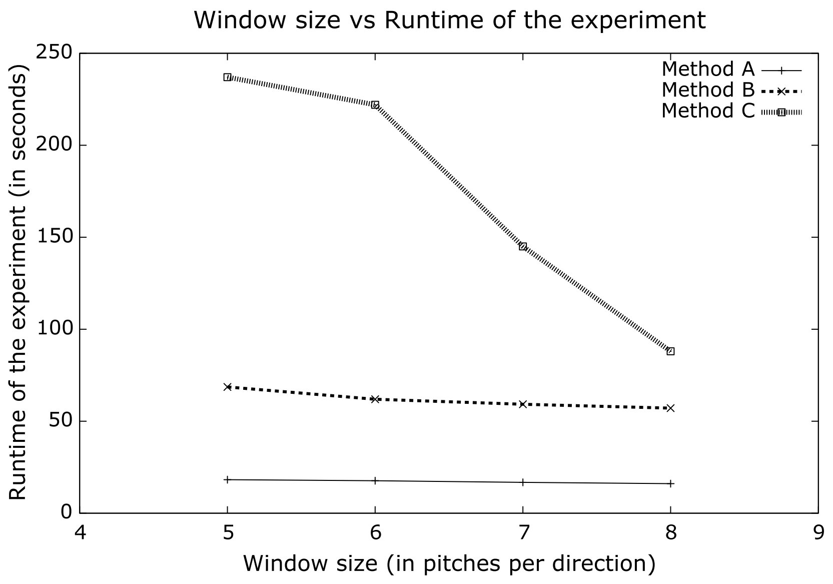


Figure 5.18. Variation of useful gauges w.r.t the window size.



!h

Figure 5.19. Variation of runtimes of the experiments w.r.t the window size.

Table 5.5. Gauge distribution for marker covering: A, B, C are gauge distribution for three different ways of covering markers that is, marker center inside the window, markers are completely inside the window respectively, and markers are overlapping with windows. Notations in the table: r_i, r_e, r_s, and r_t are normalized range of scores of internal, external, sandwich and T gauges respectively. N_i, N_e, N_s, and N_t are number of internal, external, sandwich and T gauges respectively.

	W_s	N_i	r_i	N_e	r_e	N_s	r_s	N_t	r_t	Total
A	5×5	30	[0.2 - 0.4]	38	[0.05 - 1.0]	5388	[0.075 - 0.3]	252	[0.16 - 0.32]	5708
	6×6	6	[0.2 - 0.4]	13	[0.05 - 0.55]	5240	[0.075 - 0.1]	47	[0.16 - 0.16]	5306
	7×7	1	[0.2 - 0.2]	7	[0.05 - 0.1]	5005	[0.075 - 0.1]	16	[0.16 - 0.16]	5029
	8×8	0	[0 - 0]	6	[0.05 - 0.05]	4753	[0.075 - 0.1]	8	[0.16 - 0.16]	4767
B	5×5	57	[0.2 - 0.4]	56	[0.05 - 0.6]	4686	[0.075 - 0.3]	403	[0.16 - 0.16]	5202
	6×6	12	[0.2 - 0.4]	18	[0.05 - 0.55]	5141	[0.075 - 0.2]	92	[0.16 - 0.16]	5263
	7×7	4	[0.2 - 0.4]	9	[0.05 - 0.4]	5078	[0.075 - 0.2]	35	[0.16 - 0.16]	5126
	8×8	1	[0.2 - 0.2]	7	[0.05 - 0.1]	4880	[0.075 - 0.1]	11	[0.16 - 0.16]	4899
C	5×5	15	[0.2 - 0.4]	29	[0.05 - 0.6]	5424	[0.075 - 0.2]	139	[0.16 - 0.32]	5607
	6×6	4	[0.2 - 0.4]	11	[0.05 - 0.4]	5135	[0.075 - 0.1]	30	[0.16 - 0.16]	5180
	7×7	1	[0.2 - 0.2]	7	[0.05 - 0.1]	4888	[0.075 - 0.1]	14	[0.16 - 0.16]	4910
	8×8	0	[0 - 0]	6	[0.05 - 0.05]	4673	[0.075 - 0.1]	7	[0.16 - 0.16]	4686

We have not included information on comb type of gauges in the tables. This is because comb gauges were not used to cover any marker and no comb gauge failed to cover a marker. The range of normalized gauge score for comb gauge is [0.15 - 0.15]. The score range of comb gauges suggests that they are useful. They did not appear in the distribution of useful gauges because they cover markers, which have already been covered by gauges of higher priority (normalized score ≤ 0.15).

Reduction of patterns by a simple heuristic

We tried to obtain a lower bound on the number of windows required to cover all the marker centers, using the simple heuristic algorithm of Section 5.2. For this we took a list of geometrical centers of all the markers as an input to the heuristic. It would be desirable to reach the bound mentioned in the W_m column of Table 5.7 with our gauge locations.

We tried to further reduce the number of patterns using the simple heuristic algorithm of Section 5.2. We took the set of gauge locations corresponding to the set of windows obtained in our experiment using method (A), as an input to the heuristic algorithm. We observe a reduction in the number of patterns by

Table 5.6. Distribution of gauges do not covering any markers.

	W_s	N_i	r_i	N_e	r_e	N_s	r_s	N_t	r_t	Total
A	5×5	0	[0 - 0]	3	[0.05 - 0.05]	1635	[0.1 - 0.1]	0	[0 - 0]	1638
	6×6	0	[0 - 0]	3	[0.05 - 0.05]	672	[0.1 - 0.1]	0	[0 - 0]	675
	7×7	0	[0 - 0]	3	[0.05 - 0.05]	380	[0.1 - 0.1]	0	[0 - 0]	383
	8×8	0	[0 - 0]	3	[0.05 - 0.05]	351	[0.1 - 0.1]	0	[0 - 0]	354
B	5×5	0	[0 - 0]	3	[0.05 - 0.05]	2275	[0.1 - 0.1]	0	[0 - 0]	2278
	6×6	0	[0 - 0]	3	[0.05 - 0.05]	688	[0.1 - 0.1]	0	[0 - 0]	691
	7×7	0	[0 - 0]	3	[0.05 - 0.05]	331	[0.1 - 0.1]	0	[0 - 0]	334
	8×8	0	[0 - 0]	3	[0.05 - 0.05]	367	[0.1 - 0.1]	0	[0 - 0]	370
C	5×5	0	[0 - 0]	3	[0.05 - 0.05]	1624	[0.1 - 0.1]	0	[0 - 0]	1627
	6×6	0	[0 - 0]	3	[0.05 - 0.05]	338	[0.1 - 0.1]	0	[0 - 0]	341
	7×7	0	[0 - 0]	3	[0.05 - 0.05]	316	[0.1 - 0.1]	0	[0 - 0]	319
	8×8	0	[0 - 0]	3	[0.05 - 0.05]	336	[0.1 - 0.1]	0	[0 - 0]	339

Table 5.7. Covering of gauge locations and marker centers by our heuristic algorithm. Notations in the table: W_b = Number of windows before using heuristic, W_a = Number of windows obtained after using heuristic, W_m = Number of windows required by heuristic to cover marker centers, R_w = Reduction in windows from W_b to W_a.

W_s (*pitches* $\times$ *pitches*)	W_b	W_a	R_w	W_m
5×5	5709	4102	28.14%	4442
6×6	5306	3555	33.00%	4146
7×7	5029	3221	35.95%	3934
8×8	4767	3013	36.79%	3783

around 30%. We have not evaluated the usefulness of reduced set of gauges. We report the results in Table 5.7.

Similarity of failure patterns

We compare the patterns which failed in a layout in terms of context and proximity. The patterns extracted by our experiments is based on the score of gauge locations. The score of a gauge location and the type of the gauge encodes the context and proximity information of the problematic location. We get a representative set of patterns (R_g) for a layout by clustering the gauges based on their score and type. Each element is the representative set indicates a group of gauges (or patterns) with same score and type. We consider the set of gauge locations corresponding to the set of patterns obtained in our experiment using method (A) with window size 7×7. We report the results in Table 5.8. For

Table 5.8. Representative gauges Notations in the table: R_g = Representative gauge, S_n = Normalized score, N_g = Number of gauges of same type with same score, $\mathscr{S}$ Similarity percentage.

R_g	S_n	N_g	$\mathscr{S}$
i_1	0.2	1	0.0198%
e_1	0.05	6	0.1193%
e_2	0.1	1	0.0198%
s_1	0.075	1	0.0198%
s_2	0.1	5004	99.50%
t_1	0.16	16	0.3182%

our given layout we have one internal guage representative (i_1), two external gauge representatives (e_1 and e_2), two sandwich gauge representatives (s_1 and s_2), and one t gauge representative. The similarity percentage $\mathscr{S}$ is computed as $\frac{N_g}{\textit{Total gauges}} \times 100$, which indicates percentage of gauges of a type with same score. It can be observed that vast number of gauge locations are identical with respect to their topological context, which effectively reduces the number of patterns for analysis. There are only 6 representative gauges for the 5029 gauge locations in our considered layout.

Similarity measure of two layouts

We compare two layouts and give a similarity measure based on their representative set of gauges. Let L_1 and L_2 be two layouts with set of markers M_1 and M_2 respectively. Let $R_1 = \{P_1^1, P_2^1, \cdots, P_p^1\}$ and $R_2 = \{Q_1^2, Q_2^2, \cdots, Q_q^2\}$ be the set of representative gauges for L_1 and L_2 respectively. Then, L_1 has p representatives and L_2 has q representatives. The total number of gauges in L_1 and L_2 for covering markers is given by $N_t^1 = \sum_{i=1}^{p} |P_i^1|$ and $N_t^2 = \sum_{i=1}^{q} |Q_i^2|$ respectively. Let S_{ij} denotes the set of similar gauges in P_i^1 and Q_j^2 then $S_{ij} = P_i^1 \cap Q_j^2$, where $1 \le i \le p$ and $1 \le j \le q$. We give the percentage similarity measure between two layouts L_1 and L_2 by $\mathbb{S}_{12} = \frac{\sum_{i=1}^{p} \sum_{j=1}^{q} |S_{ij}|}{N_t^1 + N_t^2} \times 100$.

For a sample experiment we obtained L_1 and L_2 by partitioning our given layout in to two parts, and M_1 and M_2 by partitioning the given ORC marker's list in to two parts. The partioning of the given layout is done with respect to a vertical line passing trough a x-coordinate, which is computed as $(x_{min} + x_{max})/2$ (where x_{min} and x_{max} are the minimum and miximum x-coordinate among all the points in the given layout). Similar is the partitioning of the marker's list. We considered method (A) with window size 7×7 for obtaining

Table 5.9. Representative gauges in L_1 and L_2 Notations in the table: $R_g^1 =$ Representative gauge in layout 1, $R_g^2 =$ Representative gauge in layout 2, $S_n =$ Normalized score, $N_g^1 =$ Number of gauges of same type with same score in layout 1, and $N_g^2 =$ Number of gauges of same type with same score in layout 2.

R_g^1	R_g^1	S_n	N_g^1	N_g^1
i_1^1	i_1^2	0.2	0	1
e_1^1	e_1^2	0.05	4	2
s_1^1	s_1^2	0.075	1	0
s_2^1	s_2^2	0.1	2883	1998
t_1^1	t_1^2	0.16	7	9
			$N_t^1 = 2895$	$N_t^2 = 2000$

gauges. We report our results in Table 5.9. The similarity measure $\mathbb{S}_{12}$ for L_1 and L_2 is approximately 82% which is computed from the data provided in Table 5.9. The similarity measure between two layouts will considerably reduce the inclusion of redundant patterns for analysis.

5.4 Summary

In this Chapter, we described problems in VLSI pattern analysis and our approach to address them. We discussed a new method to select a set of problematic patterns, which is based on topological information extracted from the Voronoi diagram of layout shapes. We use the line-segment Voronoi diagram to identify a variety of gauge locations. We also introduced a scoring method for the identified gauge locations which allows to obtain a priority based sorted list of gauges. We then extract one pattern per gauge location. Our method is fast and gives an automatic way to discover the potentially problematic areas when printing VLSI layout patterns. We verified our windows by covering ORC-generated markers. Using our pattern selection tool, we covered 7079 ORC markers for a design layout of 38584 design polygons using nearly 5000 extracted patterns. The patterns extracted covered all the ORC markers. Applying a simple heuristic algorithm we further reduced the number of patterns by approximately 30%. We also measured the similarity between the layouts based on their representative gauges. We also derived a similarity measure between patterns and between layouts with respect to the gauges. The similarity measure helped to obtain representative gauge locations for a layout and thus potentially decrease the number of effective patterns for analysis. The reduction is approximately by 99.8% for the layout considered in our experiment.

We believe that our line-segment Voronoi based tool for pattern analysis can be enhanced to address optical proximity correction (OPC) problem, which is currently highly rated important topic of research in the domain of design for manufacturability in VLSI CAD. OPC is used in lithography to increase the achievable resolution as well as to increase the layout-to-wafer fidelity for integrated circuit (IC) manufacturing. More specifically, the goal of OPC is to find a mask design such that the final pattern remaining after complete lithography process is as close as possible to the desired pattern on the wafer. For this, OPC must be able to reliably identify the possible problematic patterns causing failure of the design. The main reason to support our case is the ability of the tool to generate a great variety of gauges that has potential to extract topology and context based interesting features of patterns.

The main requirement of OPC is to have a clear classification of good, possibly faulty and faulty patterns. Our tool can generate a wide range of gauges, and with the scoring method we are able to prioritize them. We can incorporate a set of rules for optical error checks in the patterns specified by the chip designers and lithographers, in our scoring method. Then we should be able to partition our prioritized set of gauges into three different sets of good, fault prone and faulty patterns. Nevertheless, the choice of threshold of scores for partitioning can be a tricky process and requires an extensive experimentation work. This lays a foundation for future work to obtain a set of patterns for calibration and verification of MB-OPC (model based optical proximity correction).

The development of this application has been a joint work with Maria Gabrani of IBM Research Zurich, Dr. Panagiotis Cheilaris, and Prof. Evanthia Papdopoulou.

Chapter 6

Conclusion

"Your purpose in life is to find your purpose
and give your whole heart and soul to it."

Gautama Buddha

Our research focused on three major aspects in computational geometry, that is, theory, implementation, and application.

In the theoretical front, we described the structural properties of the farthest line-segment Voronoi diagram in the general L_p metric, $1 \leq p \leq \infty$. In particular, we introduced the farthest line-segment hull and its Gaussian map. The farthest line-segment hull is a closed polygonal curve that fully characterizes the faces of the farthest line-segment Voronoi diagram.

We gave improved structural bounds for the farthest line-segment Voronoi diagram in the Euclidean metric. We proved that the total number of faces of the farthest line-segment Voronoi diagram of n arbitrary line-segments is at most $6n-6$ which improved the existing bound $8n+4$, and we also showed a corresponding lower bound equal to $5n-6$ which improved the existing bound $4n-4$. In the L_∞ metric, we proved that the farthest Voronoi diagram has at most $n+8$ faces and this is tight. For non-crossing line-segments this number is eight. We also showed that a single input line-segment can have at most 5 faces, and this is tights.

We presented algorithms to construct the farthest line-segment hull. We adapted some of the standard Convex hull techniques to construct the farthest-line segment hull. This implied that the farthest line-segment hull for n line segments can be constructed in $O(n \log n) time$ and also in $O(n \log h)$ time by an output sensitive algorithm, where h is the size of the farthest line-segment hull. This unified the techniques to construct the farthest site Voronoi diagram of points and line segments.

In the L_∞ (L_1) metric, provided the farthest line-segment hull, the farthest Voronoi diagram of line segments can be computed in $O(h)$ time.

We implemented the L_∞ Voronoi diagram of points and line-segments in the CGAL environment, in a way that is robust, correct, extendable, flexible, reusable and easy to use. We also discussed all the in-square predicates, which can make the implementation faster. The open source contribution increases possibilities of its usage in practice. Our code is accepted and integrated as a package in the upcoming CGAL-4.7 version and will be released in September 2015.

In the application front, we discussed a new method to identify the gauges in a VLSI layout and select a set of problematic patterns, which is based on topological information extracted from the Voronoi diagram of the layout shapes. Our method is fast and gives an automatic way to discover the potentially problematic areas when printing VLSI layout patterns. We verified our windows by covering ORC-generated markers. Our tool helps in analyzing the printability problem in the VLSI domain. The variety of gauges has a potential to extract topology and context based interesting features of patterns which motivates the future work on improving the tool to facilitate MB-OPC in terms of reliability and time.

6.1 Future directions

"The future depends on what you do today."

Mahatma Ghandhi

In this section we discuss some future directions of research from this dissertation. In particular, we mention possibilities of research in farthest line-segment Voronoi diagram, higher order line-segment Voronoi diagram and its implementation in CGAL, and also about utility of line-segment Voronoi diagram in OPC modelling.

Closing the bound on number of faces of farthest line-segment Voronoi diagram:

The lower bound and the upper bound on the number of faces of farthest line-segment Voronoi diagram are $5n-6$, and $6n-6$ respectively. It would be interesting to close the gap. We believe the tight bound is $5n-6$.

Algorithm for the L_∞ k-order Voronoi diagram for points and segments:

As mentioned in Chapter 2, there is an iterative approach for computing order-k Voronoi diagrams which computes the diagram iteratively from order 1 to order $n-1$. Iterative approaches are good for small values of k. For large k the approach is not efficient [18]. Existing iterative algorithms do not do justice for *higher* order ($k > n/2$, where n is the number of input points) diagrams. Our goal is to develop an algorithm which can compute the higher order Voronoi diagram iteratively from order $n-1$ to order 1, targeting an efficient iterative construction for large k. First step is to derive the algorithm for points and axis-parallel segments in the L_∞ metric, and then generalize for arbitrary line segments. The goal would be to compute the *k-nearest neighbor Delaunay graph* [53] iteratively from order 1 to order $n-1$ or iteratively from order $n-1$ to order 1 depending on the size of k. Once the k-nearest neighbor Delaunay graph is constructed, we can also construct the order-k Voronoi diagram.

CGAL implementation of L_∞ higher order line-segment Voronoi diagram:

The second order segment Voronoi diagram has shown to be very useful in VLSI applications. Since, we already have the first order Voronoi, we can go for an iterative approach to construct the second order, and consequently higher order diagrams can also be computed. We have some of the ingredients for the implementation, like the in-circle predicates, point location structures, triangulation data structure for storing the diagram. A clear design of the algorithm and its implementation in CGAL can be an interesting future research work.

CGAL implementation of a plane sweep algorithm for L_∞ line-segment Voronoi diagrams:

It will be interesting to investigate a CGAL implementation of a plane sweep algorithm for computing L_∞ Voronoi diagrams for points and segments. In Figure 6.1, we show a high level view of the components involved in the sweep-line algorithm. We have already implemented the component geometric traits that involves geometric objects such as point, segment, line, ray, polygon and operations (predicates and constructions) on geometric objects. The topology traits provide a data structure to update and store the segment Voronoi diagram computed by the plane sweep algorithm.

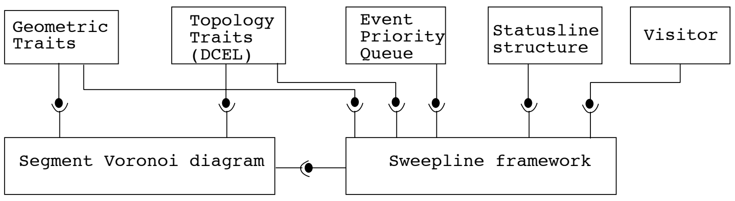


Figure 6.1. High level view of the components in the implementation.

What remains is the modification and adaptation of the topology traits provided in the CGAL packages. The components event queue and the status-line structure are both balanced binary search trees, and are provided in C++ STL. The visitor component needs to be implemented to provide call backs at strategic points of the sweep-line algorithm. The visitor component will be used to provide output of the sweep-line algorithm. The component sweep-line framework is only meant for a sweep-line algorithm to construct L_∞ segment Voronoi diagrams and is not a general framework for other sweep-line algorithms. Investigation of the *Kinetic framework* and *Kinetic Data structures* provided in CGAL to check for their suitability to have a plane sweep construction of Voronoi diagrams is also interesting. Although completing such a package is by far a non-trivial task and could be an important research work.

Facilitating the OPC modeling by using the line-segment Voronoi diagram:
As discussed in Chapter 5, printability problem of VLSI shapes appear largely due to interactions between neighboring shapes. Proximity among VLSI shapes

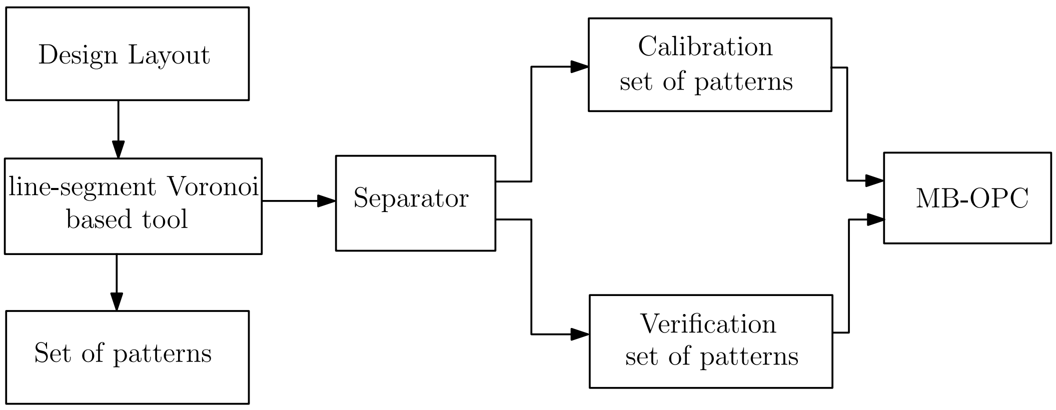

Design Layout
line-segment Voronoi based tool
Set of patterns
Separator
Calibration set of patterns
Verification set of patterns
MB-OPC

Figure 6.2. High level block diagram for supporting MB-OPC.

is a key factor in variability sensitivity (deviation of a shape from desired form during printing) and failure susceptibility. It is important to identify the problematic locations in the VLSI layout for correct calibration and verification of the VLSI layout. The variety of gauges that we identify using our Voronoi tool has a potential to extract topology and context based interesting features of patterns. This lays a foundation for this future work to obtain a set of patterns for calibration and verification of MB-OPC.

A high level description of our idea is shown in Figure 6.2. The block of design layout in the form of a file containing all the design polygon with its coordinates is the input to our tool. The heart of the the tool is the line-segment Voronoi diagram which will feed in the design polygons, and reports a set of pattern based prioritized by context and proximity information of a pattern. The set of patterns obtained from the Voronoi based tool is then passed to the separator block. The separator block must have a formula or rule to divide the set of patterns in to two parts the calibration set and the verification set. The calibration set contains the non faulty patterns, where as the verification set contains the faulty ones. Finally the calibration and verification set of patterns are feed to the OPC tool.

Appendix A

CGAL

1. The trait requirements for L_∞ segment Voronoi diagrams, that we have already developed are the following (For the complete set of trait requirements and their reference manual, please see the CGAL Manual 4.6. [90]):

   (a) *SegmentDelaunayGraphLinfTraits_2:: Intersections tag*
   Indicates if the intersecting segments are supported or not. Although for VLSI application the input segments are non intersecting, but we support intersecting segments also. This indicator helps in geometric filtering.

   (b) *SegmentDelaunayGraphLinfTraits_2:: Site_2*
   The sites are point or segments in 2-d.

   (c) *SegmentDelaunayGraphLinfTraits_2:: RT*
   A type for the ring number type for the arithmetic. There is also FT, a field number type which supports square-root and divisions. For L_∞, the arithmetic is simpler and RT is sufficient.

   (d) *SegmentDelaunayGraphLinfTraits_2:: Object_2*
   A type representing different types of objects in two dimensions, namely: Point_2, Site_2, Line_2, Ray_2 and Segment_2. No parabolic arcs are required in L_∞.

   (e) *SegmentDelaunayGraphLinfTraits_2:: Construct_svd_vertex_2*
   A constructor for a point of the L_∞ segment Voronoi diagram equidistant from three sites. Must provide *Point_2 operator()(Site_2 s1, Site_2 s2, Site_2 s3)*, which constructs a point equidistant from the sites $s1$, $s2$ and $s3$. Which is basically center of the minimum enclosing square.

(f) *SegmentDelaunayGraphLinfTraits_2:: Orientation_2*
Must provide *Orientation operator()(Site_2 s1, Site_2 s2, Site_2 s3)*, which performs the usual orientation test for three points $s1$, $s2$ and $s3$.

(g) *SegmentDelaunayGraphLinfTraits_2:: Oriented_side_of_bisector_2*
Must provide *Oriented_side operator()(Site_2 s1, Site_2 s2, Point_2 p)*, which returns the oriented side of the bisector of $s1$ and $s2$ that contains p. Returns ON POSITIVE SIDE if p is closer to $s1$ than $s2$; returns ON NEGATIVE SIDE if p lies closer to $s2$ than $s1$; returns ON ORIENTED BOUNDARY if p lies on the bisector of $s1$ and $s2$.

(h) *SegmentDelaunayGraphLinfTraits_2:: oriented_side_of_square*
Must provide *Oriented_side operator()(Site_2 s1, Site_2 s2, Site_2 s3, Point_2 p)*, which returns the oriented side of the minimum square formed by $s1$, $s2$, and $s3$ containing point p.

(i) *SegmentDelaunayGraphLinfTraits_2::bounded_side_of_square*
Must provide *Bounded_side operator()(Site_2 s1, Site_2 s2, Site_2 s3, Point_2 p)*, which returns the bounded side of the minimum square formed by $s1$, $s2$, and $s3$ containing point p.

(j) *SegmentDelaunayGraphLinfTraits_2:: Vertex_conflict_2*
Must provide *Sign operator()(Site_2 s1, Site_2 s2, Site_2 s3, Site_2 q)*, which returns the sign of the distance of q from the Voronoi square of $s1$, $s2$, $s3$. Must provide *Sign operator()(Site_2 s1, Site_2 s2, Site_2 q)*, which returns the sign of the distance of q from the degenerate Voronoi square of $s1$, $s2$, with its center at infinity.

(k) *SegmentDelaunayGraphLinfTraits_2:: Finite_edge_interior_conflict_2*
Must provide *bool operator()(Site_2 s1, Site_2 s2, Site_2 s3, Site_2 s4, Site_2 q, Sign sgn)*. The sites $s1$, $s2$, $s3$ and $s4$ define a Voronoi edge that lies on the bisector of $s1$ and $s2$ and has as endpoints the Voronoi vertices defined by the triplets $s1$, $s2$, $s3$ and $s1$, $s4$ and $s2$. The sign *sgn* is the common sign of the distance of the site q from the Voronoi square of the triplets $s1$, $s2$, $s3$ and $s1$, $s4$ and $s2$. In case that *sgn* is equal to NEGATIVE, the predicate returns true if and only if the entire Voronoi edge is in conflict with q. If *sgn* is equal to POSITIVE or ZERO, the predicate returns false if and only if q is not in conflict with the Voronoi edge. Must also provide *bool operator()(Site_2 s1, Site_2 s2, Site_2 s3, Site_2 q, Sign sgn)*. The sites $s1$, $s2$, $s3$ and the site at infinity $s\infty$ define a Voronoi edge that lies on the bisector of $s1$ and

$s2$ and has as endpoints the Voronoi vertices v_{123} and $v_{1\infty 2}$ defined by the triplets $s1, s2, s3$ and $s1, s\infty$ and $s2$ (the second vertex is actually at infinity). The sign *sgn* is the common sign of the distance of the site q from the two Voronoi squares centered at the Voronoi vertices v_{123} and $v_{1\infty 2}$. Must finally provide *bool operator()(Site_2 s1, Site_2 s2, Site_2 q, Sign sgn)*. The sites $s1$, $s2$ and the site at infinity $s\infty$ define a Voronoi edge that lies on the bisector of $v_{12\infty}$ and $v_{1\infty 2}$ s1 and s2 and has as endpoints the Voronoi vertices defined by the triplets $s1$, $s2$, $s\infty$ and $s1$, $s\infty$ and $s2$ (both vertices are actually at infinity). The sign *sgn* denotes the common sign of the distance of the site q from the Voronoi squares centered at $v_{12\infty}$ and $v_{1\infty 2}$.

(l) *SegmentDelaunayGraphLinfTraits_2:: Infinite_edge_interior_conflict_2* Must provide *bool operator()(Site_2 s1, Site_2 s2, Site_2 s3, Site_2 q, Sign sgn)*. The sites $s\infty$, $s1$, $s2$ and $s3$ define a Voronoi edge that lies on the bisector of $s\infty$ and $s1$ and has as endpoints the Voronoi vertices $v_{\infty 12}$ and $v_{\infty 31}$ defined by the triplets $s\infty$, $s1$, $s2$ and $s\infty$, $s3$ and $s1$. The sign *sgn* is the common sign of the distances of q from the Voronoi squares centered at the vertices $v_{\infty 12}$ and $v_{\infty 31}$. If *sgn* is NEGATIVE, the predicate returns true if and only if the entire Voronoi edge is in conflict with q. If *sgn* is POSITIVE or ZERO, the predicate returns false if and only if q is not in conflict with the Voronoi edge.

Lines of code	Comment lines	Filename
96	35	*Orientation_Linf_2.h*
530	126	*Polychain_2.h*
11	1	*basic.h*
1466	197	*Basic_predicates_C2.h*
529	105	*Bisector_Linf.h*
892	174	*Constructions_C2.h*
273	118	*Filtered_traits_base_2.h*
1036	128	*Finite_edge_interior_conflict_C2.h*
496	85	*Infinite_edge_interior_conflict_C2.h*
135	8	*Orientation_Linf_C2.h*
349	59	*Oriented_side_C2.h*
248	30	*Oriented_side_of_bisector_C2.h*
19	1	*Predicates_C2.h*
624	42	*Segment_Delaunay_graph_Linf_2_impl.h*
839	185	*Segment_Delaunay_graph_Linf_hierarchy_2_impl.h*
83	38	*Traits_base_2.h*
1034	186	*Vertex_conflict_C2.h*
36	23	*Voronoi_vertex_C2.h*
3169	253	*Voronoi_vertex_ring_C2.h*
3533	608	*Voronoi_vertex_sqrt_field_new_C2.h*
288	41	*Segment_Delaunay_graph_Linf_2.h*
189	15	*Segment_Delaunay_graph_Linf_filtered_traits_2.h*
45	4	*Segment_Delaunay_graph_Linf_hierarchy_2.h*
106	15	*Segment_Delaunay_graph_Linf_traits_2.h*
495	35	*Side_of_bounded_square_2.h*
118	17	*Side_of_oriented_square_2.h*
17252	2694	*Total*

Table A.1. Lines of code used for implementing the Voronoi diagram of line segments in the max-norm

Bibliography

[1] A. Abdo and R. Viswanathan. The feasibility of using image parameters for test pattern selection during OPC model calibration. In *Proc. SPIE*, volume 7640, pages 76401E–76401E–11, 2010.

[2] P. K. Agarwal, M. De Berg, J. Matousek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM journal on computing*, 27(3):654–667, 1998.

[3] A. Aggarwal, L. J. Guibas, and P. W. Saxe, J.and Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4(1):591–604, 1989.

[4] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *COCOON*, pages 117–126. Springer-Verlag, 1996.

[5] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[6] F. Aurenhammer, R. L. S. Drysdale, and H. Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006.

[7] F. Aurenhammer, R. Klein, and D. T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[8] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *International Journal of Computational Geometry & Applications*, 2(4):363–381, 1992.

[9] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[10] C. Bohler, C. H. Liu, E. Papadopoulou, and M. Zavershynskyi. A randomized divide and conquer algorithm for higher-order abstract Voronoi diagrams. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 27–37, 2014.

[11] J. D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete & Computational Geometry*, 8(1):51–71, 1992.

[12] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1):25–47, 2001.

[13] C. Burnikel. *Exact computation of Voronoi diagrams and line segment intersections*. PhD thesis, Universität des Saarlandes, 1996.

[14] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. *Algorithms-ESA'94*, pages 227–239, 1994.

[15] N. Casati, M. Gabrani, R. Viswanathan, Z. Bayraktar, O. Jaiswal, D. L. DeMaris, A. Y. Abdo, J. Oberschmidt, and R. A. Krause. Automated sample plan selection for OPC modeling. In *Optical Microlithography XXVII, Proceedings of SPIE*, volume 9052, 2014.

[16] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[17] T. M. Chan. Random sampling, halfspace range reporting, and construction of $\leq$ k-levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000.

[18] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing kth-order Voronoi diagrams. *IEEE Transactions on Computers*, 100(11):1349–1354, 1987.

[19] P. Cheilaris, S. K. Dey, M. Gabrani, and E. Papadopoulou. Implementing the L_∞ segment Voronoi diagram in CGAL and applying in VLSI pattern analysis. pages 198–205, 2014.

[20] O. Cheong, H. Everett, M. Glisse, J. Gudmundsson, S. Hornus, S. Lazard, M. Lee, and H.S. Na. Farthest-polygon Voronoi diagrams. *Algorithms-ESA 2007*, pages 407–418, 2007.

[21] L. P Chew. Building voronoi diagrams for convex polygons in linear expected time. Technical report, Hanover, NH, USA, 1990.

[22] K. L. Clarkson and P. W. Shor. Application of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4:387–421, 1989.

[23] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer, 2008.

[24] O. Devillers. The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13:163–180, 2002.

[25] S. K. Dey and E. Papadopoulou. The L_∞ (L_1) farthest line-segment Voronoi diagram. In *Ninth International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 49–55. IEEE, 2012.

[26] D. Ding, J. A. Torres, and D. Z. Pan. High performance lithography hotspot detection with successively refined pattern identifications and machine learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(11):1621–1634, 2011.

[27] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, E. Welzl, and D. Wood. Stabbing line-segments. *BIT Numerical Mathematics*, 22(3):274–281, 1982.

[28] A. Fabri, G. J. Giezeman, L. Kettner, S. Schirra, and S. Schonherr. The CGAL kernel: A basis for geometric computation. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 191–202. Springer Berlin Heidelberg, 1996.

[29] A. Fabri, G. J. Giezeman, L. Kettner, S. Schirra, and S. Schonherr. On the design of CGAL, a Computational Geometry Algorithms Library. *Special issue on discrete algorithm engineering*, 30(11):1167–1202, 2000.

[30] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.

[31] S. Fortune. Numerical stability of algorithms for 2D Delaunay triangulations. In *Proceedings of the eighth annual symposium on Computational geometry*, pages 83–92. ACM, 1992.

[32] L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier, and P. Zimmermann. GNU MPFR: The GNU Multiple Precision Floating-point computations with Correct rounding, 2005.

[33] J. R. Gao, B. Yu, and D. Z. Pan. Accurate lithography hotspot detection based on pca-svm classifier with hierarchical data clustering. In *Proc. SPIE*, volume 9053, pages 90530E–90530E–10, 2014.

[34] T. Granlund. GNU MP: The GNU Multiple Precision arithmetic library, 4.1., 2004.

[35] P. J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, 1978.

[36] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(1-6):381–413, 1992.

[37] M. Held. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry*, 18(2):95–123, 2001.

[38] M. Held. Vroni and ArcVroni: Software for and applications of voronoi diagrams in science and engineering. In *Proceedings of the 8th International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'11, pages 3–12, Washington, DC, USA, 2011. IEEE Computer Society.

[39] M. N. Kamarianakis and M. I. Karavelas. Analysis of the incircle predicate for the euclidean voronoi diagram of axes-aligned line segments. *arXiv preprint arXiv:1107.5204*, 2011.

[40] M. N. Kamarianakis and M. I. Karavelas. Analysis of the incircle predicate for the Euclidean Voronoi diagram of axes-aligned line segments. *European Workshop on Computational Geometry*, pages 117–120, 2012.

[41] M. Karavelas. 2D segment Delaunay graphs. *CGAL Editorial Board, CGAL User and Reference Manual*, 4.6, 2012.

[42] M. Karavelas. 2D Voronoi diagram adaptor. *CGAL Editorial Board, CGAL User and Reference Manual*, 4.6, 2012.

[43] M. Karavelas. 2D segment Delaunay graphs. In *CGAL User and Reference Manual*. CGAL Editorial Board, 2015. 4.6.

[44] M. I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *International symposium on Voronoi diagrams in science and engineering*, pages 51–62, 2004.

[45] M. I. Karavelas and M. Yvinec. The Voronoi diagram of planar convex objects. *Algorithms-ESA*, pages 337–348, 2003.

[46] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *20th Annual Symposium on Foundations of Computer Science*, pages 18–27. IEEE, 1979.

[47] R. Klein. *Concrete and abstract Voronoi diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989.

[48] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry*, 3(3):157–184, 1993.

[49] D. T. Lee. *Farthest neighbor Voronoi diagrams and applications*. Northwestern University, Department of Electrical Engineering and Computer Science, 1980.

[50] D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computers*, c-31(6):478–487, 1982.

[51] D. T. Lee and R. L. Drysdale. Generalization of Voronoi diagrams in the plane. *SIAM Journal on Computing*, 10(1):73–87, 1981.

[52] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM Journal on Computing*, 28(3):864–889, 1998.

[53] C. H. Liu, E. Papadopoulou, and D. T. Lee. An Output-sensitive Approach for the L_1/L_∞ k-Nearest-Neighbor Voronoi Diagram. In *Proceedings of the 19th European Conference on Algorithms*, ESA'11, pages 70–81, Berlin, Heidelberg, 2011. Springer-Verlag.

[54] MATLAB. *version 8.4 (R2014b)*. The MathWorks Inc., Natick, Massachusetts, 2014.

[55] K. Mehlhorn, S. Meiser, and R. Rasch. Furthest site abstract Voronoi diagrams. *International Journal of Computational Geometry & Applications*, 11(06):583–616, 2001.

[56] K. Mehlhorn and S. Näher. *LEDA: a platform for combinatorial and geometric computing*. Cambridge University Press, 1999.

[57] M. Mukherjee, Z. Baum, J. Nickel, and T. G. Dunham. Optical rule checking for proximity-corrected mask shapes. In *Proc. SPIE, Optical Microlithography XVI*, volume 5040, pages 420–430, 2003.

[58] K. Mulmuley. Output sensitive and dynamic constructions of higher-order Voronoi diagrams and levels in arrangements. *J. Comput. Syst. Sci.*, 47(3):437–458, 1993.

[59] N. C. Myers. Traits: a new and useful template technique. *C++ Report*, 7(5):32–35, 1995.

[60] H. Nosato, H. Sakanashi, E. Takahashi, M. Murakawa, T. Matsunawa, S. Maeda, S. Tanaka, and S. Mimotogi. Hotspot prevention and detection method using an image-recognition technique based on higher-order local autocorrelation. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 13(1):011007, 2014.

[61] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics., second edition, 2000.

[62] E. Papadopoulou. Critical area computation for missing material defects in VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):583–597, 2001.

[63] E. Papadopoulou. The higher order Hausdorff Voronoi diagram and VLSI critical area extraction for via-blocks. In *Proc. 5th Int. Symposium on Voronoi diagrams in Science and Engineering*, pages 181–191, 2008.

[64] E. Papadopoulou. Net-aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):704–716, 2011.

[65] E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. *International Symposium on Algorithms and Computation,* pages 187–196, 2012.

[66] E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. *European Workshop on Computational Geometry*, pages 237–240, 2012.

[67] E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. *International Journal of Computational Geometry Applications*, 23(6):443–460, 2013.

[68] E. Papadopoulou and D. T. Lee. Critical area computation via Voronoi diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(4):463–474, 1999.

[69] E. Papadopoulou and D. T. Lee. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry and Application*, 11(5):502–528, 2001.

[70] E. Papadopoulou and D. T. Lee. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40:63–82, 2004.

[71] E. Papadopoulou and J. Xu. The L_∞ Hausdorff Voronoi Diagram Revisited. In *Eighth International Symposium on Voronoi Diagrams in Science and Engineering, ISVD 2011, Qingdao, China, June 28-30, 2011*, pages 67–74, 2011.

[72] E. Papadopoulou and M. Zavershynskyi. On higher order Voronoi diagrams of line segments. *In Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC)*, pages 177–186, 2012.

[73] E. Papadopoulou and M. Zavershynskyi. A sweepline algorithm for higher order Voronoi diagrams. pages 16–22, 2013.

[74] E. A. Ramos. On range reporting, ray shooting and k-level construction. In *Symposium on Computational Geometry*, pages 390–399, 1999.

[75] H. Rosenberger. Order-k Voronoi diagrams of sites with additive weights in the plane. *Algorithmica*, 6(4):490–521, 1991.

[76] M. I. Shamos and D. Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162, october 1975.

[77] S. Shim, W. Chung, and Y. Shin. Synthesis of lithography test patterns through topology-oriented pattern extraction and classification. In *Proc. SPIE, Design-Process-Technology Co-optimization for Manufacturability VIII*, volume 9053, pages 905305–905305–10, 2014.

[78] J. Siek, L. Q. Lee, and A. Lumsdaine. Boost Graph Library. http://www.boost.org/.

[79] M. C. Simmons, J. H. Kang, Y. Kim, J. I. Park, S. W. Paek, and K. S. Kim. A state-of-the-art hotspot recognition system for full chip verification with lithographic simulation. In *Proc. SPIE*, volume 7974, pages 79740M–79740M–9, 2011.

[80] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2013.

[81] K. Sugihara. A simple method for avoiding numerical errors and degeneracy in Voronoi diagram construction. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 75(4):468–477, 1992.

[82] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation - An approach to robust geometric algorithms. *Algorithmica*, 27(1):5–20, 2000.

[83] Y. Sun, Y. M. Foong, Y. Wang, J. Cheng, D. Zhang, S. Gao, N. Chen, B. I. Choi, A. J. Bruguier, M. Feng, J. Qiu, S. Hunsche, L. Liu, and W. Shao. Optimizing opc data sampling based on "orthogonal vector space". In *Proc. SPIE*, volume 7973, pages 79732K–79732K–10, 2011.

[84] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015.

[85] D. Vengertsev, K. Kim, S. H. Yang, S. Shim, S. Moon, A. Shamsuarov, S. Lee, S. W. Choi, J. Choi, and H. K. Kang. The new test pattern selection method for OPC model calibration, based on the process of clustering in a hybrid space. In *Proc. SPIE*, volume 8522, page 85221A, 2012.

[86] G. Viehoever, B. Ward, and H. J. Stock. Pattern selection in high-dimensional parameter spaces. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8326, page 37, 2012.

[87] R. Viswanathan, O. Jaiswal, M. Gabrani, N. Casati, A. Y. Abdo, J. Oberschmidt, and J. Watts. Experiments using automated sample plan selection for OPC modeling. In *Optical Microlithography XXVIII, Proceedings of SPIE*, volume 9426, 2015.

[88] Voronoi CAA: Voronoi Critical Area Analysis. IBM CAD Tool, Department of Electronic Design Automation, IBM Microelectronics Division, Burlington, VT. Initial patents: US6178539, US6317859.

[89] A. Wallin. OpenVoronoi. https://github.com/aewallin/openvoronoi.

[90] Webpage. CGAL Computational Geometry Algorithms Library. http://www.cgal.org.

[91] J. Y. Wuu, F. G. Pikus, A. Torres, and S. M. Marek. Rapid layout pattern classification. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, ASPDAC '11, pages 781–786, Piscataway, NJ, USA, 2011. IEEE Press.

[92] J. Xu, L. Xu, and E. Papadopoulou. Computing the map of geometric minimal cuts. *Algorithms and Computation*, pages 244–254, 2009.

[93] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2(1):365–393, 1987.

[94] C. K. Yap. Towards exact geometric computation. *Computational Geometry*, 7(1):3–23, 1997.

[95] G. Yoo, J. Kim, T. Lee, A. Jung, H. Yang, D. Yim, S. Park, K. Maruyama, M. Yamamoto, A. Vikram, and S. Park. OPC verification and hotspot management for yield enhancement through layout analysis. In *Proc. SPIE*, volume 7971, pages 79710H–79710H–11, 2011.