Implementing the L_∞ segment Voronoi diagram in CGAL and an application in VLSI pattern analysis

Panagiotis Cheilaris*, Sandeep Kumar Dey*, Maria Gabrani†, Evanthia Papadopoulou*
* Faculty of Informatics, Università della Svizzera italiana
† IBM Zurich Research Laboratory

***Abstract*—In this work we present a CGAL (Computational Geometry Algorithm Library) implementation of the line segment Voronoi diagram under the L_∞ metric, building on top of the existing line segment Voronoi diagram under the Euclidean (L_2) metric. CGAL is an open-source collection of geometric algorithms implemented in C++, used in both academia and industry. We also discuss a possible application of the L_∞ segment Voronoi diagram in the area of VLSI pattern analysis. In particular, we identify potentially critical locations in VLSI design patterns, where the pattern, when printed with the photolithography process and depending on its context and various process conditions, may differ substantially from the original intended VLSI design, improving on existing methods.**

***Index Terms*—Voronoi diagram; Delaunay graph; line segment; CGAL; VLSI pattern analysis.**

I. Introduction

Let S be a set of n sites in the plane (simple geometric shapes, such as points, line segments, or circular arcs). The (nearest-neighbor) *Voronoi diagram* [3], [4] of S is a subdivision of the plane into regions such that the region of a site $s \in S$ is the locus of points closer to s than to any other site in S. The *distance* of a site s from a point q in the plane is defined as $d(s,q) = \min_{p \in s} d(p,q)$, where the the interpoint distance $d(p,q)$ can be the Euclidean (L_2) distance or any other metric.

In this paper, we focus on the Voronoi diagram of line segments in the plane, under the L_∞ metric (or maximum norm): $d(p,q) = d_\infty(p,q) = \max(|p_x - q_x|, |p_y - q_y|)$. Let's assume that the segments are not crossing in their interiors (if they are, we can break each crossing segment into smaller segments until there is no crossing). In Figure 1, we show with red the Voronoi diagram of the same set of segments under the L_2 and the L_∞ metric.

The Voronoi diagram is a plane graph. Each *face* corresponds to the Voronoi region of a site $s \in S$:

$$\mathrm{reg}(s) = \{q \in \mathbb{R}^2 \mid d(s,q) < d(s',q), \forall s' \in S \setminus \{s\}\}.$$

Each region contains its defining segment (see Figure 1). The boundary between two neighboring faces is an *edge* of the diagram. Edges meet at *vertices* of the diagram.

The Voronoi diagram of segments under the L_∞ metric has some nice properties, compared to the corresponding L_2 diagram.

Supported in part by the Swiss National Science Foundation grant 134355, under the auspices of the ESF EUROCORES program EuroGIGA/VORONOI.

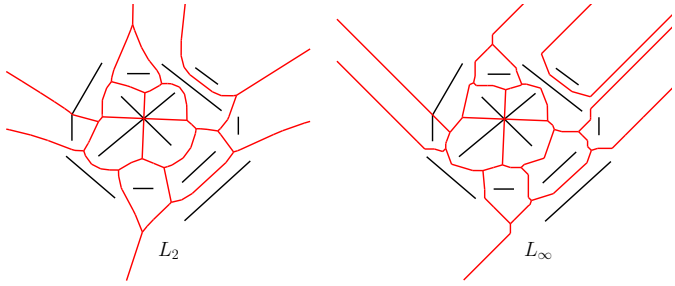

Fig. 1. Segment Voronoi diagram under L_2 and L_∞ metric.

1) The L_∞ diagram consists solely of *straight line segments* [17], whereas the L_2 diagram can also have *parabolic arcs* (see Figure 1).
2) If the coordinates of the endpoints of the input 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 [15] 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 segment is altering or erasing an existing vertex of the diagram. The L_2 in-circle test for arbitrary segments can be implemented with degree 40 [6], [7], whereas the corresponding L_∞ test only with degree 5 [17].

Segment Voronoi diagrams encode proximity information between polygonal objects. In many applications proximity is most naturally expressed with the L_2 distance, but there are applications, particularly in VLSI pattern analysis, for which the L_∞ distance is a good and simpler approximation to the L_2 distance [22], [17], [23], [16]. We remark that the *straight skeleton* [2] captures the shape of polygonal objects

in a natural manner, avoiding parabolic arcs (even in the L_2 metric), however, straight skeletons do not provide proximity information and therefore cannot be used in place of Voronoi diagrams. It is worth mentioning that the straight skeleton and the Voronoi diagram under the L_∞ coincide when the input consists of axis-parallel segments, which is predominant in VLSI designs.

Therefore, an implementation of the L_∞ segment Voronoi diagram is desirable, but, as far as we know, there is none freely available (except the proprietary [22]). Instead of building such an algorithm from scratch, we decided to develop it in the CGAL framework, on top of the existing L_2 segment Voronoi diagram of CGAL [10]. The Computational Geometry Algorithm Library (CGAL) is an open-source collection of a wide range of geometric algorithms implemented in C++. It is used in both academia and industry in various application domains, such as computer graphics, scientific visualization, computer aided design and modeling, mesh generation, etc. The geometric algorithms and data structures in CGAL are implemented with the design goals of robustness, genericity, flexibility, efficiency, and ease of use [9]. These design goals are fulfilled in CGAL by employing C++ generic programming [5], through template classes, and function templates [18].

CGAL is built in a modular way and there is provision for code reuse. We can exploit this provision by using a significant part of the L_2 segment Voronoi diagram incremental construction code [10] which is already in CGAL. In particular, most geometric predicates related to the L_2 diagram (like the in-circle test) are included in a *traits* class that is passed as a template parameter to the generic L_2 segment algorithm. Ideally, we can substitute this traits class with an analogous traits class for the L_∞ diagram. Still, writing geometric traits for the L_∞ geometric predicates and constructions is far from a trivial task, but the generic L_2 algorithm can be mostly reused. In practice, we also have to make some changes to the generic L_2 algorithm, but they are relatively few. In section III, we explain some details of our implementation that we consider the most interesting; a complete description would be out of scope for this paper.

Finally, we use our code for the L_∞ diagram for an application in VLSI pattern analysis. With the increase in miniaturization of current VLSI patterns, there is a significant rise in printability problems of such patterns, during the photolithography process. In order to address these printability problems, VLSI designers have developed several models, including *model-based optical proximity correction* (OPC). Currently OPC models are mainly based on image parameters of the test patterns [19], [1], [21]. We discuss the potential of using the L_∞ segment Voronoi diagram to identify critical locations in a VLSI layout pattern. This is an important step in selecting interesting test patterns from a large pool of patterns. These selected patterns will be sufficient to assess a big layout.

The rest of this work is organized as follows. We review the details of the L_2 segment Voronoi diagram implementation in CGAL, especially the ones that are relevant to our L_∞ implementation in section II. We explain some details of our L_∞ segment Voronoi diagram implementation in section III. We describe the application of the L_∞ segment Voronoi diagram in detecting potentially critical locations in VLSI design patterns in section IV.

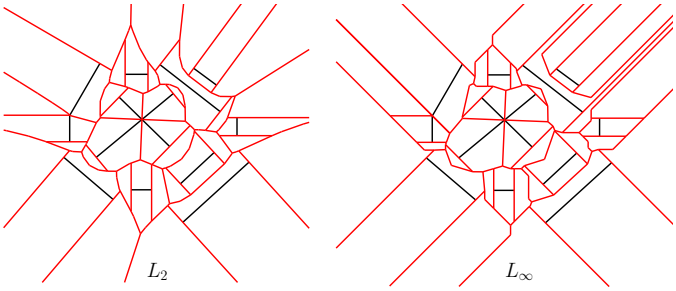

Fig. 2. Segment Voronoi diagram under L_2 and L_∞ metric, with distinct sites for interiors of segments and their endpoints.

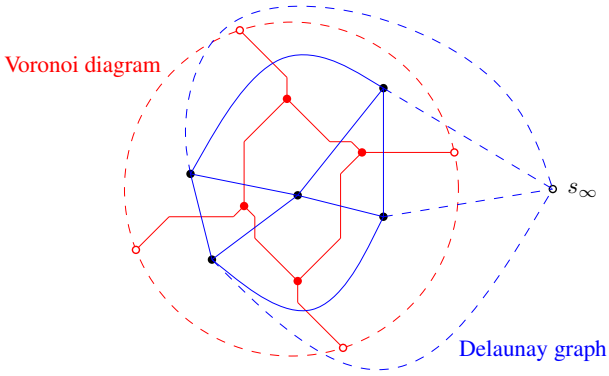

Fig. 3. L_∞ Voronoi diagram and Delaunay graph of five point sites (black), together with an additional site s_∞ at infinity. Infinite edges are shown dashed. Voronoi vertices are shown with red (finite ones are filled and infinite ones are unfilled).

II. L_2 SEGMENT VORONOI DIAGRAM IN CGAL

The *2D segment Delaunay graph package* [11] of CGAL provides a randomized incremental construction of the L_2 segment Voronoi diagram [13], [10]. The input S is a set of points and segments (possibly intersecting). The package supports intersections of segments by computing internally the *arrangement* $\mathcal{A}(S)$ 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) (see Figure 2 and compare with Figure 1). 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 $\mathcal{A}(S) = \{A, B, C, D, E, (AE), (EB), (CE), (ED)\}$. 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 [14], 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.

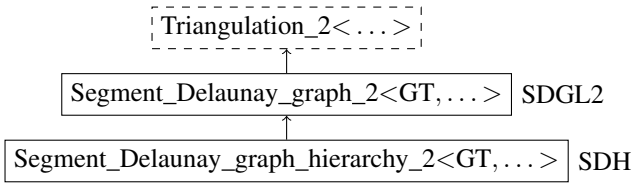

Fig. 4. Existing algorithm classes for the SDG L_2 package.

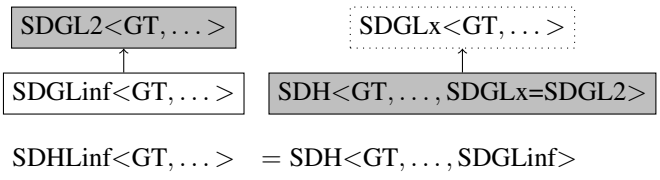

Fig. 5. New algorithm classes for SDG L_2 and L_∞ packages.

The dual graph of the segment Voronoi diagram is also a plane graph and is called the *segment Delaunay graph*. It is typical to always include an additional site s_∞ at infinity, as it simplifies the construction algorithms [13]. See Figure 3. As its name suggests, the Segment Delaunay graph package of CGAL computes in fact the *segment Delaunay graph* (SDG) under the L_2 metric. 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) 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 [8], [10]).

Both template classes have a mandatory template argument (denoted by GT in Figure 4), that must be instantiated with a geometric traits class, which contains geometric predicates related to the L_2 diagram (like the in-circle test). The requirements of this traits class are elaborated in the CGAL Segment-DelaunayGraphTraits_2 concept [11]. 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.

III. L_∞ SEGMENT VORONOI DIAGRAM IN CGAL

A. Design and relation with SDG L_2

In this section, we explain how we implement the segment Delaunay graph under the L_∞ metric (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. 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_∞ metric. Unfortunately, since the SDG L_2 algorithm classes were not designed with provision for other metrics except L_2, 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).

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 5).

Since 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 5, 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_∞ metric, for which we also create an alias template

class Segment_Delaunay_graph_hierarchy_Linf_2 (abbreviation: SDHLinf) (see Figure 5), 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.

B. 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 in the L_∞ setting. Since the incremental construction algorithm [13] expects uni-dimensional bisectors, we resort to a 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 [17].

If two different points p, q share one coordinate, then their L_∞ bisector is bi-dimensional, as shown in Figure 6. 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 7. We 1-dimensionalize this bisector by taking instead the line passing through the endpoint that is perpendicular to the segment.

C. 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 [17].

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 8. On the other hand, unbounded L_∞ parabolic arcs can survive in the corresponding L_∞ diagram. Even complete L_∞ parabolas can survive (see Figure 8).

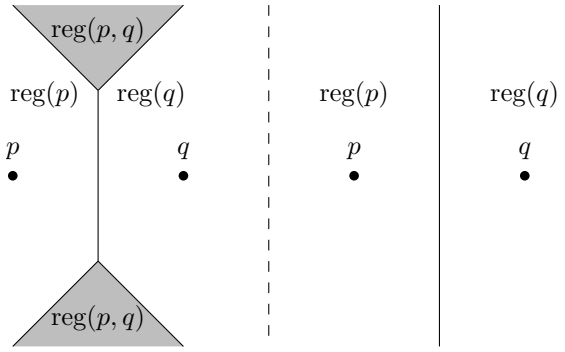

Fig. 6. The L_∞ bisector between two points with the same y coordinate and its 1-dimensionalization.

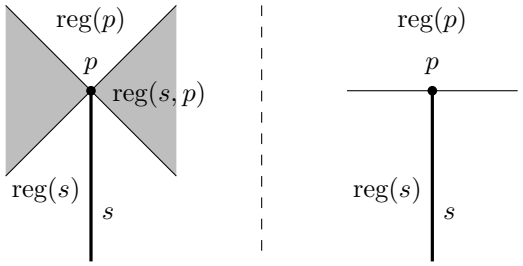

Fig. 7. The L_∞ bisector between a vertical segment and one of its endpoints and its 1-dimensionalization.

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 8, there are six distinct infinite edges neighboring with the region of the open segment.

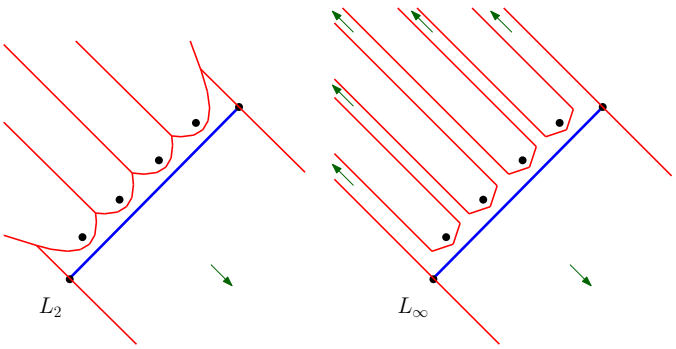

Fig. 8. Only bounded parabolic arcs survive in the L_2 diagram, whereas even complete L_∞ parabolas can survive in the L_∞ diagram. Arrows point to distinct infinite edges of the diagrams.

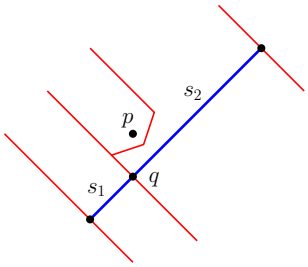

Fig. 9. The bisector that passes through q touches the parabolic arc at the parabolic arc's portion which is parallel to this bisector.

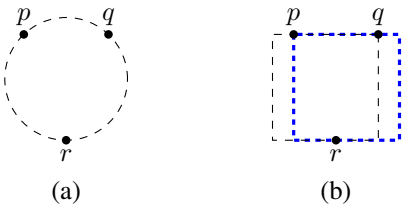

Fig. 10. (a) p, q, and r define a unique circle, (b) p, q, and r may not define a unique axis-parallel square.

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 9). 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.

D. 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 is implemented with template metaprogramming, using the *Substitution failure is not an error* (SFINAE) principle [18].

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.

E. 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 [13]. 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.

The vertex conflict predicates are also known as in-circle tests. 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 associated with an existing Voronoi vertex. 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 10). 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.

IV. Application in VLSI pattern analysis

VLSI patterns are shrinking in size 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 process. There are mainly two kind of faults that can occur during printing: a *pinch* and a *bridge*. 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.

A. VLSI problem — Identifying 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

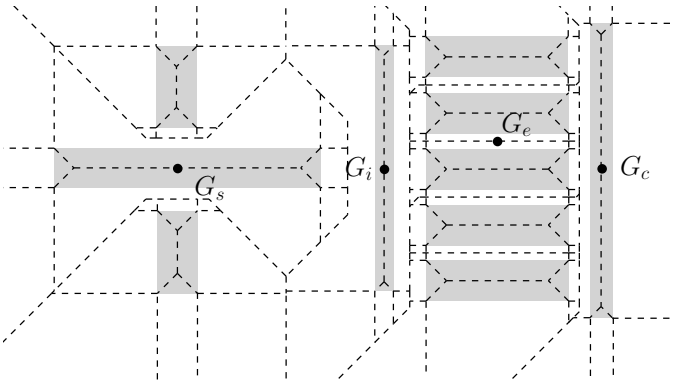

Fig. 11. Different gauge suggestions.

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. Existing pattern selection techniques are mainly based on image features [19], [1], [21], [20]. 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.

B. Hotspot identification and gauge suggestion problem

The measurement location in each pattern is called a *gauge* [21]. 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.

C. Gauge suggestion using the segment Voronoi diagram

We propose to use the L_∞ segment Voronoi diagram to suggest good gauge locations based on the proximity information of the shapes of a pattern, e.g., the spacing between shapes or the extent of interaction between neighboring shapes.

After studying several clips with existing gauge suggestions and SEM images of their prints zoomed and centered around the suggested gauge locations, that were supplied by IBM Zurich Research Laboratory, we developed our code (based on the L_∞ segment Voronoi diagram) that suggests four types of gauges (examples in Figure 11). The first two types of gauges are related to distances of *first-order* neighboring elements of shapes in the diagram. The last two types of gauges detect *second-order* distance interactions, e.g., elements of shapes that are close but have another shape element between them.

1) *Internal gauge* (inside a shape), G_i: It lies on the center of the Voronoi edge inside the polygonal shape of minimum width in the pattern. The position of G_i is the most probable for a pinch, when printing the pattern. In case there are many shapes with the minimum width, we prefer the one where the length of the Voronoi edge is greater, since long and thin shapes are more probable to give rise to a pinch.
2) *External gauge* (between different neighboring shapes), G_e: It lies on the center of the Voronoi edge between the two shapes that are closest in the pattern. The position of G_e is the most probable for the formation of a bridge between the two corresponding shapes, when printing the pattern. When there are more than two pairs of shapes with the same distance, we prefer the pair whose corresponding Voronoi edge is longer, as it implies more interaction between the shapes and higher probability of a bridge.
3) *Sandwich gauge*, G_s: It 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. There is a probability of a pinch happening at P_1 around G_s because of the influence of P_2 and P_3. If there are many (P_1, P_2, P_3) triples with the same distance between P_2 and P_3 in the pattern, we prefer the triple where the relevant Voronoi edges between P_1, P_2, P_3 overlap more.
4) *Comb gauge*, G_c: It lies on the center of the Voronoi edge inside a long polygonal shape P_1 (the base of the comb) that has close to it and on one side other polygonal shapes (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. The position of G_c is dangerous for a pinch, when printing the pattern.

The existing gauge suggestions are performing well only for a limited number of patterns and our internal and external gauges cover those limited cases. On the other hand, our sandwich and comb gauges can be potentially used to identify critical locations missed by the existing gauge suggestions. In the following, we validate the usefulness of the gauges suggested by our code.

D. Experiment

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 12–21). 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 I. 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.

Our experiment shows that the L_∞ segment Voronoi diagram can be used effectively to identify potentially critical locations of VLSI layouts.

E. Future work

A *model-based OPC* (MB-OPC) simulator [20], [1] is essential for advanced lithography processes. The simulator must be efficient and robust. Current simulators are not optimized and there is a tradeoff of efficiency and robustness. More input patterns to the simulator result in improved model accuracy, but the model build time is directly proportional to the number of input patterns. The main bottleneck is the difficulty to obtain an optimum set of patterns describing the faults and error-prone locations for a whole VLSI layout. Optimization for MB-OPC requires selection of optimal test patterns that would cover the whole layout. The current test pattern selection techniques are mainly based on image parameters [12], [19], [1], [21], [20], which are indicators of the 3D profile of the printed shapes. While these are very descriptive, they are very expensive to compute. The main advantage of our proposed methodology is that is provides gauges that are pattern dependent, include context information, and at the same time are orders of magnitude faster to compute. As future work, we intend to use the L_∞ segment Voronoi diagram for obtaining a good set of patterns for a VLSI layout that would in turn enhance MB-OPC. We would also like to analyze bigger layout clips on the order of tens of thousands of shapes.

V. Conclusions

We presented an implementation of the segment Voronoi diagram in the L_∞ metric based on the Segment Delaunay graph package of CGAL. The implementation involved three parts: (1) some generalization of the algorithm for existing segment Delaunay graph package to adapt it for the L_∞ segment Delaunay graph, (2) implementing the traits classes containing predicates and constructions for building Voronoi diagram of segments and points under the L_∞ metric, and (3) providing GUI demo, examples and ipelet for the package. In addition, we discussed some applications in the VLSI CAD area, in particular, the discovery of potentially problematic areas when printing VLSI layout patterns and a possible future work of obtaining an optimized set of patterns of interest (POIs), using proximity information from the L_∞ segment Voronoi diagram, with the goal of enhancing of the model-based optical proximity correction.

Acknowledgment

We thank Menelaos Karavelas for providing us insight into the L_2 segment Voronoi diagram implementation in CGAL.

References

[1] A. Abdo and R. Viswanathan, "The feasibility of using image parameters for test pattern selection during OPC model calibration," in *Proc. SPIE*, vol. 7640, 2010, p. 76401E.

[2] O. Aichholzer and F. Aurenhammer, "Straight skeletons for general polygonal figures in the plane," in *COCOON*, 1996, pp. 117–126.

[3] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

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

[5] H. Brönnimann, L. Kettner, S. Schirra, and R. C. Veltkamp, "Applications of the generic programming paradigm in the design of CGAL," in *Generic Programming*, 1998, pp. 206–217.

[6] C. Burnikel, "Exact computation of Voronoi diagrams and line segment intersections," Ph.D. dissertation, Universität des Saarlandes, 1996.

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

[8] O. Devillers, "The Delaunay hierarchy," *International Journal of Foundations of Computer Science*, vol. 13, pp. 163–180, 2002.

[9] A. Fabri, G. J. Giezeman, L. Kettner, S. Schirra, and S. Schonherr, "On the design of CGAL, a Computational Geometry Algorithms Library," *Software: Practice and Experience, Note: Special issue on discrete algorithm engineering*, vol. 30, no. 11, pp. 1167–1202, 2000.

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

[11] ——, "2D segment Delaunay graphs," in *CGAL User and Reference Manual*. CGAL Editorial Board, 2013, 4.3.

[12] A. Khoh, S.-F. Quek, Y.-M. Foong, J. Cheng, and B.-I. Choi, "Maximizing test pattern coverage for OPC model build," in *Proc. SPIE*, vol. 6154, 2006, p. 615437.

TABLE I
COMPARISION OF CD MEASUREMENT AT DIFFERENT GAUGE LOCATIONS

Pattern	existing 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%

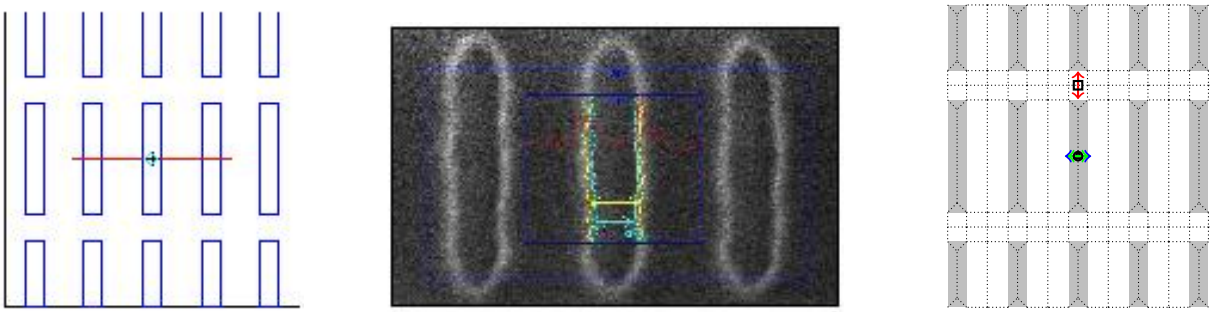

Fig. 12. Pattern A: our suggestion: G_i ($\triangle$, blue, same as existing gauge) coincides with G_s ($\circ$, green)

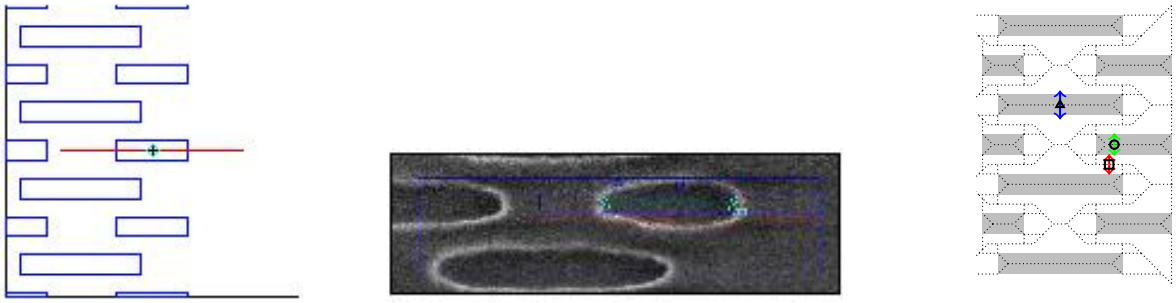

Fig. 13. Pattern B: our suggestion: G_e ($\square$, red) improves on existing gauge

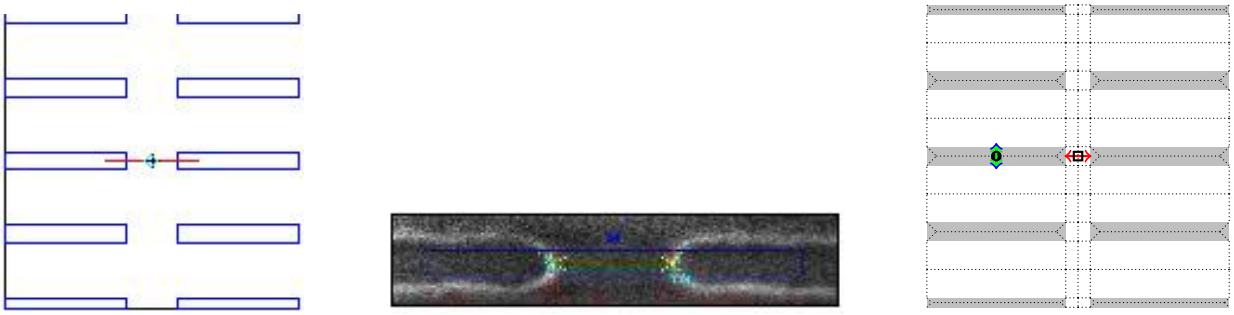

Fig. 14. Pattern C: our suggestion: G_i ($\triangle$, blue) coincides with G_s ($\circ$, green), improves on existing gauge

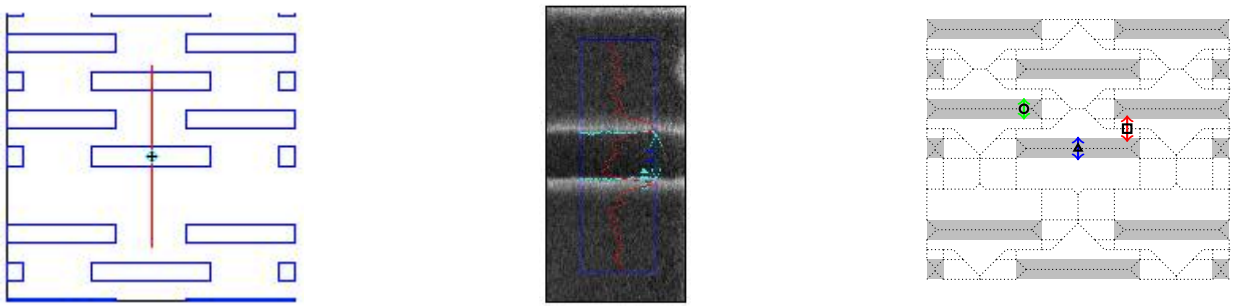

Fig. 15. Pattern D: our suggestion: G_e (□, red) improves on existing gauge

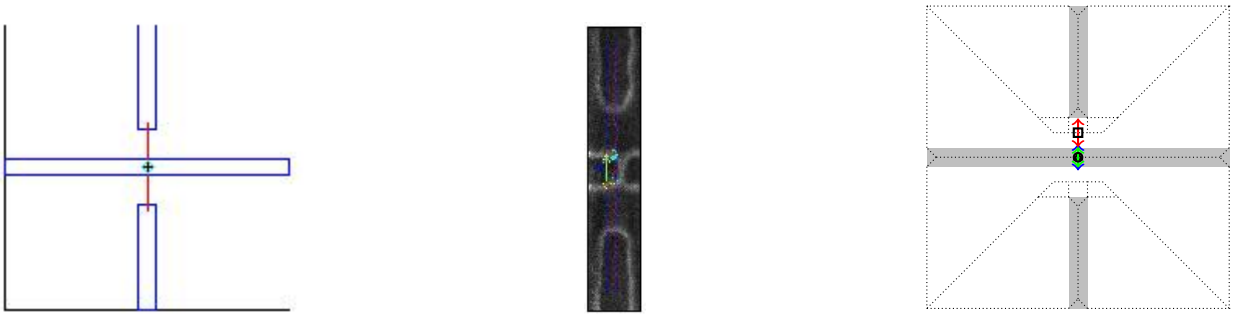

Fig. 16. Pattern E: our suggestion: G_s (○, green, same as existing gauge) coincides with G_i (△, blue)

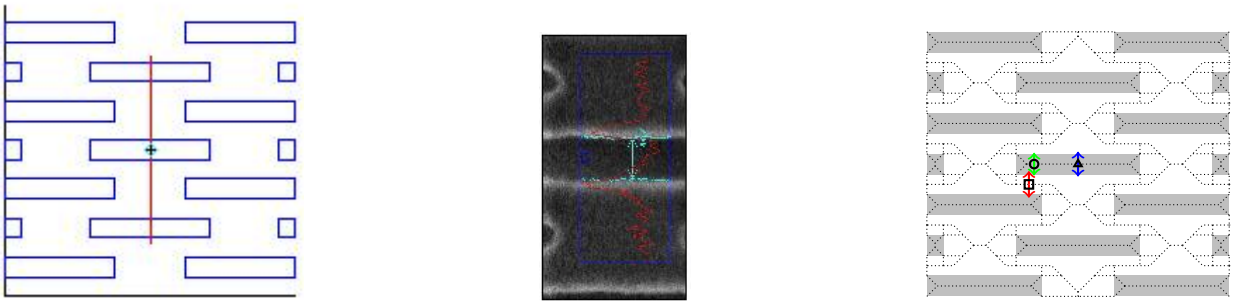

Fig. 17. Pattern F: our suggestion: G_e (□, red) improves on existing gauge

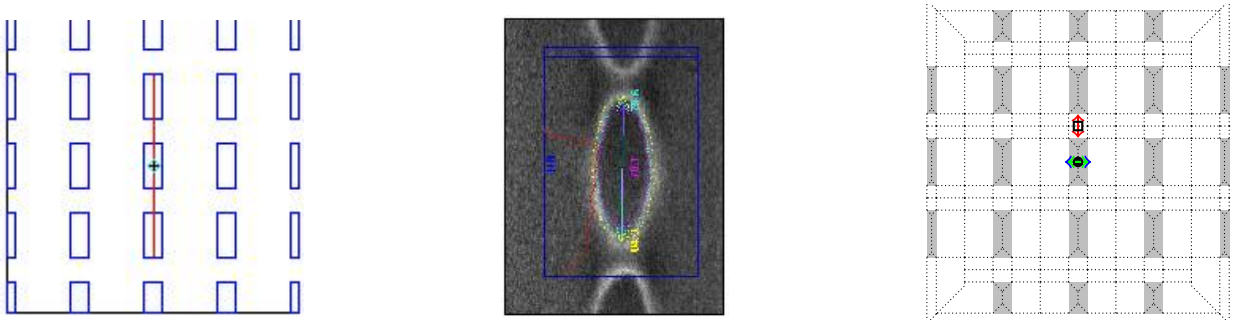

Fig. 18. Pattern G: our suggestion: G_e (□, red) improves on existing gauge

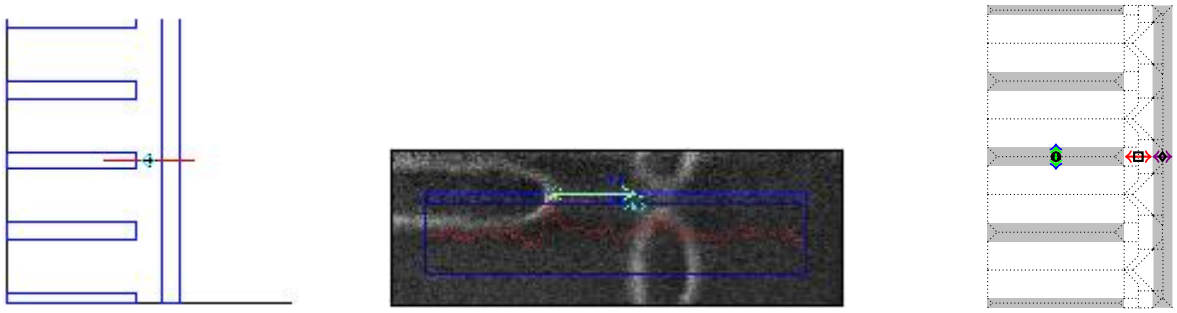

Fig. 19. Pattern H: our suggestion: G_c ($\diamond$, purple) detects a pinch, improves on existing gauge

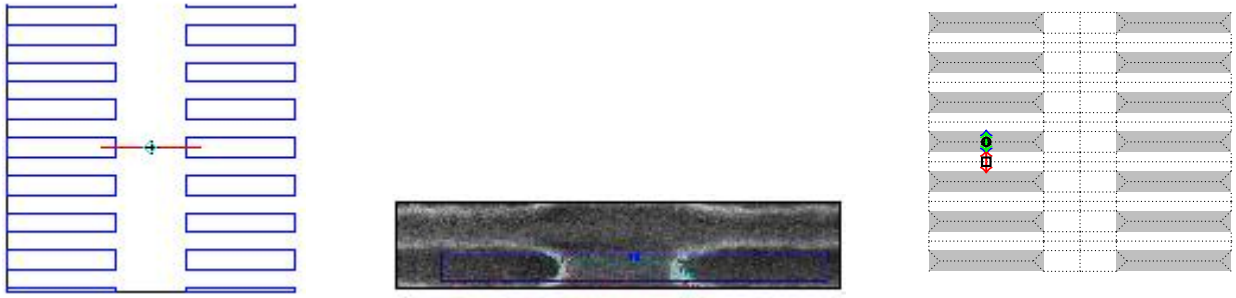

Fig. 20. Pattern I: our suggestion: G_e ($\square$, red) improves on existing gauge

Fig. 21. Pattern J: our suggestion: G_e ($\square$, red) improves on existing gauge

[13] R. Klein, K. Mehlhorn, and S. Meiser, "Randomized incremental construction of abstract Voronoi diagrams," *Computational Geometry*, vol. 3, no. 3, pp. 157–184, 1993.

[14] R. Klein, *Concrete and abstract Voronoi diagrams*, ser. Lecture Notes in Computer Science. Springer, 1989, vol. 400.

[15] G. Liotta, F. P. Preparata, and R. Tamassia, "Robust proximity queries: An illustration of degree-driven algorithm design," *SIAM Journal on Computing*, vol. 28, no. 3, pp. 864–889, 1998.

[16] 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*, vol. 30, no. 5, pp. 704–716, 2011.

[17] E. Papadopoulou and D. T. Lee, "The L_∞ Voronoi diagram of segments and VLSI applications," *International Journal of Computational Geometry and Application*, vol. 11, no. 5, pp. 502–528, 2001.

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

[19] 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*, vol. 7973, 2011, p. 79732K.

[20] 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*, vol. 8522, 2012, p. 85221A.

[21] G. Viehoever, B. Ward, and H. J. Stock, "Pattern selection in high-dimensional parameter spaces," in *Proc. SPIE*, vol. 8326, 2012, p. 832618.

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

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