Critical Area Computation for Missing Material Defects in VLSI Circuits

Evanthia Papadopoulou

***Abstract*—We address the problem of computing critical area for missing material defects in a circuit layout. The extraction of critical area is the main computational problem in very large scale integration yield prediction. Missing material defects cause open circuits and are classified into breaks and via blocks. Our approach is based on the L_∞ medial axis of polygons and the weighted L_∞ Voronoi diagram of segments. We also introduce the min–max Voronoi diagram of rectangles, a combinatorial structure of independent interest. The critical area problem for breaks and via blocks is reduced to variations of weighted L_∞ Voronoi diagram of segments. Plane sweep algorithms to compute the appropriate Voronoi diagrams for each case are presented. As a result, the critical area for breaks and via blocks on a single layer can be computed accurately in one pass of the layout. The time complexity is $O(n \log n)$ in the case of breaks and $O((n + K) \log n)$ in the case of via blocks, where n is the size of the input and K is upper-bounded by the number of interacting vias (in practice K is small). The critical area computation assumes square defects and reflects all possible defect sizes following the $D(r) = r_0^2/r^3$ defect size distribution. The method is presented for rectilinear layouts.**

***Index Terms*—Breaks, critical area, missing material defects, opens, via blocks, Voronoi diagrams, VLSI layout, yield prediction.**

I. Introduction

VERY LARGE scale integration (VLSI) yield prediction is based on the concept of *critical area*, which reflects the sensitivity of a design to spot defects occurring during the manufacturing process (see, for example, [6], [10], [11], [16], [19], and [22]–[24]). Yield prediction is of growing importance in modern VLSI design due to the need to control the cost of manufacturing. Spot defects are caused by particles such as dust and other contaminants in materials and equipment. They are classified into "extra material" defects causing shorts between different conducting regions and "missing material" defects causing open circuits. Missing material defects are classified into *breaks* and *via blocks*. A break is a defect interfering with the continuity of an interconnect; a via block is a defect destroying a contact on a via layer. The extraction of critical area for both extra material defects (shorts) and missing material defects (opens) causes the main computational bottleneck in yield prediction methodology. A new approach to critical area computation for shorts via Voronoi diagrams was given in [16]. This paper extends the framework introduced in [16] with the ability to handle opens, i.e., breaks and via blocks, offering a unifying approach to critical area extraction via Voronoi diagrams. To the best of our knowledge, it gives the first low-polynomial order algorithm to accurately compute critical area for opens due to spot defects on a single layer. As a byproduct, it also introduces the L_∞ min–max Voronoi diagram of rectangles, a combinatorial structure of independent interest.

The critical area in one layer of a circuit layout C is defined as

$$A_c = \int_0^\infty A(r)D(r)\,dr$$

where $A(r)$ denotes the area in which the center of a defect of radius r must fall in order to cause a circuit failure and $D(r)$ is the density function of the defect size. A circuit failure represents an open circuit due to a break or a via block. The defect density function has been estimated as follows [6], [7], [23], [24]:

$$D(r) = \begin{cases} cr^q/r_0^{q+1}, & 0 \le r \le r_0 \\ cr_0^{p-1}/r^p, & r_0 \le r \le \infty \end{cases} \tag{1}$$

where p, q are real numbers (typically $p = 3$, $q = 1$), $c = (q + 1)(p-1)/(q+p)$, and r_0 is some minimum optically resolvable size. Using typical values for p, q, and c, we derive the widely used defect size distribution $D(r) = r_0^2/r^3$. (r_0 is typically smaller than the minimum feature size, thus, $D(r)$ is ignored for $r < r_0$.) In this paper, a defect of size r is modeled as a square of radius r, i.e., a square of side $2r$. As discussed in [16], modeling defects as squares corresponds to computing critical area in the L_∞ metric.[1] In reality, spot defects have any kind of shape; thus, the square defect model (a common simplification in the critical area literature) is good enough for all practical purposes. Square defects are among the most common simplifications found in the critical area literature. A formal bound is given in Section VI.

A variety of methods to extract critical area have been proposed in the past, the majority dealing with shorts. For breaks, assumptions are usually made in order to treat the problem as dual to shorts. Existing methods for breaks can be summarized as follows.

1) *Monte Carlo Simulation:* Draw a large number of defects with their radii distributed according to $D(r)$, check for each defect if it causes an open, and divide the number of defects causing faults by the total number of defects to derive the probability of fault [25].

Manuscript received July 17, 2000; revised November 7, 2000. This paper was recommended by Associate Editor D. Hill.

The author is with the IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (e-mail: evanthia@watson.ibm.com).

Publisher Item Identifier S 0278-0070(01)03078-0.

[1]The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is the maximum of the horizontal and the vertical distance between p and q, i.e., $d(p, q) = \max\{|x_p - x_q|, |y_p - y_q|\}$.

2) *Geometric Methods:* Compute the area of critical region $A(r)$ for several different values of r independently; use the results to approximate the total critical area. Opens are treated geometrically without considering actual breaks of connectivity. They are usually based on shape manipulation tools providing operations such as *shrink-shape-by-r* and *find-area* (see [12], [14]). The time complexity for each defect radius depends on the underlying shape manipulation algorithms. A more efficient scan-line method to compute $A(r)$ for rectilinear layouts is given in [19].
3) *Grid Method:* A fine grid is assumed over the layout and the *critical radius*[2] for every grid point is computed. The run-time is $O(I^{1.5})$ time, where I is the number of grid points [24].

A more thorough analysis to critical area computation for breaks is given in [21]. $A(r)$, for a given defect radius r, is calculated strictly over each shape (critical regions expanding in the free space are ignored). This method, unlike the geometric ones, considers actual breaks of connectivity and runs in $O(n^2 \log n)$ time where n is the size of the layout. Via blocks have not received much attention in the literature with the exception of [12] and [25], although they form a serious factor of yield loss in current technologies. In [12], a simplified definition for a via block is used: a defect is a via block if it overlaps any portion of the contact or via; for multiple redundant vias a defect is a block if it overlaps at least k of these vias ($k = 2$). The method falls in category 2 of geometric approaches, where $A(r)$ is computed by operations usually available in design rule check (DRC) tools.

In this paper, we first give a geometric modeling of breaks and via blocks. A break is formally defined using the concept of the *medial axis* of a shape. For via blocks we use the definition of [13] and [25], which is accepted by IBM manufacturing. This definition is more involved but more realistic than the one used in [12]. Then the problem of computing critical area for breaks and via blocks on a single design layer is reduced into variations of (weighted) L_∞ Voronoi diagrams. These variations are interesting in their own right. In particular, the via blocks problem introduces a min–max type of Voronoi diagram that had not been addressed before. Once the appropriate Voronoi diagrams are computed, the total critical area integral for breaks and via blocks can be computed analytically following the framework introduced in [16] for shorts. We present plane sweep algorithms to compute the Voronoi diagram needed for each case. The time complexity is $O(n \log n)$ for the Voronoi diagram of breaks and $O((n+K) \log n)$ for the Voronoi diagram of via blocks, where n is the size of the input and K is upper-bounded by the total number of *interacting vias*[3]; in practice, K is negligible. Thus, the total critical area integral for breaks and via blocks can be computed in time $O(n \log n)$ and $O((n + K) \log n)$, respectively, in one pass of the layer under consideration. Note that previous methods, except the Monte Carlo simulation and the grid-based method, compute only the critical region $A(r)$ for a specific defect size r. The method is presented for rectilinear layouts. Spot defects affecting more than one layer are not considered in this paper.

[2]The critical radius at point t is the radius of the smallest defect centered at t causing an open.

[3]See Section IV-A for the definition.

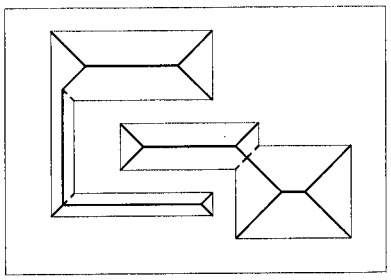

Fig. 1. L_∞ medial axis of rectilinear polygons.

This paper is organized as follows. Section II reviews some basic concepts of L_∞ Voronoi diagrams and introduces the weighted L_∞ Voronoi diagram of axis-parallel segments. Section III gives a geometric interpretation of breaks and reduces the critical area computation for breaks to a weighted L_∞ Voronoi diagram of segments. In Section IV, the critical area problem for via blocks is reduced to a min–max type of Voronoi diagram resulting in a weighted L_∞ Voronoi diagram of segments. In Section V, the plane sweep algorithm to compute the Voronoi diagram for breaks and via blocks is described. Finally, Section VI shows that the critical area integral can be discretized as a summation of Voronoi edges. Section VII gives experimental results.

II. L_∞ Voronoi Diagrams

In this section, we give the background on Voronoi diagrams needed for subsequent sections. We first review some basic concepts and then introduce the weighted L_∞ Voronoi diagram of axis-parallel segments. The reader is referred to [17] and [27] for a more detailed discussion of L_∞ Voronoi diagrams.

The Voronoi diagram of a set of polygonal sites is a partitioning of the plane into regions, called *Voronoi cells*, such that the Voronoi cell of a site s is the locus of points closer to s than to any other site. The Voronoi cell of s, also called the Voronoi region of s, is denoted as $\mathrm{reg}(s)$ and s is referred to as the *owner* of $\mathrm{reg}(s)$. The boundary that borders two Voronoi cells is called a *Voronoi edge* and consists of portions of *bisectors* between the owners of the cells. The point where three or more Voronoi edges meet is called a *Voronoi vertex*. In the interior of a simple polygon P the Voronoi diagram is also called *medial axis*.[4] Formally, the medial axis is the locus of points $\{q\}$ internal to P such that there are at least two points on the object's boundary that are equidistant from $\{q\}$ and are closest to $\{q\}$ (see [2] and [8]). Fig. 1 illustrates the medial axis of rectilinear shapes in the L_∞ metric. Associated with the medial axis, there is a radius function R that defines for each point on the axis its distance to the boundary of the object. For details on Voronoi diagrams, see [2] and [20].

Throughout this paper, we use the L_∞ metric. The use of the L_∞ metric simplifies the Voronoi diagram of polygonal objects

[4]There is a minor difference in the definition that we ignore in this paper (see [2] and [8]).

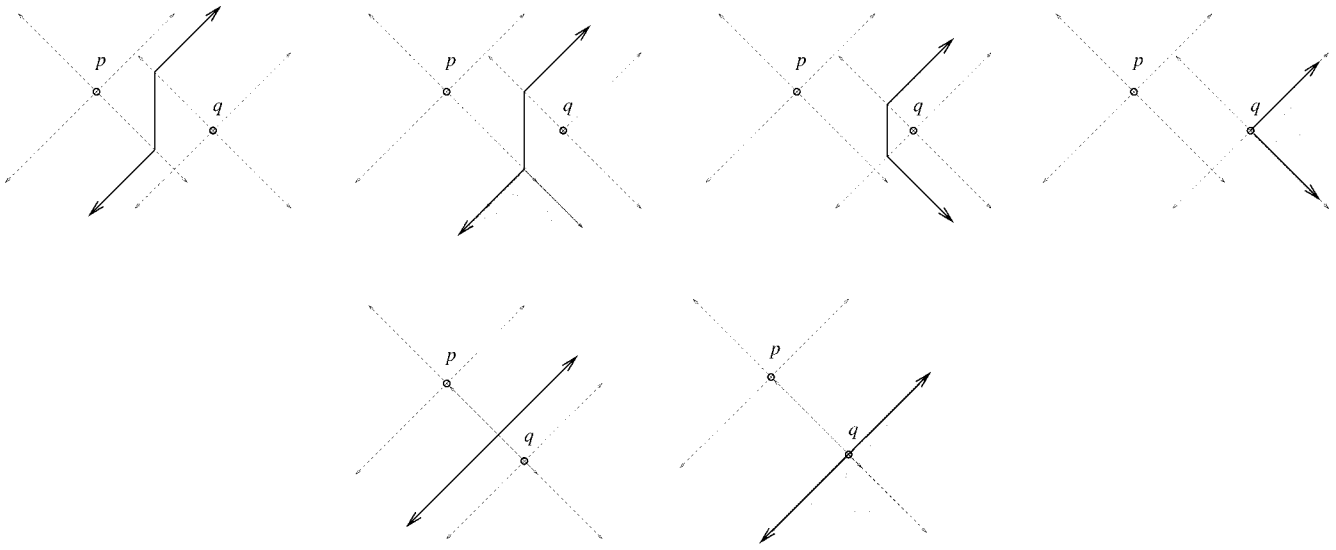

Fig. 2. L_∞ bisector of additively weighted points ($w_p < w_q \leq w_p + d(p, q)$).

and makes its computation feasible in practice [17]. The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is $d(p, q) = \max\{|x_p - x_q|, |y_p - y_q|\}$. Intuitively, the L_∞ distance between two points p and q is the side of the smallest square touching p and q. The L_∞ distance between a point p and a line l is $d(p, l) = \min\{d(p, q), \forall q \in l\}$.

The L_∞ bisector of two polygonal elements (points or lines) is the locus of points at equal L_∞ distance from the two elements. The leftmost figure in Fig. 2 illustrates the L_∞ bisector of two points. Recall that the 45° rays[5] emanating from a point p partition the plane into four quadrants such that for any point q within a quadrant, the L_∞ distance simplifies to the vertical (north and south quadrant) or to the horizontal (east and west quadrant) distance between p and q (see Fig. 2). In each quadrant, a point is equivalent to an axis-parallel line. For more information see [16].

Let us now assume that points p and q are weighted with additive weights w_p and w_q, respectively, such that $0 \leq w_p \leq w_q \leq w_p + d(p, q)$. The (weighted) L_∞ bisector of p, q is the locus of points equidistant from p and q in a weighted sense, i.e., $b(p, q) = \{t|d(t, p) + w_p = d(t, q) + w_q\}$. Note that if $w_q > w_p + d(p, q)$, there is no bisector between p and q: all points in the plane (including q) are closer to p than q. If $w_q = w_p + d(p, q)$, then the area enclosed by two 45° rays through q is equidistant from both points. Fig. 2 shows L_∞ bisectors of additively weighted points as w_q increases ($w_p < w_q$). Without creating any significant difference, when a whole region is equidistant from both points (shaded regions in Fig. 2), we assign it to one of the points and consider only the outermost boundary of the bisecting region as the bisector (thick rays in Fig. 2). The L_∞ Voronoi diagram of additively weighted points is similar to the unweighted one. The main difference is that an arbitrarily weighted point may or may not have a Voronoi region. The size of the diagram remains linear.

The Voronoi diagram of additively weighted segments has not been given any attention in the literature (to the best of our knowledge). In this paper, we consider the L_∞ Voronoi diagram of additively weighted axis-parallel segments where the weights are nonnegative constants. The (weighted) L_∞ distance between a point t and a weighted segment s is $d_w(t, s) = d(t, s) + w(s)$, where $w(s)$ denotes the weight of s and $d(t, s)$ denotes the ordinary L_∞ distance between t and s. A segment is assumed to consist of three elements: two endpoints and an open line segment. The definition of a bisector between two segments s_i, s_j remains the same: $b(s_i, s_j) = \{y|d_w(s_i, y) = d_w(s_j, y)\}$. Fig. 3 illustrates the bisector of two additively weighted orthogonal lines where $w(l_1) < w(l_2)$; the weights basically correspond to shifting the lines in parallel. Fig. 4 depicts the bisector of two weighted axis-parallel segments for different weights. The 45° rays emanating from the endpoints of the segments are bisectors between the endpoints and the open parts of the segments. In Fig. 4(a), both segments have equal weights. Fig. 4(b)–(e) depicts $b(s_1, s_2)$ as the weight of s_2 increases ($w(s_1) < w(s_2)$). Fig. 4(f)–(j) depicts $b(s_1, s_2)$ as the weight of s_1 increases ($w(s_1) > w(s_2)$). Shaded regions indicate areas equidistant from both segments. Equidistant regions can be assigned arbitrarily to one of the two segments. A convention to maximize similarity with the unweighted case is to assign the equidistant region to the segment of larger weight. Note, in Fig. 4(e), that if $w(s_2)$ is increased further, there will be no bisector between s_1, s_2. This is similar for Fig. 4(j).

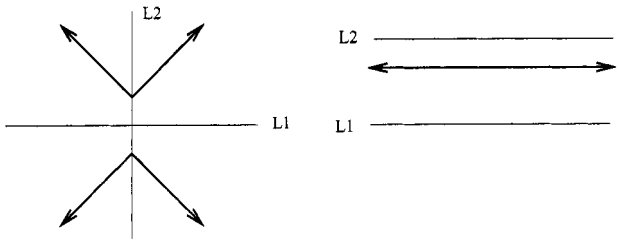

Fig. 3. L_∞ bisector of weighted lines.

The L_∞ Voronoi diagram of arbitrarily weighted axis-parallel segments may contain disconnected regions. See, for example,

[5]A 45° ray is a ray of slope $+1$ or -1.

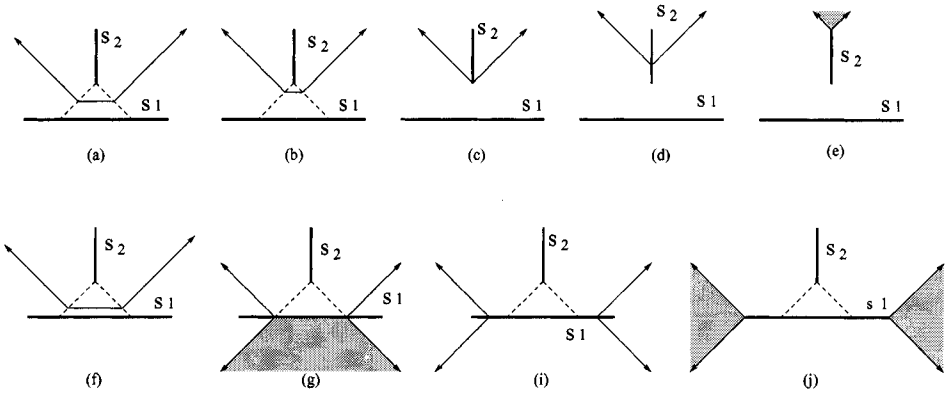

Fig. 4. L_∞ bisector of additively weighted segments. (a)–(e) $w(s_1) \leq w(s_2)$. Weight of s_2 increases from (a) to (e). (f)–(j) $w(s_2) < w(s_1)$. Weight of s_1 increases from (f) to (j).

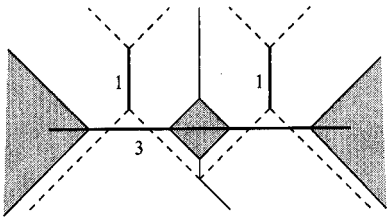

Fig. 5. L_∞ Voronoi diagram of weighted segments may contain disconnected regions.

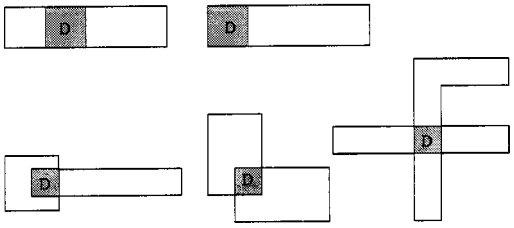

Fig. 6. D is a minimal break.

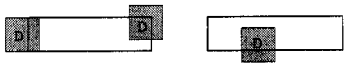

Fig. 7. D is not a break.

Fig. 5, where shaded regions depict the Voronoi cell(s) of the horizontal segment. The reason is that weighted bisectors may be disconnected [see Fig. 4(i) and (j)]. In this paper, weighted segments are not arbitrary and their bisectors, if they exist, are always connected [the configuration depicted in Fig. 4(i) and (j) cannot occur]. Under the connectivity assumption, the bisector of two weighted segments (if any) partitions the plane into two "half-planes," where $H(s_i, s_j)$ denotes the "half-plane" closer to s_i. If $d(s_i, s_j) \geq |w(s_i) - w(s_j)|$, each segment is contained entirely in one half-plane [see Fig. 4(a)–(c), (f), and (g)]. Otherwise, the bisector intersects the segment of larger weight [see Fig. 4(d)]. By definition, $\mathrm{reg}(s_i) = \cap_{i \neq j} H(s_i, s_j)$ and, thus, $\mathrm{reg}(s_i)$ must be a connected region, if nonempty (see proof of Theorem 3). Clearly, there may be segments contained only partially within their Voronoi cells and segments with no Voronoi cell. If $d(s_i, s_j) \geq |w(s_i) - w(s_j)|$ for all pairs of segments, we have exactly one Voronoi cell for each segment [using the above tie breaking convention in case of $d(s_i, s_j) = |w(s_i) - w(s_j)|$].

In the following, we reduce the problem of critical area computation for breaks and via blocks into variations of additively weighted L_∞ Voronoi diagrams of segments. The weights as well as the length of segments are not arbitrary resulting in Voronoi cells that are always connected. In the case of breaks, unlike via blocks, the weights follow the restriction $d(s_i, s_j) \geq |w(s_i) - w(s_j)|$ for any two segments s_i, s_j. The connectivity of Voronoi cells assures the linear size of the Voronoi diagrams.

III. Geometric Modeling of Breaks

In this section, we give a formal definition of a break and describe the Voronoi diagram variation needed for critical area computation. Consider a layer M, where missing material defects break the continuity of interconnections and contact plugs. We assume that overlapping shapes have been unified into single shapes and, thus, all polygons are disjoint.

A *simple* shape corresponds to a simple polygon and contains no holes. A shape with hole(s) is called *complex*. For a simple shape, a defect D is a *minimal break* if D breaks the shape into two or more pieces and D has minimal size, i.e., if D is shrunk by $\epsilon \geq 0$ then D will be entirely contained in the interior of the shape. A piece of a shape may trivially consist of a single edge. Fig. 6 shows examples of defects considered to be minimal breaks. Fig. 7 shows examples of defects that are not considered to be breaks. A minimal break is called *strictly minimal* if it contains no other minimal break in its interior. A *break* is any defect totally covering a minimal break. For a complex shape, a break is additionally any defect overlapping the outer and inner boundary of the shape or any two distinct inner boundaries (see Fig. 8). A limitation to the geometric modeling of breaks is that by ignoring the actual connections, critical area may be overestimated in case of interconnect shapes that contain redundant or nonconducting regions. Similar to existing geometric methods, we ignore such redundant regions since their identification can be treated as a separate problem and be removed prior to critical area calculations.

Let us now define the *core* of a rectilinear shape P (simple or complex) as a generator for breaks for P. Consider the L_∞ me-

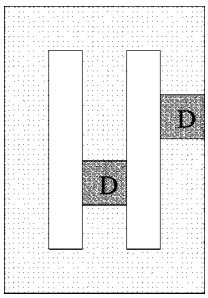

Fig. 8. D is a minimal break of a complex shape.

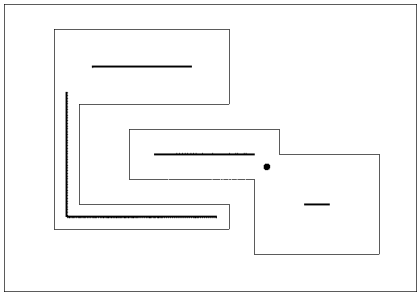

Fig. 9. Core of simple shapes.

dial axis of P. The radius function associated with the medial axis can be treated as an additive weight function. That is, any point p along the medial axis is weighted by $w(p) = d(p, P)$, the distance of p from the boundary. Consider the square D centered at p of radius $w(p)$. If p is induced by nonconsecutive edges of P, then D must be a minimal break (by the definition of the medial axis). We say that D is *generated* by point p. Let the *core* of P, denoted as $core(P)$, be the collection of vertices and axis-parallel edges of the medial axis. In Fig. 9, $core(P)$ is shown thickened. Core segments that are adjacent and core segments that are incident to the same 45° medial axis edge are called *neighboring*. It is not hard to see that any strictly minimal break of P must be centered along the core of P.

The *critical radius* of any point t is the radius of the smallest defect centered at t causing a break. The (weighted) distance of t from a core element s is $d_w(t, s) = d(t, s) + w(s)$. Note that the critical radius of a point in the interior of a shape need not be obtained by that shape. For example, in Fig. 10, the critical radius of $t \in S$ is determined by edge e in P. Our goal is to obtain a subdivision of the plane into regions such that the critical radius of any point is easy to derive.

Lemma 1: For any break of radius r centered at a point t, there is a core element $s \in core(P)$ such that $r \geq d(t, s) + w(s)$.

Proof: By definition, any break must overlap a strictly minimal break D. Let p be the core point generating D. The radius of D is $w(p)$. Since the break entirely overlaps D, $r \geq d(t, p) + w(p)$. Let s be the core segment where p belongs. Since all points along s have the same weight and $d(t, p) \geq d(t, s)$, the lemma follows. □

Lemma 2: The critical radius of any point t on a layer M is $r_c(t) = \min\{d(t, s) + w(s),\ \forall s \in G = \cup_P core(P)\}$, where G is the set of core elements of all polygons on M.

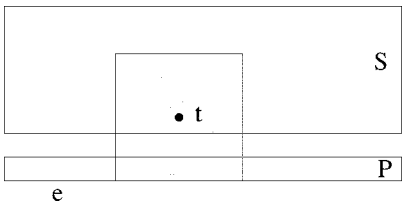

Fig. 10. $r_c(t) = d(t, e)$

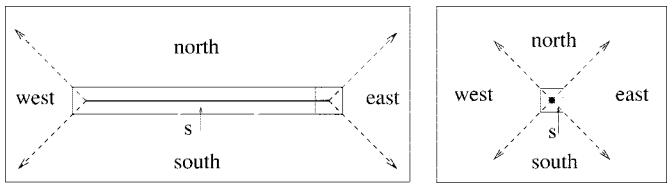

Fig. 11. $r_c(t) = d(t, s) + w(s)$.

Proof: By Lemma 1, $r_c(t) \geq \min\{d(t, s) + w(s),\ \forall s \in G\}$. Let s' be the core element nearest t (in a weighted sense) i.e., $d_w(t, s') = \min\{d(t, s) + w(s),\ \forall s \in G = \cup_P core(P)\}$ and let p be the point along s' such that $d_w(t, s') = d_w(t, p) = d(t, p) + w(p)$. The square, Q, centered at t of radius $d_w(t, p)$ must clearly overlap the minimal break generated by p having radius $w(p)$. That is, Q is a break. But then $r_c(t) \leq d_w(t, p)$. Thus, $r_c(t) = \min\{d(t, s) + w(s),\ \forall s \in G\}$. □

Let us now give an intuitive explanation of Lemma 2 (see Fig. 11). Consider a horizontal core segment $s = core(R)$, where R is a rectangle. Let s^n, s^s, s^e, s^w denote the north, south, east and west side of s and let R^n, R^s, R^e, and R^w denote the north, south, east and west edge of R. The 45° rays emanating from s^e, s^w, in Fig. 11, are bisectors between the endpoints and the open portion of s and partition the plane into four quadrants. For any point t in the north (resp. south) quadrant the critical radius of t is determined by the distance to R^s (resp. R^n) i.e., $r_c(t) = d(t, R^s) = d(t, s) + w(s)$ [resp. $r_c(t) = d(t, R^n) = d(t, s) + w(s)$]. For a point t in the west (resp. east) quadrant any break has to overlap with the minimal break centered at s^w (resp. s^e) of radius $w(s)$. Thus, the critical radius of t is $r_c(t) = d(t, s^w) + w(s^w)$ [resp. $r_c(t) = d(t, s^e) + w(s^e)$]. In all cases, $r_c(t) = d(t, s) + w(s)$ as shown in Lemma 2.

Consider the weighted L_∞ Voronoi diagram of the union of core elements, G, denoted as $\mathcal{V}(G)$. $\mathcal{V}(G)$ is a subdivision of the plane into regions such that the Voronoi region of a core element s is the locus of points closer to s (in a weighted sense) than to any other core element. $\mathrm{reg}(s)$ is further subdivided into finer regions by the 45° rays emanating from the endpoints of s (if not incident to another core segment). In every subregion the L_∞ distance simplifies to either the vertical or the horizontal distance from s. Fig. 12 illustrates the weighted L_∞ Voronoi diagram of a set of core elements where the numbers indicate weights; dashed lines indicate the finer partition of Voronoi cells. By Lemma 2 and the definition of $\mathcal{V}(G)$, we derive the main result of this section.

Theorem 1: Given the (weighted) L_∞ Voronoi diagram of the set of core segments G on layer M, the critical radius for

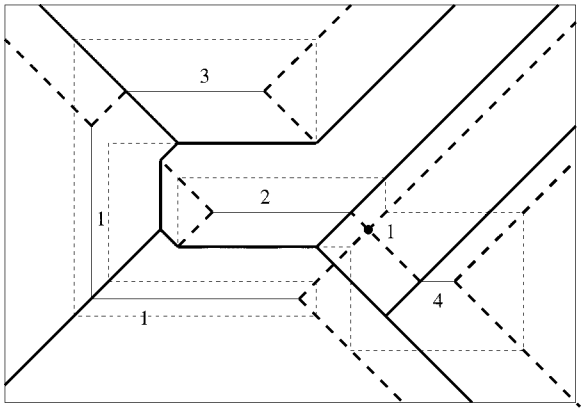

Fig. 12. Weighted Voronoi diagram of core elements.

Fig. 13. Contact as the minimum enclosing rectangle of redundant vias. (a) Redundant vias. (b) Unified contacts.

breaks for any point t is $r_c(t) = d_w(t, s) = d(t, s) + w(s)$, where $t \in \mathrm{reg}(s)$.

Note that polygons are disjoint and therefore $d(s_i, s_j) \geq |w(s_i) - w(s_j)|$ for any two core elements s_i, s_j. Equality can only hold for neighboring core segments. As a result, core segments are entirely contained within their Voronoi cell and the combinatorial properties of $\mathcal{V}(G)$ are similar to the ordinary L_∞ Voronoi diagram of axis-parallel segments. Once $V(G)$ is computed the total critical area for breaks can be computed analytically as a summation of terms derived from Voronoi edges as shown in Section VI.

IV. Geometric Modeling of Via Blocks

Vias between different layers are typically realized by square shapes. To reduce the probability of missing contacts or to achieve a desired resistance, designers often use *redundant vias*, a group of multiple vias that connect the same shapes on different layers. Redundant vias are usually grouped together side by side and, thus, they can be regarded as a single via of larger size (see Fig. 13). Because of redundant vias, contacts can not be assumed to be squares but rectilinear shapes (often rectangles) of any size. A *via block* (for brevity, *block*) is a defect that completely destroys a contact, i.e., a defect that completely covers a whole via or a group of redundant vias [13], [25]. Since defects are assumed to be squares, a via block must totally cover the minimum enclosing rectangle of a contact. Note that a square can cover a rectilinear shape completely if and only if it totally covers the minimum enclosing rectangle of the shape. Note also that the minimum enclosing rectangles of two disjoint shapes need not be disjoint.

Redundant vias can be identified and be grouped together into atomic shapes using existing shape-processing tools such as [14] (see also [12]). Those shapes are disjoint and are referred to as *unified vias*. Unified vias can then be substituted by their minimum enclosing rectangle with no penalty on critical area calculation (assuming square defects). The resulting rectangular shapes will be referred to as *contacts*. Note that contacts, although unlikely in practice, may be nondisjoint. Since the underlying unified vias are disjoint, however, the intersection is restricted, i.e., no two contacts can cross. Thus, for critical area calculations, we can assume that a via layer is a collection of rectangles of various sizes. Those rectangles are in the majority disjoint but they may as well intersect in a restricted fashion, i.e., never crossing each other. Such rectangles will be referred to as *noncrossing*. A defect forms a *via block* if and only if it totally covers a rectangular contact.

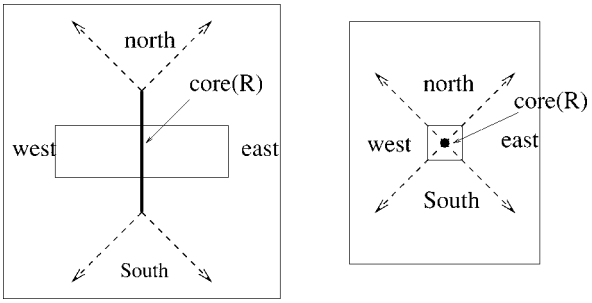

Fig. 14. Core of a contact R.

The *critical radius* of a point t on a via layer V is the radius of the smallest defect centered at t totally covering a contact. Let the max-*distance* of t from a contact R be the maximum L_∞ distance of t from any point of R, i.e., $d_{\max}(t, R) = \max\{d(t, y), \ \forall y \in R\}$. The max-distance between two contacts R_1, R_2 is $d_{\max}(R_1, R_2) = \max\{d(x, y), \ \forall x \in R_1, y \in R_2\}$. Clearly, the critical radius of t with respect to R is $d_{\max}(t, R)$. In the presence of many contacts, the critical radius of t is $r_c(t) = \min\{d_{\max}(t, R), \ \forall R \in V\}$. Under the max-distance function we can define the min–max Voronoi diagram of a set of rectangles as a subdivision of the plane into regions such that the Voronoi region of a rectangle R (if any), denoted as $\mathrm{reg}(R)$, is the locus of points $\{t\}$ with $d_{\max}(t, R) \leq d_{\max}(t, P)$ for any rectangle $P \neq R$. The min–max Voronoi diagram of a via layer reveals the critical radius for any point t: $r_c(t) = d_{\max}(t, R)$, such that $t \in \mathrm{reg}(R)$.

Let us now define the *core* of a contact R as a *generator* of via blocks for R. The *core* of R, denoted as $\mathrm{core}(R)$, is defined as the locus of centers of the minimum enclosing squares of R (see Fig. 14). It is easy to see that the core is an axis-parallel segment s of length $l - w$ centered at the same point as R, where l, w denote the length and the width of R ($l \geq w$). The core is a horizontal (resp. vertical) segment if the length of R is vertical (resp. horizontal) and is a single point if R is a square. The core is weighted by the radius of the min-enclosing square of R i.e., $w(s) = l/2$. The weighted distance of a point t to a core segment s is $d_w(t, s) = d(t, s) + w(s)$. Lemma 3 shows that $d_{\max}(t, R)$ is equivalent to the weighted distance of t from $s = \mathrm{core}(R)$.

Lemma 3: Given a rectangle R of length l and a point t, $d_{\max}(t, R) = d_w(t, s) = d(t, s) + w(s)$, where $s = \mathrm{core}(R)$ and $w(s) = l/2$.

Proof: Consider the 45° rays emanating from the endpoints of s. They are portions of the bisectors between adjacent edges of R (see Fig. 14) and they partition the plane into

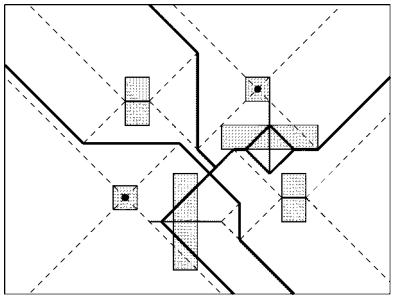

Fig. 15. Min–max L_∞ Voronoi diagram of rectangles is equivalent to the weighted L_∞ Voronoi diagram of core segments.

four quadrants. Let t be a point in the north quadrant. Then, $d_{\max}(t, R) = d(t, R^s)$, where R^s denotes the south edge of R. However, $d(t, R^s) = d(t, s^n) + l/2$, where s^n denotes the north endpoint of s. Hence, $d(t, R^s) = d(t, s) + w(s)$. This is similar for any point t in the remaining quadrants. □

By Lemma 3, the weighted Voronoi diagram of the set G of core segments is equivalent to the min–max Voronoi diagram of the corresponding rectangles. Fig. 15 illustrates the min–max Voronoi diagram of a set of rectangles; core segments are also depicted. The following theorem summarizes the above discussion.

Theorem 2: Given the L_∞ min–max Voronoi diagram of a set of rectangular contacts on a via layer, the critical radius for via blocks for any point t is $r_c(t) = d_{\max}(R) = d(t, s) + w(s)$, where $t \in \mathrm{reg}(R)$ and $s = \mathrm{core}(R)$.

In the remainder of this section, we list properties of the min–max Voronoi diagram of noncrossing rectangles. The following notation is used. Given a rectangle R, the north, south, east, and west edges of R are denoted as R^n, R^s, R^e, and R^w, respectively. The north, south, east, and west sides of the core segment $s = \mathrm{core}(R)$ are denoted as s^n, s^s, s^e, and s^w, respectively. The same notation is used to also denote the y-coordinate of R^s, R^n, s^s, and s^n and the x-coordinate of R^e, R^w, s^e, and s^w.

A. Min–Max L_∞ Voronoi Diagram of Rectangles

The min–max L_∞ Voronoi diagram of a set of rectangles is equivalent to the weighted Voronoi diagram of the set of core segments derived from the rectangles (Lemma 3). Thus, it is a special case of the L_∞ Voronoi diagram of weighted axis-parallel segments that was discussed in Section II. Rectangles are assumed to be noncrossing. The following Theorem 3 shows that Voronoi cells are connected and that the size of the min–max Voronoi diagram is linear in the number of rectangles. Each rectangle has at most one Voronoi cell. A necessary and sufficient condition for a rectangle to have a nonempty Voronoi region is given in Lemma 5. Core segments are entirely or partially contained within their Voronoi cell. The portion of a core segment within its Voronoi cell $s \cap \mathrm{reg}(s)$ is called *active* and its endpoints are called *active endpoints*; the remaining portion is called *inactive*. Active endpoints are determined as shown in Lemma 6. A rectangle having no Voronoi region is also called *inactive*.

Let the union of all minimum enclosing squares of a rectangle R be denoted as the *core rectangle* of R (see Fig. 16). Let the

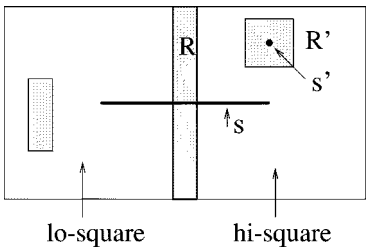

Fig. 16. Core rectangle of R containing interacting contacts.

lo-square denote the leftmost (resp. bottommost) minimum enclosing square and *hi-square* denote the rightmost (resp. topmost) minimum enclosing square; *core rectangle= lo-square* ∪ *hi-square*. For a square R, the core rectangle is identical to R. A rectangle is said to be *interacting* with R if and only if it is totally contained within the core rectangle of R. Since contacts cannot cross each other, an interacting rectangle must be contained either in the lo-square or the hi-square of R.

Lemma 4: The (weighted) bisector of any two core segments is connected and intersects the segment of larger weight at most once.

Proof: It is enough to consider a contact R such that there is a contact R' with $w(s) \geq w(s') + d(s, s')$, where $s = \mathrm{core}(R)$ and $s' = \mathrm{core}(R')$. Note that $d(s, s') = \min_{p \in s}\{d(p, s')\}$. Thus, there must be a point p along s such that $d_{\max}(p, R') = d(p, s') + w(s') \leq w(s)$. Then the square centered at p of radius $w(s)$ must entirely contain both R and R', i.e., R' must be interacting with R. Let us assume without loss of generality that s is horizontal and that R' lies in the hi-square of R (see Fig. 16). Then for any point $t \in s$, $d_{\max}(t, R) = d(t, R^n) = w(s)$. Let p be a point along s such that $d_{\max}(p, R) > d_{\max}(p, R')$. Since R' is assumed to be in the hi-square of R, $d_{\max}(t, R') \leq d_{\max}(p, R')$ for any point $t \in s$ to the right of p. Thus, $d_{\max}(t, R') \leq d_{\max}(p, R') < d_{\max}(p, R) = d_{\max}(t, R)$ for any point $t \in s$ to the right of p. Thus, the portion of s to the right of p is closer to s' than s. Hence, bisector $b(s, s')$ may intersect s at most once. Thus, $b(s, s')$ must be connected. □

Theorem 3: The min–max L_∞ Voronoi diagram of noncrossing rectangles consists of connected Voronoi cells, at most one for each rectangle, and has size linear in the number or rectangles.

Proof: By Lemma 4, any bisector of s intersects s at most once. Thus, the active portion of s, $s \cap H(s, s_j)$ must consist of a single connected component, if any (see Section II for notation). It is easy to see that for any point $t \in \mathrm{reg}(s)$ there must be an active point $p \in s$ such that segment $\overline{tp}$ is entirely contained within $\mathrm{reg}(s)$ ($d_w(t, s) = d(t, p) + w(s)$). Thus, $\mathrm{reg}(s)$ must be a connected region (if nonempty). The connectivity of Voronoi cells implies the linear size of the diagram similarly to the ordinary case. □

Lemma 5: A rectangle R has no Voronoi cell if and only if there are two rectangles interacting with R at opposite sides of R within max-distance l from each other, where l is the length of R.

Proof: Let $s = \mathrm{core}(R)$ be the core segment of R. If there are two rectangles R', R'' interacting with R at opposite sides

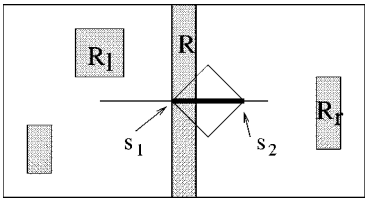

Fig. 17. Partially active core segment $s = \mathrm{core}(R)$.

of R within max-distance l from each other, then for any point $p \in s$, $d_{\max}(p, R) \geq \min\{d_{\max}(p, R'), d_{\max}(p, R'')\}$, i.e., s is entirely inactive. Thus, the Voronoi cell of R must be empty. Suppose now that R has no Voronoi cell. Then, for every point $p \in s$, there is some rectangle R' such that $d_{\max}(p, R') \leq d_{\max}(p, R)$. That is, for every $p \in s$, the minimum enclosing square of R centered at p must totally cover some rectangle $R' \neq R$. Thus, there must be at least one minimum enclosing square totally covering two distinct rectangles at opposite sides of R. □

Fig. 17 depicts the active portion of a horizontal core segment $s = \mathrm{core}(R)$ thickened; the left and right active endpoints are determined by edges R_l^w and R_r^e, respectively. In the following, s_1, s_2 denote the left and right (resp. bottom and top) active endpoints of s. Suffix $.x$ (resp. $.y$) denotes x-coordinate (resp. y-coordinate).

Lemma 6: A core segment $s = \mathrm{core}(R)$ is entirely active if and only if there are no rectangles interacting with R. The active endpoints of a horizontal (resp. vertical) core segment s are determined as follows: $s_1.x = \max\{R_j^w | R_j \in lo\text{-}square(R)\} + w(s)$ and $s_2.x = \min\{R_j^e | R_j \in hi\text{-}square(R)\} - w(s)$ (resp. $s_1.y = \max\{R_j^s | R_j \in lo\text{-}square(R)\} + w(s)$ and $s_2.y = \min\{R_j^n | R_j \in hi\text{-}square(R)\} - w(s)$).

Proof: Core segment s is entirely active if and only if s lies entirely in the half-plane $H(s, s_j)$ for any other core segment s_j. This is the case if and only if $d_w(s, s_j) \geq w(s) - w(s_j)$, i.e., iff the core rectangle is empty. Let us assume without loss of generality that s is horizontal (see Fig. 17). Then, s_1 must be equidistant from R^n and R_j^w for some contact R_j in the lo-square of R. Thus, s_1 must be equidistant from R^n and R_l^w, where R_l denotes the rectangle with the rightmost west edge ($R_l^w = \max\{R_j^w | R_j \in lo\text{-}square(R)\}$). Note that $d_{\max}(s_1, R) = d_{\max}(s_1, R_l)$. Similarly for s_2 and for a vertical core segment. □

V. Computing the L_∞ Voronoi Diagram of Core Segments

Plane sweep algorithms to compute the L_∞ Voronoi diagram of rectangles and arbitrary line segments were given in [16] and [17], respectively. Here, we briefly review the main points of the algorithm. Imagine a sweepline L sweeping the layout from left to right. Associated with a plane-sweep algorithm, there are two major components: a *sweep-line status* $\mathcal{T}$ maintaining the status of the sweeping process and an *event list* Q containing the events where the sweep-line status changes. Q is ordered in increasing *priority* value. Throughout the sweeping process, a partial Voronoi diagram of all segments to the left of the sweeping line, including the sweepline, is maintained. The *wavefront* is the collection of Voronoi edges (portions of bisectors) bounding the Voronoi cell of the sweepline. The bisectors incident to the wavefront are called *spike bisectors*. As the sweepline moves to the right the wavefront as well as the endpoints of spike bisectors also move to the right. The *sweep-line status* maintains the combinatorial structure of the wavefront and it is implemented as a *height-balanced* tree; details on a data structure are given in [5]. The event list is implemented as a *priority queue*. We have two types of events: *site events* and *spike events*. Site events are caused by the endpoints of segments and correspond to new waves entering the wavefront. Spike events correspond to potential Voronoi vertices and are determined by the intersection of two neighboring spike bisectors. At a spike event a wave gets deleted from the wavefront. Each event has a *priority* value that determines the order in which an event is processed. The priority of a site event is given by its x-coordinate. That is, a site event is processed as soon as it is reached by the sweepline. The priority of a spike event q is $q.x + d(e, q)$, where e is the owner of q.[6] In other words, a spike event is not processed as soon as it is reached by the sweepline, but later when the sweepline reaches position $q.x + d(e, q)$, i.e., when the sweepline reaches the right side of the square centered at q having radius $d(e, q)$. The number of events throughout the sweeping process is $O(n)$, where n is the total number of line segments. The total time complexity of the algorithm is $O(n \log n)$. For details, see [16] and [17].

In this paper, we compute the weighted L_∞ Voronoi diagram of weighted axis-parallel segments. In the case of breaks, the weights are such that $d(s_i, s_j) \geq |w(s_i) - w(s_j)|$ for any two segments s_i, s_j. As a result, we only need to perform minor modifications to the algorithm outlined above. The main modification is that the weight of a core element needs to be added to its priority. That is, a core point p is not processed as soon as it is reached by the sweep line, but later when the sweep line reaches position $p.x + w(p)$. This is the time when L reaches the right edge of the square induced by p. When bisectors are computed the weights of core elements are always added to the equations. Due to the restriction on the weights, the invariance of the plane sweep paradigm for Voronoi diagrams [4] is still valid: any site with priority higher than the current sweepline position lies ahead of the sweepline. The core segments themselves can be computed by plane sweep as shown in [16] as they are portions of the medial axis of shapes. Since the priority of a core point is the same as the priority of a Voronoi vertex in the original algorithm, core segments can be computed on the fly as the plane sweep proceeds.

In the case of the min–max Voronoi diagram, the above invariance no longer holds: a core segment with priority higher than the current sweepline position may be partially covered or even entirely covered by the wavefront. This is caused by core segments that are partially or entirely inactive. Additional active events need to be generated to identify the active endpoints of horizontal core segments. These active events are responsible for the extra K factor in the time complexity of the algorithm. In summary, we have four types of events: *active*, *left*, *right*, and *spike* events. An *active* event corresponds to a potential left active endpoint of a horizontal core segment and

[6] Suffix $.x$ denotes x-coordinate.

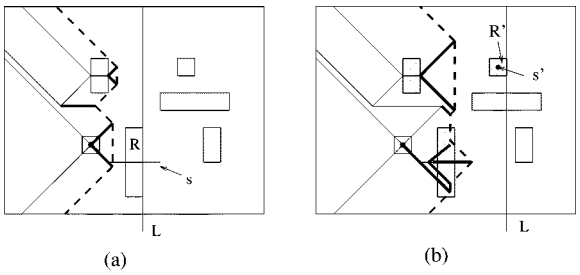

Fig. 18. Wavefront at instances. (a) $t = priority(s^w) = R^e$. (b) $t' = priority(s'^w) = R'^e$.

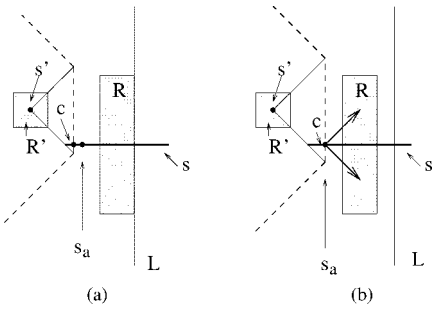

Fig. 19. (a) Generation of an active event s_a at time $t = priority(s^w) = R^e$. (b) Handling of an active event s_a at time $t = priority(s_a) = s_a.x + w(s)$.

has priority $s_a + w(s)$, where s_a denotes both the potential active endpoint and its x-coordinate. A left event and a right event correspond to the occurrence of s^w and s^e, respectively, for $s = \text{core}(R)$ and have priority $priority(s^w) = s^w + w(s) = R^e$ and $priority(s^e) = s^e + w(s) = R^w + 2w(s)$ (see Section IV for notation). That is, the priority of a left event is reached when the sweep line reaches the right edge of a rectangle and the priority of a right event is reached when the sweepline reaches the right edge of *hi-square*. Note that $s^i + w(s) = R^j$ for $i = s, w$ and $j = n, e$, respectively, and $s^i - w(s) = R^j$ for $i = n, e$ and $j = s, w$, respectively. A spike event corresponds to the intersection of two neighboring spike bisectors and has the same priority as in [16]. Spike events are treated similarly to [16] and, thus, they are not discussed in this paper.

Fig. 18 illustrates the status of the min–max Voronoi diagram construction at two instances: $t = priority(s^w) = R^e$ and $t' = priority(s'^w) = R'^e$; the wavefront is depicted in dashed lines. Horizontal core segments incident to the wavefront are called *spike* core segments. Recall that the bisectors incident to the wavefront are called *spike* bisectors. In Fig. 18, spike bisectors and spike core segments are shown thickened. Note that a spike core segment is essentially a portion of the bisector $b(R^n, R^s)$. At any instance of the sweeping process, the wavefront is the polygonal line connecting the endpoints of spike bisectors and spike core segments in increasing y-coordinate. A *wave* is a segment along the wavefront. The wavefront can never touch the sweepline since the weights of core segments are always nonzero. The combinatorial structure of the wavefront is maintained in the sweepline status $\mathcal{T}$. In particular, $\mathcal{T}$ maintains one node for each *wave*. The endpoints of a wave are derived by the endpoints of the incident spike bisector or spike core segment. A node also keeps information for the bounding spike bisectors or core segments. The *owner* of a node is an element of a weighted core segment (i.e., s^j, $j = e, w, n, s$) or equivalently the corresponding original edge of the rectangle (i.e., R^j, $j = w, e, s, n$).

Let us now discuss how to determine the active portions of core-segments. This is essentially the novel portion of the algorithm. Recall that active events are generated to determine the left active endpoint of horizontal core segments. As will be evident in the sequel, no special attention needs to be given to the determination of right active endpoints. The following lemmas show how the active portions of core segments can be derived. Lemma 9 is used in establishing correctness of the algorithm.

Lemma 7: The active portion of a vertical core segment $s = \text{core}(R)$ is given by the intersection (if any) of s with the wavefront at instance $t = priority(s^w) = s^w + w(s) = R^e$. If s lies ahead of the wavefront, then s is totally active; if s is totally covered by the wavefront, then R must be inactive.

Proof: At instance $t = priority(s^w)$, for any point p along s, $d(p, L) = d_{\max}(p, R) = w(s)$. The lemma follows by the definition of the wavefront at instance t. □

Lemma 8: The left and right active endpoints of a horizontal core segment $s = \text{core}(R)$ (if different than s^e and s^w, respectively) are given by the intersection of s with the wavefront at instances $t_l = R_l^w + 2w(s)$ and $t_r = R_r^e$, respectively, where R_l denotes the contact in the $lo\text{-}square$ of R with the rightmost west edge (if any) and R_r denotes the contact in the $hi\text{-}square$ of R with the leftmost east edge (if any).

Proof: The lemma is derived by the definition of the wavefront and Lemma 6. Note that the priority of the active endpoints s_1 and s_2 is $priority(s_i) = s_i.x + w(s)$, $i = 1, 2$. □

Lemma 9: At any instance between $priority(s^w)$ and $priority(s^e)$, $s = \text{core}(R)$, any wave overlapping s is induced by a contact interacting with R.

Proof: For a vertical s, this is clear by the definition of the wavefront and the core rectangle (note $priority(s^w) = priority(s^e) = s.x + w(s)$). For a horizontal s, let c be the intersection point of s and the wavefront at any instance t, $priority(s^w) \le t \le priority(s^e)$. Then, $d(c, L) = d_{\max}(c, R_j)$, where R_j is the owner of c in $\mathcal{T}$ and L is the sweeping line. However, $d(c, L) \le w(s)$ since $priority(c) \ge t$. Thus, R_j must be totally contained within the square centered at c of radius $w(s)$, i.e., R_j must be interacting with R. □

Let us now discuss how an active event is generated and handled (see Fig. 19). The purpose is to identify R_l as defined in Lemma 8. Let $s = \text{core}(R)$ be a horizontal core segment such that s^w is not active. Then, at instance $t = priority(s^w) = R^e$, s must be already (partially) covered by the wavefront. Let R' be the rectangle whose wave in $\mathcal{T}$ intersects s. R' must be a rectangle interacting with R and it may or may not be R_l. We say that R' is the *owner* of the intersection $s \cap wavefront$. At this time, we generate an active event s_a induced by R' having priority $priority(s_a) = R'^w + 2w(s)$ [see Fig. 19(a)]. The active event is processed when its priority is reached. Consider the intersection of s with the wavefront at time $t = priority(s_a)$, i.e., $c = s \cap wavefront$. If the owner of the intersection is R', then the active event is *valid* and reveals the left active endpoint of s [$R' = R_l$, see Fig. 19(b)]. Otherwise, the active event is *invalid* ($R' \neq R_l$). If the active event is invalid, either a new ac-

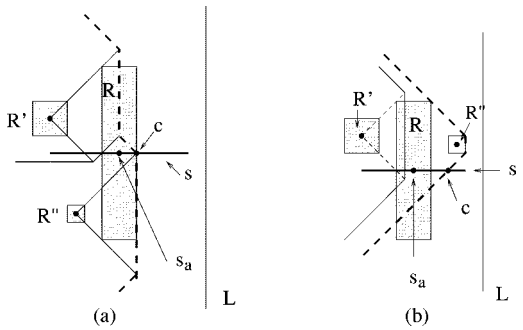

Fig. 20. Invalid active events at instance $t = priority(s_a)$. (a) $R' \neq R_l$. (b) s is inactive.

tive event is generated or s is determined to be totally inactive as follows. Let R'' be the owner of the intersection c. If R'' is in the hi-square of R, then R must be totally inactive: no point of s can be active (Lemma 5). Otherwise, R'' must be in the lo-square of R with $R''^w > R'^w$. In this case a new active event for s is generated, induced by R'' (R'' is the new candidate for R_l). Fig. 20 depicts invalid active events. No special care needs to be taken for the identification of the right active endpoint of s. It is determined by the normal expansion of the wavefront at instance $t = priority(R_r) = R_r^e$, where R_r is as defined in Lemma 8.

In the following, we give pseudocode for the handling of *left*, *right*, and *active* events. Spike events are handled similarly to [16]. Suffix $.x$ denotes the x-coordinate of a point. The reader is referred to [16] for details on the basic plane-sweep algorithm. We start with a left event for contact R having priority $t = priority(s^w) = s^w + w(s) = R^e$. Explanatory remarks follow the pseudocode.

Procedure *Process-Left-Event* $(R, s = \text{core}(R))$
begin
1. $a' =$ intersection $(wavefront, b(R^s, R^e))$;
2. $b' =$ intersection $(wavefront, b(R^n, R^e))$;
3. **if** (s is horizontal) AND ($a'.x > s^w$ OR $b'.x > s^w$) **then**
4. $c =$ intersection $(wavefront, s)$;
5. $q =$ *CreateActiveEvent*(c, R);
6. $push(q, Q)$;
7. **Return**;
8. **if** (s is vertical) AND ($a'.x > s^w$) **then** $a' =$ upper-intersection $(wavefront, s)$;
9. **if** (s is vertical) AND ($b'.x > s^w$) **then** $b' =$ lower-intersection $(wavefront, s)$;
10. **if** no intersection was detected at steps 8, 9 **Return**; (*s is totally covered by wavefront; R is inactive*)
11. $W = \{w | w =$ segment of the wavefront between a' and $b'\}$;
12. Initialize $reg(s)$ to W;
(* $reg(s)$: Voronoi region of s*)
13. Update $reg(s')$ $\forall\, s' \in W$;
14. Update $\mathcal{T}$ by deleting nodes between a' and b';

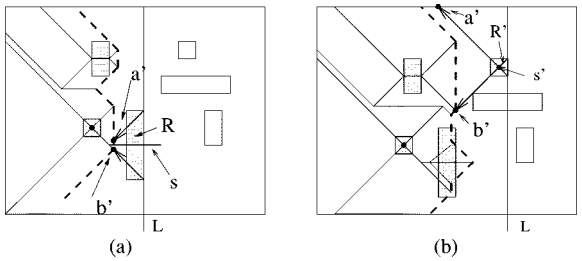

Fig. 21. Processing a left event. (a) Generate active event. (b) Entirely active core element.

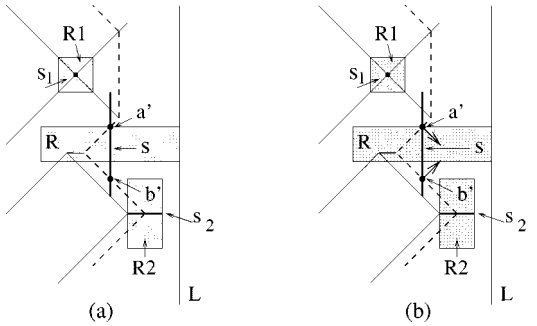

Fig. 22. Partially active vertical core segment. (a) Handling the left event. (b) Handling the right event.

15. if $a' \notin s$ **then** (*see [6] *)
16. $b_1 = b(e_a, s^n)$; (*$e_a =$ owner of a'*)
17. Insert to $\mathcal{T}$ b_1 and s^n; (* the weighted s^n corresponds to unweighted R^s *)
18. $q_1 =$ intersection $(b_1, prev(b_1))$;
19. Insert in Q a spike event corresponding to q_1;
20. If $b' \notin s$ **then** (*see [16] *)
21. $b_2 = b(e_b, s^s)$; (*$e_b =$ the owner of b'*)
22. Insert to $\mathcal{T}$ b_2 and s^s; (* the weighted s^s corresponds to unweighted R^n *)
23. $q_2 =$ intersection $(b_2, next(b_2))$;
24. Insert in Q a spike event corresponding to q_2;

end

In Steps 1 and 2, the intersection points a', b' of bisectors $b(R^n, R^e)$ and $b(R^s, R^e)$ with the wavefront are determined (see Fig. 21). Note that the 45° rays emanating from the endpoints of s^w backward are portions of those bisectors. If both a', b' are to the left of s^w [see Fig. 21(b)], then the wavefront does not overlap s and we basically have the ordinary case. Otherwise, we generate an active event for a horizontal core segment [Steps 4–6, see Fig. 21(a)] or we determine the active portion of a vertical core segment (if any) by computing the intersection of the wavefront with s [Steps 8–10, see Fig. 22(a)]. In Step 11, the portion of the wavefront W between a' and b' is traced (see [16] for details). At this step, the endpoints of spike bisectors and active core segments between a' and b' are determined. All Voronoi regions adjacent to W are updated in Step 13. The remaining steps are similar to the handling of a *left-edge*

event shown in [16] and, thus, we skip the details. $prev(b_1)$ [resp. $\text{next}(b_2)$] indicate the bisector in $\mathcal{T}$ preceding b_1 (resp. following b_2). Note that in step 16 (resp. 21), instead of computing a weighted bisector involving s^n (resp. s^s) we can instead compute the ordinary bisector involving R^s (resp R^n).

Lemma 10: All intersection functions in *process-left-event* can be implemented by binary search in $O(\log|WF|)$ time, where $|WF|$ is the size of the wavefront.

Proof: For Steps 1 and 2 , this was shown in [16]. This is clearly the case for the intersection of a horizontal core segment and the wavefront. The intersection of a vertical core segment with the wavefront can also be determined by binary search in $\mathcal{T}$ due to the following observation. The upper (resp. lower) intersections must be generated by a contact in the hi-square (resp. $lo\text{-}square$) of R. Moreover, both intersections (if any) have to lie between a' and b'. Given a wave between a' and b' induced by a contact R', we do the following: if both endpoints of the wave are to the left of s, then we have to move up in $\mathcal{T}$ to determine the upper intersection or move down in $\mathcal{T}$ to determine the lower intersection. If both endpoints are to the right of s, then we move up in $\mathcal{T}$, if R' is in the $lo\text{-}square$ of R and move down in $\mathcal{T}$, if R' is in the $hi\text{-}square$ of R. Otherwise, we have determined one intersection. Thus, all the above functions *intersection* can be implemented by binary search in $\mathcal{T}$. □

In the following, we give pseudocode for a right event s^e. When the priority of s^e is reached the wavefront may already be covering s^e. In this case, s^e is an *invalid* event and must be discarded. This would be the case for a totally inactive s or for a partially active horizontal s whose right active endpoint is not s^e. To determine whether s^e is valid, we do the following. For a horizontal core segment we consider the intersection c of a horizontal line through s and the wavefront: s^e is valid if and only if c coincides with s^e. For a vertical core segment, this is determined at Step 10 of the left-event procedure: s^e is invalid iff s is totally covered by the wavefront. The processing of a valid right event is similar to the processing of a right-edge event in [16]. Fig. 22(b) illustrates the handling of a valid right event induced by a partially active vertical core segment.

Procedure *Process-Right-Event* $(R, s = \text{core}(R))$
begin
1. **if** s_e is invalid **then**
2. **Return**; (* the wavefront covers s_e*)
3. **if** s is horizontal **then**
4. $a' = s^n$;
5. $b' = s^s$;
6. **else**
7. a'= upper active endpoint (* a' may be s^n *)
8. b'= lower active endpoint (* b' may be s^s *)
9. $b_1 = b(e_a, s^e)$; (*e_a = the owner of a' *);
10. $b_2 = b(e_b, s^e)$; (*e_b = the owner of b' *)
11. Update the nodes of e_a, e_b by inserting b_1, b_2 in $\mathcal{T}$ and deleting s^n, s^s if present;
12. Insert the node of s^e in $\mathcal{T}$ between e_a, e_b; (*the weighted s^e is equivalent to R^w *)
13. q_1 = intersection $(b_1, prev(b_1))$;
14. q_2 = intersection $(b_2, next(b_2))$;
15. Insert in Q spike events corresponding to q_1, q_2;
end

The following is pseudocode for the handling of an active event s_a. Event s_a is valid if and only if s_a coincides with the intersection c of the wavefront with the horizontal line through s at priority $priority(s_a) = s_a.x + w(s)$ [see Fig. 19(b)]. If s_a is invalid, we either generate a new active event induced by the owner of c (Step 4) or determine that R is a totally inactive as explained above. To summarize, at an active event s_a we do the following:

Procedure *Process-Active-Event* (s_a)
begin
1. c = intersection $(wavefront, s)$;
2. **if** s_a is invalid **then**
3. **if** s is not totally inactive **then**
4. q = *CreateActiveEvent* (c, R);
5. $push(q, Q)$;
6. **Return**;
7. $e_a = e_b$ = the owner of c in $\mathcal{T}$.
8. **if** c coincides with the endpoint of a spike bisector **then**
9. e_a = upper owner of c;
10. e_b = lower owner of c;
11. $b_1 = b(e_a, s^n)$;
12. $b_2 = b(e_b, s^s)$;
13. Initialize $reg(s)$ to $b_1 \cup b_2$;
14. **if** $e_a = e_b$ **then**
15. split the node of e_a into two nodes bounded by b_1, b_2; (* e_b is owner of the lower node*)
16. Insert nodes for s^n, s^s between the nodes of e_a, e_b in $\mathcal{T}$;
17. q_1 = intersection $(b_1, prev(b_1))$;
18. q_2 = intersection $(b_2, next(b_2))$;
19. Insert in Q spike events corresponding to q_1, q_2;
end

Fig. 19(b) illustrates the handling of a valid active event s_a. In the figure, $e_a = e_b = s'^e$ (equivalently $e_a = e_b = R'^w$). The new spike bisectors b_1 and b_2 are shown in arrows. Note that b_1 can be defined either as the weighted bisector of s'^e and s^n or equivalently as the ordinary bisector of R'^w and R^s and is similar for b_2. The node of s'^e (or equivalently, the node of R'^w) is split into two and the nodes of s^n, s^s (or equivalently, the nodes of R^s, R^n) are inserted in between.

The plane sweep proceeds until all the events are processed. Correctness and time complexity is established in the following theorem. Let K denote the total number of active events. By

Lemma 9, K is upper bounded by number of interacting rectangles. In practice, contacts are well spaced and the number of interacting contacts is rather small. Hence, on any typical via layer, K is not significant and can be ignored (see, for example, Table I in Section VII).

Theorem 4: The min–max L_∞ Voronoi diagram of n noncrossing rectangles can be computed by plane sweep in $O((n+K)\log n)$ time, where K is upper bounded by the number of interacting rectangles.

Proof: Since the correctness of the basic plane sweep algorithm was established in [16], it is enough to show that the active portions of core segments are correctly determined. By Lemma 7, this is clear for a vertical core segment (see Steps 8–10 in procedure process-left-event). For a horizontal core event $s = \text{core}(R)$, active events are produced at Step 5 of procedure process-left-event and Step 4 of procedure process-active-event. By Lemma 9, each active event is induced by the west edge of an interacting rectangle in the lo-square of R. Note that at instance $t = priority(s^w)$, the rectangle R_l inducing this actual active event need not even be in the wavefront yet; thus, a number of active events may need to be generated until R_l is determined. Each active event for s is generated by a distinct interacting contact. Thus, the total number of active events generated K cannot exceed the number of interacting contacts. The right active endpoint of a horizontal core segment is determined normally by the left event induced by R_r (see Lemma 8).

Any operation in the above procedures to handle events requires $O(\log |WF|)$ time (see Lemma 10). Thus, every event can be processed in time $O(\log |WF|)$. The total number of left, right, and spike events is $O(n)$ as in the ordinary case (see [16]). Thus, the total time complexity of the algorithm is $O((n+K)\log n)$. □

VI. Computing the Critical Area Integral

Once the L_∞ weighted Voronoi diagram of core segments $\mathcal{V}(G)$ for either breaks or via blocks is available the critical area integral can be computed similarly to [16]. In particular, the critical area integral can be discretized as a summation of terms derived from Voronoi edges. The computation is performed within the boundary of the layout which is assumed to be a rectangle B, often the bounding box of the layout. We assume that the fine version of $\mathcal{V}(G)$ is available. That is, the active portions of core segments as well as the 45° rays emanating from their endpoints and boundary edges are considered to be part of $\mathcal{V}(G)$ and are treated as Voronoi edges. In case of degenerate bisectors, as the one depicted in Fig. 4(g), a core segment may be collinear with a Voronoi edge. We adopt the convention that the active portion of a core segment is only the one lying strictly in the interior of the Voronoi cell; the portion coinciding with a Voronoi edge is considered inactive and it is ignored. Under these assumptions, each Voronoi cell is a cycle with exactly one edge or one vertex corresponding to an active core segment or a core endpoint. The core segment or point is considered to be the *owner* of the cell. Note that a core point is equivalent to an axis parallel segment depending on which quadrant is under consideration.

For the discretization of the critical area integral, Voronoi edges are classified into red and blue. Red edges have positive contribution to critical area while blue edges have negative contribution. The classification of an edge as red or blue is made with respect to a single cell. Since a Voronoi edge bounds two neighboring cells, it may get different coloring with respect to each cell. The contribution of such an edge simply cancels out and is assumed to be colored *neutral*. Voronoi edges of zero contribution to critical area correspond to degenerate bisectors. The criteria under which an edge is classified into red or blue is given in [16]. The classification can be adapted for the L_∞ weighted Voronoi diagram of core segments as the following.

1) All edges corresponding to active core segments are colored red.
2) Axis parallel Voronoi edges are colored blue with respect to a cell as long as they are not collinear with the owner of the cell; otherwise, they are colored red.
3) Boundary edges parallel to the owner are colored blue. Boundary edges perpendicular to the owner are ignored, i.e., they are colored neutral.
4) A 45° Voronoi edge is called *converging* with respect to a neighboring Voronoi cell if the underlying 45° line forms an acute angle with the axis-parallel line through the owner of the cell (as seen form the interior of the cell); otherwise , it is called *diverging*. Diverging 45° Voronoi edges are colored red and converging 45° Voronoi edges are colored blue.

In the following, we only state the final result and refer the reader to [16] for the proof. An edge receiving different coloring with respect to its neighboring cells is assumed to be colored neutral. Active core segments are included in the first summation of the following formula. Boundary edges are only considered in the last summation. Note that the critical radius of any Voronoi point p is $d(p, s) + w(s)$, where s is the owner of the cell. The critical radius of an active core segment s is clearly $w(s)$.

Theorem 5: Given the (weighted) L_∞ Voronoi diagram of all core elements of shapes in a layer of a circuit layout and assuming that defects are squares following the "r_0^2/r^3" defect density distribution, the critical area for breaks or via blocks in that layer is given by the following formula:

$$A_c = r_0^2\left(\sum_{\text{red } e}\frac{l}{r_c} - \sum_{\text{blue } e}\frac{l}{r_c} + \sum_{\text{red } e_{45}}\ln\frac{r_{\max}}{r_{\min}} - \sum_{\text{blue } e_{45}}\ln\frac{r_{\max}}{r_{\min}} - \frac{1}{2}\sum_{\text{blue } b}\frac{l_b}{r_b}\right)$$

where

l and r_c	length and critical radius of an axis-parallel Voronoi edge;
$r_{\max}$, $r_{\min}$	maximum and minimum critical radius of a 45° Voronoi edge;
l_b, r_b	length and critical radius of a boundary edge.

The first two summations are taken over all red and all blue axis-parallel Voronoi edges, respectively, including active core

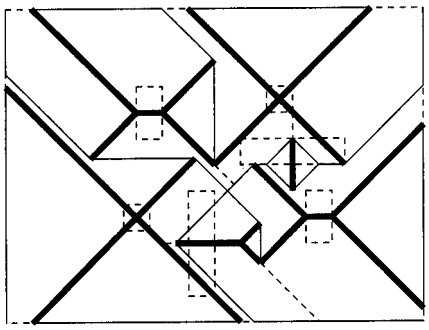

Fig. 23. Coloring of Voronoi edges for critical area calculation. Thick lines depict red edges and normal lines depict blue edges. Dashed lines do not participate in critical area calculation.

segments. The third and forth summations are taken over all red and all blue 45° Voronoi edges, respectively. The last summation is taken over all blue boundary edges.

Fig. 23 illustrates the coloring of the min–max Voronoi diagram of Fig. 15. Red edges are depicted thickened while blue edges are depicted thin. Neutral edges as well as the inactive portions of core segments are depicted in dashed lines. Note that the L_∞ Voronoi diagram of segments is not unique due to the presence of degenerate configurations. The exact Voronoi diagram depends on the convention one adopts to partition equidistant regions between owners. The critical area result, however, is the same no matter which version of the Voronoi diagram is used. Note that the critical radius of a point within a "grey" region equidistant from two elements is the same no matter which element is assumed to be the owner of the region.

In the remainder of this section, we show a worst case bound between critical area in the Euclidean and the L_∞ metric. First, we need to give a mathematical definition for a break in the Euclidean metric. As we showed in Section III, the medial axis of a shape provides the means to formalize the intuitive concept of a defect *breaking* a shape into two pieces. In the Euclidean metric, we define the *core* of a shape P to be the portion of the medial axis induced by nonconsecutive edges in P. Every point p along the core is *induced* by a pair of boundary points (u, v) such that $w(p) = d_e(p, P) = d_e(p, u) = d_e(p, v)$, where $d_e(p, P)$ denotes the Euclidean distance of p from the boundary of P. Any core point p is weighted by $w(p)$. A *minimal break* is a circle D centered at any core point p of radius $w(p)$. We say that D is *generated* by p. A *break* is any circle totally covering a minimal break. The *critical radius* of a point t is the radius of the smallest break B centered at t; $r_c(t) = d(t, p) + w(p)$, where p is the core point generating the minimal break covered by B. Note that if p is an internal core point, $r_c(t) = \max\{d_e(t, u), d_e(t, v)\}$, where (u, v) is the pair of boundary points inducing p.

Property 1: Assuming that $D(r) = r_0^2/r^3$, the critical area for breaks and via blocks in the L_∞ metric, A_c^∞, is bounded by the Euclidean critical area A_c^e with the following relation:

$$A_c^e \le A_c^\infty \le 2A_c^e.$$

Proof: Let $r_e(t)$ and $r_\infty(t)$ denote the Euclidean and the L_∞ critical radius for breaks at point t, respectively. Let B_e and B_∞ denote the minimum size Euclidean and L_∞ break at t, respectively. Let D_e be the minimal break totally covered by B_e and let p_e denote the core point at the center of D_e. Consider the square S circumscribing D_e. S must clearly be an L_∞ break. Thus, $r_\infty(t) \le d(t, p) + w(p) \le d_e(t, p) + w(p) = r_e(t)$. Consider now the circumcircle of B_∞, D. D must clearly be a Euclidean break since it must overlap with all boundary points that B_∞ is overlapping with. Thus, $r_e(t) \le \sqrt{2} r_\infty(u)$.

Now let $r_e(t)$ and $r_\infty(t)$ denote the Euclidean and the L_∞ critical radius for via blocks at point t of a via layer, respectively. Let $d^e_{\max}(t, R)$ denote the Euclidean max distance of point t from a rectangle R. Clearly, $d_{\max}(t, R) \le d^e_{\max}(t, R) \le \sqrt{2} d_{\max}(t, R)$. Thus, $\sqrt{2} r_\infty(t) \ge r_e(t) \ge r_\infty(t)$.

Now the critical area bound for both cases can be derived by integration as shown in [16]. □

TABLE I
RUNNING TIMES ON IBM 7043 43P-150

Macros	Layer	#Rectangles	K	Time
BRUBS	V1	27,514	0	16.43 secs
	V2	10,142	0	6.14 secs
	V3	8,264	0	4.95 secs
FPU A	V1	104,912	0	1:11 mins
FPU B	V1	261,919	0	3.06 mins
	V2	237,696	231	2.47 mins
	V3	157,413	158	1.52 mins
i-cache	V1	440,292	1602	5.01 mins
	V2	27,259	0	23.28 secs
	V3	2,675	0	5.70 secs

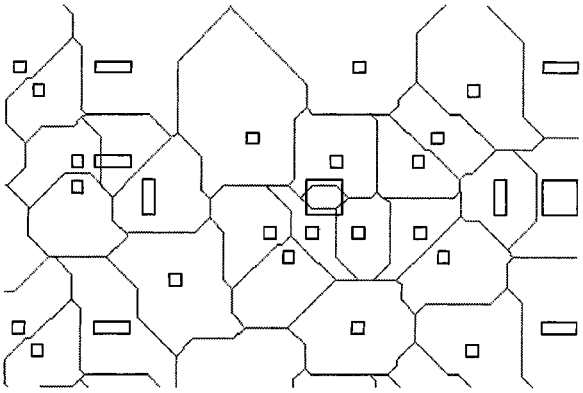

Fig. 24. Min–max Voronoi diagram on a via layer.

VII. EXPERIMENTAL RESULTS

We have implemented the plane sweep algorithm to compute the min–max Voronoi diagram of rectangles and we have integrated it in a new tool, under development, to compute critical area for shorts, via blocks, and breaks based on L_∞ Voronoi diagrams. We ran this tool on via layers of a variety of IBM macros (units of Power PC designs) to compute critical area for via blocks. Table I summarizes the running times of our experiments on an IBM 7043 43P-150 workstation, including the overhead of refining design data, and verifies the almost linear performance of the algorithm. Fig. 24 depicts a small piece of the min–max Voronoi diagram of contacts on a via layer.

We also performed comparisons between the results of our tool and CAA, the IBM tool for critical area analysis and yield prediction [3]. CAA offers two methods for critical area extraction: CAA-DotThrow and CAA-Expand. CAA-DotThrow

TABLE II
COMPARATIVE CRITICAL AREA RESULTS BETWEEN THE VORONOI DIAGRAM FRAMEWORK AND IBM-CAA

	Voronoi				CAA-Expand			CAA-DotThrow		
Macro	POF	Time	#Vias	error	POF	Time	Calc-error	POF	Time	Calc-error
BRUBS	0.000424	0:16	27,514	0%	0.000448	1:42	7.67%	0.000309	0:19	3.65%
	(0.000318)				0.000428	4:09	0.91%	0.000303	2:03	0.73%
FPU A	0.000366	1:11	104,912	0%	0.000382	6:31	6.60%	0.000275	1:27	2.30%
	(0.000274)				0.000371	15:40	0.84%	0.000272	2:36	1.86%
FPU B	0.000419	3:06	261,919	0%	0.000443	18:09	6.63%	0.000309	4:04	4.89%
	(0.000314)				0.000423	49:37	0.82%	0.000297	10:46	0.82%
i-cache	0.000549	5:01	440,292	0%	0.000580	29:45	2.90%	0.000407	0:22	3.10%
	(0.000411)				no completion in reasonable time			0.000395	5:26	0.72%

is based on Monte Carlo simulation [25] and it is enhanced by an adaptive integration method [1] that significantly speeds up the critical area computation by reducing the number of defects to be thrown. CAA-Expand falls into the category of geometric methods and computes $A(r_i)$ for a list of defect sizes that are then used to approximate the integral. CAA-Expand has been enhanced by an incremental method to compute $A(r_i)$ based on $A(r_{i-1})$ [15] and the adaptive integration technique that reduces the number of defect sizes considered. The two methods assume different defect types which creates a gap in the numeric values they produce. CAA-DotThrow approximates circles by regular octagons: a defect of size r is an octagon of diagonal r, i.e., an octagon *inscribed* to a circle of diameter r. As a result, the critical area calculated by CAA-DotThrow, even if perfect, is always a *lower bound* to the critical area of circular defects. CAA-Expand assumes square defects; in particular, a defect of size r is a square of width r, i.e., a square *circumscribed* to a circle of diameter r. Thus, the result of CAA-Expand as well as the Voronoi method is always an *upper bound* to the Euclidean critical area. This explains the gap between the numeric values produced by the two methods in Table II.

Table II illustrates comparative results for via blocks on the V1 layer of recent IBM macros of different design styles. The probability of fault (POF) is derived as the fraction of critical area over the total layout area given by the bounding box of the macro. CAA was run under two specified error bounds: 10% (first line) and 1% (second line). An upper bound of 200 000 defects was specified to allow CAA-DotThrow to finish in a timely fashion. Only in one case (line 4) CAA-DotThrow was not able to finish within the specified error bound. The calculated-errors are based on internal CAA calculations and we have not checked their accuracy. The Voronoi-based method has no error. The value in parentheses is a rough estimate of the Euclidean critical area derived from the L_∞ value as explained below. Time is reported as minutes to seconds.

The exact numeric value of the probability of fault is not as important as the ability to perform comparisons of critical area as well as the ability to quantify changes for yield in the design. The circle is already an approximation to the irregular shape of an actual defect; thus, a square, an octagon, or any other regular n-gon is also a good model. Note that each polygonal defect shape basically corresponds to a different distance metric (see, for example, [26]). If for any reason one wishes to derive a value closer to the Euclidean critical area, one could report the average between the critical area values of defects inscribed and circumscribed to the unit circle. In the following, we argue that $3/4\,A_c^\infty$ can serve as a rough approximation to A_c^e.

Let the L_∞^* distance between two points p, q denoted as $d^*(p, q)$ be defined as the length of the diagonal of the smallest square touching the two points. Then, $d^*(p, q) = \sqrt{2}d(p, q)$. The unit disks of the L_∞^* and the L_∞ metrics are squares inscribed and circumscribed respectively to the unit circle. Modeling a defect of size r as a square of diagonal $2r$ corresponds to computing critical area in the L_∞^* metric, denoted as $A_c^{\infty *}$. Then, $A_c^\infty = 2A_c^{\infty *}$ (see Property 2). Note that the difference between the L_∞ and the Euclidean distance is maximized for points, where the L_∞^* equals the Euclidean distance and vice versa. Thus, $A_c^* = (A_c^\infty + A_c^{\infty *})/2 = 3/4\,A_c^\infty$ can serve as a rough approximation to the Euclidean critical area. The values in parenthesis in Table II show A_c^*.

Property 2: $A_c^\infty = 2A_c^{\infty *}$.

Proof: Under the L_∞^* metric, the critical radius of any point t on a layer is $r_c^*(t) = \sqrt{2}r_c(t)$. As shown in [16], the critical area integral can be written as $A_c = (r_0^2/2)\int_{t\in C}(1/r_c^2(t))dt$, assuming $D(r) = (r_0^2/r^3)$. Thus, $A_c^\infty = 2A_c^{\infty *}$. □

Performance improvements in the current implementation of the Voronoi based-method are expected in the future. A constant factor is expected to be gained under implementation improvements (the current code is completely unoptimized). More important is the issue of the hierarchy in the design. The current implementation, unlike CAA, does not take any advantage of the repetition in regular designs. Note, for example, how fast CAA-DotThrow can derive a critical area estimate for the i-cache macro in Table II by exploiting the regularity of the design. Taking advantage of the design hierarchy and repetition remains a topic for future research.

VIII. CONCLUSION

We presented a geometric modeling of the critical area problem for missing material defects causing opens in VLSI circuits. This modeling generalizes the Voronoi diagram framework for critical area computation of shorts [16] with the ability to also handle opens: *breaks* and *via blocks*. We thus provide a unified approach to critical area computation for all three major failure mechanisms via Voronoi diagrams. The computation of critical area for both *breaks* and *via blocks* was reduced to variations of weighted L_∞ Voronoi diagrams of *core* segments. Furthermore, the via blocks problem introduced the min–max Voronoi diagram of rectangles, a combinatorial structure of independent interest. We presented plane sweep algorithms to compute the Voronoi diagrams needed in each case. The time complexity is $O(n \log n)$ for the Voronoi diagram for breaks and $O((n + K)\log n)$ for the min–max Voronoi diagram for blocks, where K is upper-bounded by the total number of

interacting vias; in practice, K is negligible. The total critical area integral is then derived within the same time complexity. In this paper, we considered only the rectilinear case, i.e., we assumed that shapes consist of axis-parallel edges. The method is generalizable to shapes in arbitrary orientations, but we leave the generalization for a paper with focus in computational geometry [18]. Whether the time complexity of the plane sweep algorithm to compute the min–max Voronoi diagram of rectangles can be reduced to $O(n \log n)$ remains an open problem. In future research, we expect to report results on full-chip critical area extraction based on the Voronoi diagram framework.

ACKNOWLEDGMENT

The author would like to thank Dr. G. Tellez for providing the engine on top of which the Voronoi diagram method is being developed, Prof. D. T. Lee and F. Aurenhammer for comments on a preliminary version, and Dr. A. Efrat and Dr. J. Basch for discussions regarding the size of additively weighted Voronoi diagrams.

REFERENCES

[1] A. Allen, A. Dziedzic, W. Donath, D. Newns, D. Maynard, M. Lavin, and G. Tellez, "A method in prediction of random defect yields of integrated circuits," Patent filed with the U.S. patent and trademark office, BU9-99-191.

[2] F. Aurenhammer, "Voronoi diagrams: A survey of a fundamental geometric data structure," *ACM Comput. Survey*, vol. 23, pp. 345–405, 1991.

[3] CAA: The IBM tool for critical area yield analysis, available externally as CAE: Critical Area Extraction

[4] F. Dehne and R. Klein, "'The big sweep': On the power of the wavefront approach to voronoi diagrams," *Algorithmica*, vol. 17, pp. 19–32, 1997.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*. Berlin, Germany: Springer-Verlag, 1997.

[6] A. V. Ferris-Prabhu, "Defect size variations and their effect on the critical area of VLSI devices," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 878–880, Aug. 1985.

[7] I. Koren, "The effect of scaling on the yield of VLSI circuits," in *Yield Modeling and Defect Tolerance in VLSI Circuits*, W.R. Moore, W. Maly, and A. Strojwas, Eds. Bristol, U.K.: Adam-Hilger, 1988, pp. 91–99.

[8] D. T. Lee, "Medial axis transformation of a planar shape," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 363–369, July 1982.

[9] W. Maly, "Modeling of lithography related yield losses for CAD of VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 166–177, July 1985.

[10] W. Maly, "Computer aided design for VLSI circuit manufacturability," *Proc. IEEE*, vol. 78, pp. 356–392, Feb. 1990.

[11] W. Maly and J. Deszczka, "Yield estimation model for VLSI artwork evaluation," *Electron Lett.*, vol. 19, no. 6, pp. 226–227, Mar. 1983.

[12] C. H. Ouyang, W. A. Pleskacz, and W. Maly, "Extraction of critical areas for opens in large VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 151–162, Feb. 1999.

[13] B. R. Mandava, "Critical area for yield models," IBM, East Fishkill, NY, Tech. Rep. TR22.2436, Jan. 1982.

[14] P. Nussel and G. Weinert, "System and method for verifying a hierarchical circuit design," U.S. Patent 5 528 508, Jan. 1996.

[15] E. Papadopoulou, M. Lavin, G. Tellez, and A. Allen, "An incremental method for critical area and critical region computation of via blocks," Patent filed with the U.S. patent and trademark office, YO9970-223, May 1998.

[16] E. Papadopoulou and D. T. Lee, "Critical area computation via voronoi diagrams," *IEEE Trans. Computer-Aided Design*, vol. 18, Apr. 1999.

[17] ——, "The L_∞ voronoi diagram of segments and VLSI applications," Int. J. Comput. Geometry Applicat., to be published.

[18] ——, "On the min–max Voronoi diagram of polygonal sites in the plane," unpublished.

[19] J. Pineda de Gyvez and C. Di, "IC defect sensitivity for footprint-type spot defects," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 638–658, May 1992.

[20] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York: Springer-Verlag, 1985.

[21] J. S. Rogenski, "Extraction of breaks in rectilinear layouts by plane sweeps," Univ. California, Santa Cruz, CA, Tech. Rep. UCSC-CRL-94-21, Apr. 1995.

[22] C. H. Stapper and R. J. Rosner, "Integrated circuit yield management and yield analysis: Development and implementation," *IEEE Trans. Semiconduct. Manufact.*, vol. 8, no. 2, pp. 95–101, Feb. 1995.

[23] C. H. Stapper, "Modeling of defects in integrated circuits photolithographic patterns," *IBM J. Res. Devel.*, vol. 28, no. 4, pp. 461–475, 1984.

[24] A. Wagner and I. Koren, "An interactive VLSI CAD tool for yield estimation," *IEEE Trans. Semiconduct. Manufact.*, vol. 8, no. 2, pp. 130–138, Feb. 1995.

[25] H. Walker and S. W. Director, "VLASIC: A yield simulator for integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 4, pp. 541–556, Oct. 1986.

[26] P. Widmayer, Y. F. Wu, and C. K. Wong, "On some distance problems in fixed orientations," *SIAM J. Comput.*, vol. 11, pp. 728–746, 1987.

[27] E. Papadopoulou, "L_∞ voronoi diagrams and applications to VLSI layout and manufacturing," in *Proc. 9th Int. Symp. SSAlgorithms and Computation*: LNCS, 1998, vol. 1533, pp. 9–18.

Evanthia Papadopoulou received the B.S. degree in mathematics from the University of Athens, Athens, Greece, the M.S. degree in computer science from the University of Illinois, Chicago, and the Ph.D. degree in computer science from Northwestern University, Evanston, IL, in 1995.

She spent the summer of 1993 as a Visiting Researcher at the Institute of Information Sciences, Academia Sinica, Taiwan. In 1996, she joined the IBM T. J. Watson Research Center, Yorktown Heights, NY, where she is currently a Research Staff Member in the Department of Design Automation. Her current research interests include design and analysis of algorithms, computational geometry, and VLSI computer-aided design.