Map of Geometric Minimal Cuts with Applications*

Evanthia Papadopoulou, Jinhui Xu and Lei Xu

Contents

*The work of the first author was supported in part by the Swiss National Science Foundation SNSF project 200021-127137. The work of the last two authors was supported in part by NSF through a CAREER Award CCF-0546509 and two grants IIS-0713489 and IIS-1115220.

E. Papadopoulou (✉)
Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

J. Xu • L. Xu
Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, USA

P.M. Pardalos et al. (eds.), *Handbook of Combinatorial Optimization*,
DOI 10.1007/978-1-4419-7997-1_27,

Abstract

This chapter considers the following problem of computing a map of geometric minimal cuts (called the MGMC problem): Given a graph $G = (V, E)$ and a planar embedding of a subgraph $H = (V_H, E_H)$ of G, compute the map of geometric minimal cuts induced by axis-aligned rectangles in the embedding plane. The MGMC problem is motivated by the *critical area* extraction problem in VLSI designs and finds applications in several other fields. This chapter surveys two different approaches for the MGMC problem based on a mix of geometric and graph algorithm techniques that can be regarded complementary. It is first shown that unlike the classic min-cut problem on graphs, the number of all rectilinear geometric minimal cuts is bounded by a low polynomial, $O(n^3)$. Based on this observation, the first approach enumerates all rectilinear geometric minimal cuts and computes their L_∞ Hausdorff Voronoi diagram, which is equivalent to the L_∞ Hausdorff Voronoi diagram of axis-aligned rectangles. The second approach is based on higher-order Voronoi diagrams and identifies necessary geometric minimal cuts and their Hausdorff Voronoi diagram in an iterative manner. The embedding in the latter approach includes arbitrary polygons. This chapter also presents the structural properties of the L_∞ Hausdorff Voronoi diagram of rectangles that provides the map of the MGMC problem and plane sweep algorithms for its construction.

1 Introduction

This chapter considers the following problem called *map of geometric minimal cuts (MGMC)*: Given a graph $G = (V, E)$ and a planar embedding of a subgraph $H = (V_H, E_H)$ of G, compute a map[1] $\mathcal{M}$ of the embedding plane P of H so that for every point $p \in P$, the cell in $\mathcal{M}$ containing p is associated with the "closest" *geometric cut* (in G) to p, where distance between a point p and a cut C is defined as the maximum distance between p and any individual element of C. A geometric cut C of G induced by a given geometric shape S is a set of edges and vertices in H that overlaps S in P and whose removal from G disconnects G. This chapter considers the case where geometric cuts are induced by axis-aligned rectangles and distances are measured in the L_∞ metric. The main objective of the MGMC problem is to compute the map $\mathcal{M}$ of all geometric minimal (or canonical) cuts (the exact definition of geometric minimal cuts will be given in the next section) of the planar embedding of H.

[1] A map means a partition of the embedding plane (as in a trapezoidal map) into cells so that all points in the same cell share the same "closest" geometric minimal cut.

The MGMC problem was formulated in [47]. It was introduced, as the *geometric min-cut problem*, in [34], in order to model the *critical area* computation problem for *open faults* in a VLSI design. Critical area is a measure reflecting the sensitivity of a VLSI design to random defects occurring during the manufacturing process. For a survey on VLSI yield analysis and optimization, see, for example, [16]. The critical area computation problem in a VLSI design for various types of fault mechanisms has been reduced to variants of generalized Voronoi diagrams; see, for example, [5, 32, 34–36, 46]. The Voronoi framework for critical area extraction asks for a subdivision of a layer A into regions that reveal, for every point p, the size of the smallest defect centered at p causing a circuit failure. For *open faults*, this results in the MGMC problem. In more detail, a VLSI net N can be modeled as a graph $G = (V, E)$ having a subgraph $H = (V_H, E_H)$ embedded on any conducting layer A; see, for example, [34]. The embedded subgraph $H = (V_H, E_H)$ is vulnerable to random manufacturing defects that are associated with the layer of the embedding and may create open faults. The MGMC problem computes, for every point p on layer A, the disk of minimum radius, which is centered at p and induces a cut on graph G resulting in an open fault. The subdivision of layer A that solves the MGMC problem corresponds to the *Hausdorff Voronoi diagram* of the geometric minimal cuts (see, e.g., [34, 47]). The MGMC problem finds applications in other networks, such as transportation networks, where critical area may need to be extracted for the purpose of flow control and disaster avoidance.

This chapter surveys the literature on the MGMC problem and presents two complimentary approaches. These approaches are based on a mix of geometric and graph algorithm techniques, and they produce, by different means, the L_∞ Hausdorff Voronoi diagram of geometric minimal cuts on the embedding plane. In addition, this chapter presents structural properties and algorithms for the L_∞ Hausdorff Voronoi diagram, which provides a solution to the MGMC problem, and it is a geometric structure of independent interest.

The first approach to the MGMC problem presented in this chapter is based on the results of [47] for a rectilinear embedding of the embedded subgraph H. This approach first classifies geometric cuts into two classes – 1-D cuts and 2-D cuts – and shows that the number of all possible geometric 1-D and 2-D minimal cuts, given a rectilinear embedding of H, is $O(n^2)$ and $O(n^3)$, respectively, where n is the size of H. The $O(n^3)$ bound on the number of geometric minimal cuts makes the enumeration of geometric cuts feasible. In contrast, the corresponding bound of the classic min-cut problem in graphs is exponential. Based on interesting observations and dynamic connectivity data structures, the approach in [47] directly computes all geometric minimal cuts in $O(n^3 \log n (\log \log n)^3)$ time. A special case in which the inducing rectangle of a cut has a constantly bounded edge length is also considered, in which case the time complexity of the algorithm can be significantly improved to $O(n \log n (\log \log n)^3)$. Once all geometric minimal cuts are identified, the solution to the MGMC problem is reduced to computing their Hausdorff Voronoi diagram. The method in [47] revisits the plane sweep construction of the L_∞ Hausdorff Voronoi diagram of rectangles and presents the first output-sensitive algorithm for

its construction, which runs in $O((N+K)\log^2 N \log\log N)$ time and $O(N\log^2 N)$ space, where N is the number of rectangles and K is the complexity (or size) of the Hausdorff Voronoi diagram. The algorithm is based on a plane sweep construction in 3-D and uses several interesting data structures that can achieve an output-sensitive result.

The second approach is based on results in [34], and it is based on higher-order Voronoi diagrams. The embedded subgraph needs not be rectilinear, it can be arbitrary; moreover, it corresponds to polygonal shapes. Given a geometric graph with a subgraph embedded in the plane, the method in [34] iteratively identifies geometric minimal cuts and their *generators*,[2] based on an iterative construction of order-k Voronoi diagrams in increasing order of k. In the worst case, the method requires time $O(n^3\log n)$ to compute higher-order Voronoi diagrams, plus time $O(n^3\log n(\log\log n)^3)$ to answer connectivity queries on the graph G, assuming that the dynamic connectivity data structures of [20, 45] are used for this purpose. The resulting map provides a solution to the MGMC problem and reveals the Hausdorff Voronoi diagram of all geometric minimal cuts. In practice, typically, it is not necessary to guarantee the identification of all possible geometric cuts, in which case the iterative construction of [34] can be considerably sped up. Once a sufficient set of cut generators are identified, their weighted Voronoi diagram can be computed in a simple manner, providing a map that reliably approximates the solution to the MGMC problem. In the case of rectilinear embeddings, generators are simple axis-parallel line segments of constant weights. For arbitrary embeddings, generators correspond to portions of farthest Voronoi diagrams of cuts in the final map, and their weights correspond to farthest distance functions. Assuming that the number of iterations is kept to a small constant, as experimental results in [34] suggest, this results in an $O(n\log n(\log\log n)^3)$ algorithm for an approximate map solving the MGMC problem.

The Hausdorff Voronoi diagram of a set S of point clusters in the plane is a subdivision of the plane into regions, such that the Hausdorff Voronoi region of a cluster P is the locus of points *closer* to P than to any other cluster in S, where distance between a point t and a cluster P is measured as the *farthest* distance between t and any point in P. The Hausdorff Voronoi diagram first appeared in [11] as the *cluster Voronoi diagram*, where several combinatorial bounds were derived by means of a powerful geometric transformation in three dimensions and an $O(n^2)$-time algorithm for its construction. A tight combinatorial bound on its structural complexity in the Euclidean plane, as well as a plane sweep construction were given in [33]. The L_∞ version of the problem was introduced in [32] for the VLSI *critical area extraction* problem for a defect mechanism called a *via-block*. In [32], the problem was reduced to an L_∞ Voronoi diagram of additively weighted line segments, and it was computed by a plane sweep procedure. The plane sweep construction was revisited in [47] and [39]. Additional work on Hausdorff Voronoi

[2] The generator of a cut is a portion of the farthest Voronoi diagram of the elements constituting the cut.

diagrams is included in [1,9,38]. In this chapter, we first present an output-sensitive version of the plane sweep in three dimensions for the L_∞ Hausdorff Voronoi diagram of rectangles as given in [47]. We also present the structural properties of this diagram and a simple two-dimensional plane sweep as given in [32, 39], which is not output sensitive; nevertheless, it achieves optimal $O(n \log n)$-time and $O(n)$-space in the case of non-crossing rectangles. In case of a large number of crossings, the size of the Hausdorff Voronoi diagram may be $\Omega(n^2)$ [11], and the $O(n^2)$ algorithm of [11] may result in a better approach.

This chapter is organized as follows. Section 2 refers to [47], Section 3 refers to [34], and Section 4 to [39]. The sections are presented independently and they can be read in any order. In more detail, Sect. 2.1 introduces the concepts of 1-D and 2-D geometric minimal cuts; Sect. 2.2 presents the algorithms of [47] for computing the geometric minimal cuts; and Sect. 2.3 presents the output-sensitive plane sweep algorithm in 3-dimensions of [47]. Section 3 presents an iterative approach to the MGMC problem via higher-order Voronoi diagrams based on [34]. Section 4 analyzes the structural complexity of the L_∞ Hausdorff Voronoi diagram and summarizes the simple two-dimensional plane sweep algorithm for its construction presented in [32, 39]. In Section 5 we provide concluding remarks.

2 On Geometric Minimal Cuts and Their Map

2.1 Geometric Cuts

Let $G = (V, E)$ be the undirected graph in an MGMC problem and $H = (V_H, E_H)$ be its planar subgraph embedded in the plane P with $|V| = N$, $|E| = M$, $|V_H| = n$, and $|E_H| = m$. Due to the planarity of H, $m = O(n)$. In this chapter, unless explicitly mentioned otherwise, all edges in H are either horizontal or vertical straight-line segments. A pair of vertices u and v in a graph are connected if there is a path in this graph from u to v. Otherwise, they are disconnected. A graph is connected if every pair of its distinct vertices is connected. Without loss of generality, G is assumed to be connected in this chapter. A cut C of G is a subset of edges in G whose removal disconnects G. A cut C is minimal if removing any edge from C no longer forms a cut.

Definition 1 Let R be a connected region in P, and $C = R \cap H$ be the set of edges in H intersected by R. The cut C is called a geometric cut induced by R if the removal of C from G disconnects G.

When there is no ambiguity of the region R, the cut induced by R is called a "geometric cut" for simplicity. For a given cut C, its *minimum inducing region* $R(C)$ is the minimum axis-aligned rectangle which intersects every edge of C (i.e., if we shrink the width or length of $R(C)$ by any arbitrarily small value ϵ, some edge in C will no longer be intersected by $R(C)$). For some geometric cut C, its minimum inducing region $R(C)$ could be degenerated into a horizontal or vertical

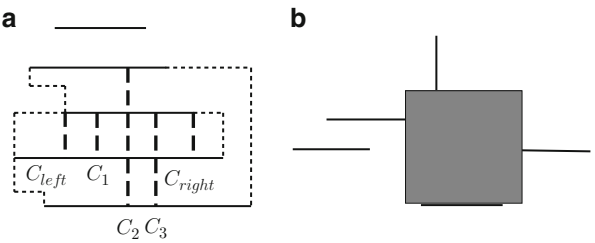

Fig. 1 (**a**) 1-D cuts C_1, $C_2, and\ C_3$ with C_3 being the minimal cut. (**b**) A 2-D cut bounded by 4 edges

line segment, or even a single point. It is easy to see that if $R(C)$ is not degenerate, it is unique.

Definition 2 A geometric cut C is called a 1-D geometric cut (or a 1-D cut) if $R(C)$ is a line segment. If $R(C)$ is an axis-aligned rectangle, then C is called a 2-D geometric cut (or a 2-D cut).

Definition 3 A geometric cut C is a geometric minimal cut if the set of edges intersected by any region obtained by shrinking $R(C)$ no longer forms a geometric cut.

The following lemma easily follows from the above definitions and the problem setting.

Lemma 1 *Let C be any geometric minimal cut. If C is a 1-D cut, then each endpoint u of $R(C)$ is incident to either an endpoint of an edge e in H with the same orientation as $R(C)$ and $e \cap R(C) = u$ or an edge in H with different orientation as $R(C)$ (see Fig. 1a). If C is a 2-D cut, each bounding edge s of $R(C)$ is incident to either an endpoint v of an edge e in H of different orientation with s and $R(C) \cap e = v$, or an edge in H with the same orientation as s (see Fig. 1b; note that s could be incident to multiple edges).*

From the above lemma, it is clear that each 1-D geometric minimal cut is bounded by up to two edges in H and each 2-D geometric minimal cut is bounded by up to 4 edges in H. For a cut C, let $B(C)$ denote the set of edges bounding C. Due to the minimality nature of C, removing any edge in $B(C)$ will lead to a non-cut. This means that any edge in $B(C)$ is necessary for forming the cut. However, this is not necessarily true for edges in $C \backslash B(C)$. Thus, a geometric minimal cut may not be a minimal cut. Note that for a given graph, the number of minimal cuts could be exponential. However, as shown later, the number of geometric minimal cuts is much less (i.e., bounded by a low degree polynomial).

For a 1-D geometric minimal cut C, it is possible that all edges in C have different orientation with $R(C)$. In this case, there may exist an infinite number of 1-D geometric minimal cuts, all cutting the same set of edges C (in other words, the minimum inducing region for such a 1-D geometric C is not unique). For example, sliding $R(C)$ along $B(C)$ will obtain an infinite number of 1-D minimal cuts.

Among these cuts, there are two extreme cuts, C_{left} and C_{right} (or C_{top} and C_{bottom} if $R(C)$ is horizontal), incident with the left and right (or up and bottom) endpoints of some edges in C, respectively. Since all these geometric cuts cut the same set of edges, they are counted as one cut.

2.2 Identifying Geometric Minimal Cuts

To solve the MGMC problem, the main idea in [47] is to first identify all possible geometric minimal cuts and then construct a Hausdorff Voronoi diagram of these cuts as the map $\mathcal{M}$. Thus, three major problems need to be solved:

1. Finding all 1-D minimal cuts;
2. Finding all 2-D minimal cuts; and
3. Efficiently constructing the Hausdorff Voronoi diagram for the geometric cuts.

This section discusses the main ideas for problems 1 and 2. The approach for problem 3 will be discussed in the next section.

2.2.1 Computing 1-D Geometric Minimal Cuts

As discussed in the previous section, a 1-D geometric minimal cut can be induced by either horizontal or vertical segments. The following lemma bounds from above the total number of 1-D geometric minimal cuts.

Lemma 2 *There are $O(n^2)$ 1-D geometric minimal cuts in H.*

Proof Only the 1-D cuts induced by vertical segments are considered. Cuts induced by horizontal segments can be proved similarly. If we place a vertical line through each vertex (or endpoint), then the plane P is partitioned into $O(n)$ vertical slabs, with each slab containing no endpoint in its interior. For a particular slab S containing k edges, say $e_1, e_2, \cdots, e_k$, it could be shown that there are at most $O(k)$ 1-D geometric minimal cuts. To see this, all 1-D minimal cuts formed by the k edges are considered. A cut C is owned by an edge e_i if $e_i \in B(C)$. Clearly, each 1-D minimal cut has at least one owner. Now consider each edge e_i, it can only be the owner of up to two 1-D minimal cuts, one bounded by e_i from the bottom and one bounded by e_i from the top. Note that due to the fact that the cuts are all minimal, it is impossible to have two cuts bounded by e_i from the top (or bottom) simultaneously. Thus, each edge in S owns at most two 1-D geometric minimal cuts. Hence, the total number of 1-D geometric minimal cuts in S is $O(k)$. Summing over all slabs, the total number of 1-D geometric minimal cuts is $O(n^2)$. Thus, the lemma follows. □

To compute the $O(n^2)$ 1-D geometric minimal cuts, since each cut is a set of edges $C \in H$ which appear consecutively in some slab and whose removal disconnects G, it is necessary to first (a) Identify all possible cuts in H, and then (b) For each possible cut, determine whether it is indeed a cut (such a test is called a *cut query*). To overcome the two difficulties, a straightforward way is to enumerate

all possible cuts, and for each possible cut C, use graph search algorithms (such as depth first search (DFS) or breadth first search (BFS)) to check the connectivity of $G \backslash C$. As shown in the proof of Lemma 2, the embedding plane P can be partitioned into $O(n)$ slabs, and for each slab with k edges, there are $O(k^2)$ subsets of consecutive edges. Thus, a total of $O(n^3)$ possible subsets need to be checked, and the connectivity checking for each subset takes $O(N + M)$ time. Therefore, this will lead to a total of $O(n^3(N + M))$ time.

To speed up the computation, the idea is to first simplify the graph G. Since the cuts involve only the edges in H, the connectivity in $G \backslash E_H$ will not be affected no matter which subset of edges in H is removed. To make use of this invariant, the connected components of $G \backslash E_H$ are firstly computed. For each connected component CC, it is contracted into a supernode v_{CC}. For each vertex $u \in H \cap CC$, an edge (u, v_{CC}) is added. Let the resulting graph be $G' = (V', E')$ (called contracted graph). The following lemma gives some properties of this graph.

Lemma 3 *The number of vertices in G' is $|V'| = O(n)$ and the number of edges in G' is $|E'| = O(n)$. Furthermore, a subset of edges in H is a cut of G if and only if it is a cut of G'.*

Proof Follows from the construction of G'. □

The above lemma shows that the size of G' could be much smaller than G. Thus, the time needed for answering a cut query is significantly reduced from $O(N + M)$ to $O(n)$.

Converting Cut Queries to Connectivity Queries As discussed previously, to compute all 1-D geometric minimal cuts, it is necessary to check $O(n^3)$ possible subsets of edges in H. Many of them are quite similar (i.e., differ only by one or a small number of edges). Thus, it would be highly inefficient to handle them independently and answer each cut query by searching the graph G'. To yield a better solution, it is better to consider all these cut queries simultaneously and share the connectivity information among similar cut queries.

To share information among slightly different cut queries, one possible way is to use persistent data structure [10]. However, due to the fact that inserting (or removing) an edge could cause a linear, instead of a constant, number of 1-D geometric minimal cuts to be destroyed or generated, persistent data structures require substantial amount of updating after each insertion or deletion, thus they do not lead to a highly efficient solution.

To overcome this difficulty, the main idea is to further decompose each cut query into a set of connectivity queries with each connectivity query involving only one edge. The reduced granularity in query enables us to better share information among related cut queries.

Connectivity Query Before continuing the discussion on connectivity queries, some existing algorithms and data structures for the dynamic connectivity problem

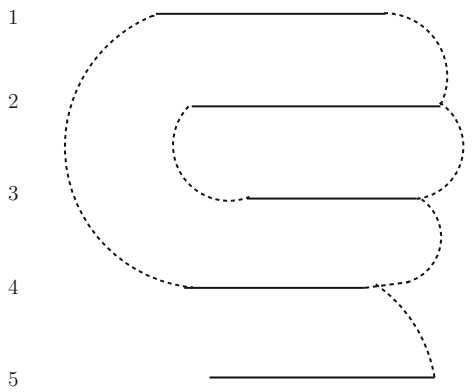

Fig. 2 An example for 1DSlab, where the *dotted lines* are the edges in $G' \setminus H$. $\{2, 3\}$ and $\{5\}$ are the 1-D geometric minimal cuts

are briefly reviewed. In the fully dynamic connectivity problem, the input is a graph G whose vertex set is fixed and whose edges can be inserted and deleted. The objective is to construct a data structure for G so that the connectivity of any pair of vertices can be efficiently determined. Most fully dynamic connectivity data structures support the following three operations: (1) *Insert*(e), (2) *Delete*(e), and (3) *Connectivity*(u, v), where the *Insert*(e) operation inserts edge e into G, the *Delete*(e) operation removes edge e from G, and the *Connectivity*(u, v) operation determines whether u and v are connected in the current graph G. Extensive research has been done on this problem, and a number of results were obtained [12, 13, 17, 18, 20, 45]. In [20], Thorup et al. gave a simple and interesting solution for this problem which answers each connectivity query in $O(\log n)$ time and takes $O(\log^2 n)$ time for each update. Later Thorup gave a near-optimal solution for this problem [45] which answers each connectivity query in $O(\log n / \log\log\log n)$ time and completes each insertion or deletion operation in $O(\log n(\log\log n)^3)$ expected amortized time.

The approach described in this chapter uses the data structure in [45] for the MGMC problem. In practice (e.g., critical area computation), the simpler algorithm in [20] may be more practical. When the choice of the connectivity data structure is unclear, $MaxQU$ is used to represent the maximum of the connectivity query time and the update operation time.

Enumerating 1-D Geometric Minimal Cuts in a Slab The problem of identifying 1-D geometric minimal cuts in a vertical slab S with k edges (see Fig. 2) is to make use of the connectivity data structure. Clearly, there are $O(k^2)$ possible 1-D geometric cuts and $O(k)$ 1-D geometric minimal cuts. To find out the $O(k)$ minimal cuts from $O(k^2)$ possible locations, edges are sorted based on the y coordinates, and let $e_1 = (u_1, v_1), e_2 = (u_2, v_2), \cdots, e_k = (u_k, v_k)$ be the k edges. A fully dynamic connectivity data structure $FDC(G')$ for the contracted graph G' is built, and then the algorithm for a slab (called 1DSlab) is run on it. The main steps of 1DSlab are the follows.

1. Initialize a variable $r = 1$ to represent the index of the next to-be-deleted edge.
2. Starting from e_r, repeatedly delete edges of S from G' according to their sorted order and store them in a queue Q. For each deleted edge e_i, query $FDC(G')$ about the connectivity of u_i and v_i. Stop the deletion until encountering the first edge e_j whose two endpoints u_j and v_j are disconnected or the last edge. In the latter case, insert all deleted edges back and stop.
3. Insert the deleted edges in Q back in the same order as they are deleted and updating $FDC(G')$. After inserting each edge e_i, query the connectivity of the two endpoints of e_j, u_j, and v_j.
4. If u_j and v_j are disconnected, add a forward pointer from e_i to e_j and insert edges in Q back to G'.
5. If u_j and v_j are connected, add a forward pointer from e_i to e_j, set $r = j + 1$, and repeat Steps 2–5 until encountering the last edge e_k. In this case, insert all remaining edges in Q back to G' and $FDC(G')$.
6. Reverse the order of the k edges and repeat the above procedure. In this step the added pointers are backward pointers.
7. For each edge e_j, find the nearest edge e_i which has a forward pointer to e_j and the nearest edge e'_i with a backward pointer to e_j. Output the set of edges between e_i and e_j (including e_i and e_j) as a 1-D geometric minimal cut and edges between e_j and e'_i (including e_j and e'_i) as another 1-D geometric minimal cut.

   The correctness of the above algorithm is shown below.

Lemma 4 *Assume G' is originally a connected graph. In Step 2 of the 1DSlab algorithm, if u_i and v_i (the endpoints of the last deleted edge e_i) are connected, the set of deleted edges (i.e., e_r to e_i) does not form a cut in G'. On the other hand, if u_i and v_i are disconnected, then the set of deleted edges does form a cut in G.*

Proof Firstly, if u_i and v_i are disconnected, obviously the set of deleted edges forms a cut. Thus, only the case that u_i and v_i are connected is considered. In this case, the set of deleted edges does not form a cut.

This could be proved by induction on the number i of deleted edges. The base case is $i = 1$. In this case, since G' is originally a connected graph and u_i and v_i are still connected in $G' \backslash \{e_i\}$, obviously there exists no cut. Assume that after deleting $i - 1$ edges, the set of edges $S_{i-1} = \{e_1, e_2, \cdots, e_{i-1}\}$ does not form a cut. Then since $G' \backslash S_{i-1}$ is a connected graph, after deleting edge e_i, the case becomes the same as the base case. Thus, the lemma follows. □

Lemma 5 *In the 1DSlab algorithm, the disconnectivity of u_j and v_j in Step 4 or the connectivity of u_j and v_j in Step 5 implies that the set $S_{i,j} = \{e_i, e_{i+1}, \cdots, e_j\}$ forms a cut in G'.*

Proof The first case (i.e., u_j and v_j are disconnected in Step 4) is obvious. For the second case (i.e., u_j and v_j are connected in Step 5), since u_j and v_j are disconnected before the insertion of e_i, it follows that $S_{i,j}$ is a cut. □

The above lemmas indicate that the cut query can be answered by a sequence of connectivity queries.

Theorem 1 *The 1DSlab algorithm generates all 1-D geometric minimal cuts in the slab S in $O(kMaxQU)$ time, where $MaxQU$ is the maximum of the query time and the updating time of the $FDC(G')$ data structure.*

Proof The reason that the 1DSlab algorithm generates all 1-D geometric minimal cuts is shown below. By Lemmas 4 and 5, it is known that the forward pointers to e_j indicate all cuts whose edge set appears consecutively and ends at edge e_j. Similarly, the backward pointers to e_j indicate all cuts whose edge set appears consecutively and starts from e_j. By finding the nearest edges with forward and/or backward pointers to e_j, the algorithm identifies the (up to) two 1-D geometric minimal cuts starting from and ending at e_j. Since the computation is for every edge in S, it generates all 1-D geometric minimal cuts.

For the running time, it is clear that the sorting takes $O(k \log k)$ time. After sorting, in each of the forward and backward computations, every edge in S is deleted and inserted once. As for the connectivity queries, the endpoints of each edge can be queried multiple times (one in Step 2 and others in Step 3). To bound the total query time, the query on u_j and v_j in Step 3 is charged to edge e_i. Since e_i is inserted only once, it will be charged only once. Thus, each edge in S will be responsible for at most two queries. Therefore, the total number of connectivity queries is $O(k)$. Since the insertion, deletion, and query operations take no more than $O(MaxQU)$ time, the theorem follows. □

From the above theorem, it is known that even though there are $O(k^2)$ consecutively appeared subsets of edges that could be 1-D geometric minimal cuts, the connectivity data structure can reduce the time to near linear.

Generating 1-D Geometric Minimal Cuts The algorithm 1DSlab is combined with a plane sweep algorithm to generate all possible 1-D geometric minimal cuts in H.

A straightforward way of using 1DSlab is to partition the plane P into $O(n)$ slabs and apply the 1DSlab algorithm in each slab. This will lead to a $O(n^2 MaxQU)$-time solution. A more output-sensitive solution is to use the following plane sweep algorithm.

In the plane sweep algorithm, a vertical sweeping line L is used to sweep through all edges in H to generate those 1-D geometric minimal cuts induced by vertical segments. Similarly, those 1-D geometric minimal cuts induced by horizontal segments can be generated by sweeping a horizontal line. For the vertical sweeping line algorithm, the event points are the set V_H of endpoints in H. At each event point, the set of edges intersecting the sweeping line L forms a slab. However, instead of applying the 1DSlab algorithm to the whole set of intersecting edges, only a subset of edges are considered.

Let u be the event point. If u is the left endpoint of an edge e, then e is the new edge just encountered by L. Thus, it is only needed to identify all 1-D geometric minimal cuts which contain e. This means that the 1DSlab algorithm needs only to consider the minimal cut C_1 bounded by e from the bottom, the minimal cut C_2 bounded by e from the top, and all minimal cuts between the top edge e_t of C_1 and the bottom edge e_b of C_2. To deal with this case, the 1DSlab algorithm can be easily modified. Particularly, the modified algorithm can start from e, have a backward computation, and stop at the first edge e_t, whose removal disconnects its two endpoints, to generate C_1. Similarly, the algorithm can start from e and have a forward computation to generate C_2. In this way, dealing with possibly many edges beyond e_t and e_b is avoided.

If u is the right endpoint of e, it is necessary to check those cuts containing e to see whether they are still geometric minimal cuts. It is also needed to check whether new cuts can be generated. Clearly, it is only needed to consider the edges between e'_t and e'_b, where e'_t is the top edge of the 1-D geometric minimal cut bounded by e from the bottom and e'_b is the bottom edge of the 1-D geometric minimal cut bounded by e from the top. Thus, such information from the previous step (i.e., the step before encountering u) are first obtained and then the 1DSlab algorithm on the subset of edges between e'_t and e'_b is applied directly.

Theorem 2 *All 1-D geometric minimal cuts of H can be found in $O(n \times MaxC \times MaxQU)$ time, where $MaxC$ is the maximum size of a 1-D geometric minimal cut.*

Proof Follows from the above discussion. □

2.2.2 Computing 2-D Geometric Minimal Cuts

This section extends the algorithm in Sect. 2.2.1 to compute 2-D geometric minimal cuts.

To compute 2-D geometric minimal cuts, it is easy to see that a 1-D geometric minimal cut is a degenerate version of a 2-D geometric minimal cut. The only difference is that the minimum inducing region of a 1-D cut has two opposite sides degenerated to points. Thus, if two opposite sides of the minimum inducing region $R(C)$ of a 2-D cut C are conceptually "contracted" into "points," the plane sweep algorithm for 1-D cuts can be applied to generate 2-D cuts.

To implement this idea, two parallel sweeping lines L_1 and L_2 (called primary and secondary sweeping lines) are used to bound the "contracted" sides of $R(C)$. By Lemma 1, it is known that each 2-D geometric minimal cut is bounded by the endpoints (or edges) of up to four edges. This suggests that the possible locations of L_1 and L_2 are the endpoints of the input edges. Similarly to the plane sweep algorithm for 1-D cuts, the edges in H are swept twice, one time vertically and the other time horizontally. The discussion below is focused on horizontal sweeping (i.e., using vertical sweeping lines).

The double plane sweep algorithm first sorts all edges in H based on the x coordinates of their left endpoints (for vertical segments, their upper endpoints are viewed as the left endpoints and their lower endpoints as the right endpoints).

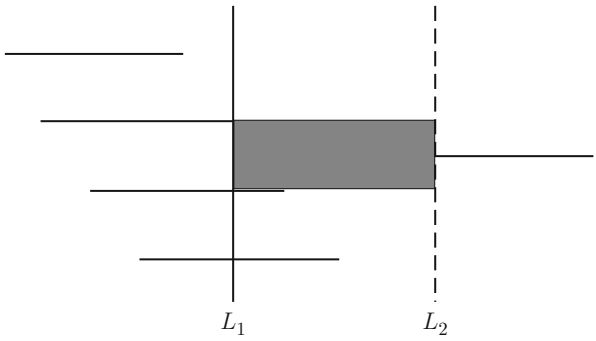

Fig. 3 An example for illustrating the double plane sweep algorithm for 2-D cuts

Let $S_1 = \{e_1, e_2, \cdots, e_n\}$ be the sorted order. The primary sweeping line L_1 stops at the left endpoint of each edge e in the order of S_1. If the right endpoint of e is to the left of the left endpoint of the next edge $e' \in S_1$, the sweep line L_1 is moved to the right endpoint of e (see Fig. 3). If e is a cut by itself, L_1 is moved to the next edge in S_1. If e is not a cut, L_1 is fixed and the secondary sweeping line L_2 is swept from the left endpoint of the next edge e' in S_1 to the last edge in S_1. When L_2 stops, the plane sweep algorithm for 1-D cuts (in Sect. 2.2) is used to compute all 2-D geometric minimal cuts formed by the set S of edges intersecting the vertical region VR bounded by L_1 and L_2.

For edges in S, there are two sorted orders S_L and S_U. S_L is sorted based on the y coordinates of the lower endpoints, and S_U is based on the y coordinates of the upper endpoints. (For horizontal edges, their left endpoints are viewed as upper endpoints and right endpoints as the lower endpoints. It is not difficult to see that the two sorted lists can be dynamically maintained in the double plane sweep algorithm.) S_L is used to generate cuts for edges above e and S_U is used to generate cuts for edges below e.

Each 2-D geometric minimal cut C in the vertical region VR is bounded by L_1 from left, L_2 from right, an edge e_u from above, and an edge e_d from below. Since all edges in S are in sorted order and $R(C)$ intersects edges in the same order in either S_L or S_U (depending on their relative locations to e), the 2-D geometric minimal cuts in VR can be viewed as 1-D geometric minimal cuts induced by vertical "segments" with a segment width equal to the horizontal distance of L_1 and L_2. Thus, they can be generated by using the plane sweep algorithm for 1-D cuts. Furthermore, similar to the plane sweep algorithm for 1-D cuts, in VR it is only needed to consider those 2-D geometric minimal cuts containing edge e (see Fig. 4).

Once all the cuts in VR are identified, L_2 is moved to the next edge in S_1. After sweeping L_2, L_1 is moved to the next edge in S_1 and the above procedure is repeated until L_1 sweeps every edge.

The following lemma is used to analyze the double plane sweep algorithm.

Lemma 6 *There are at most $O(n^3)$ 2-D geometric minimal cuts in H.*

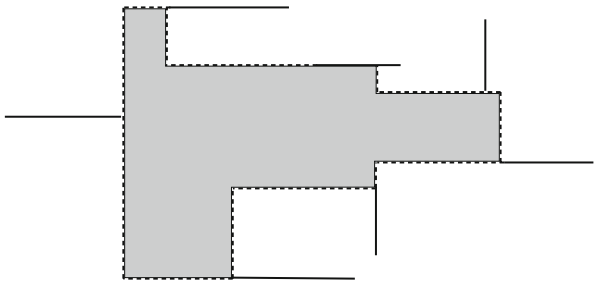

Fig. 4 Example of the double plane sweep algorithm for 2-D cuts. The region bounded by the *dotted lines* is the actual region where the algorithm searches for all 2-D geometric minimal cuts bounded by the leftmost segment from the *left*.

Proof By Lemma 1, it is known that the inducing region of each 2-D geometric minimal cut is bounded by an edge on each side. There are $O(n^2)$ pairs of edges to bound the left and right sides of the inducing region. For each pair, the vertical region is similar to a slab (according to the above discussion). By a similar argument in the proof of Lemma 2, it is easy to see that in each vertical region, there at most $O(n)$ 2-D geometric minimal cuts. Thus, the lemma follows. □

For the running time of the double plane sweep algorithm, the primary sweeping line L_1 stops at $O(n)$ location. For each fixed location of L_1, L_2 sweeps all edges not yet encountered by L_1, whose number can be $O(n)$. For each vertical region bounded by L_1 and L_2, it takes $O(MaxC \times MaxQU)$ time (by Theorem 2). Thus, the total time is $O(n^2 \times MaxC \times MaxQU)$.

Theorem 3 *All 2-D geometric minimal cuts in H can be found in $O(n^2 \times MaxC \times MaxQU)$ time.*

Proof Follows from the above discussion. □

Corollary 1 *All geometric minimal cuts in the MGMC problem can be found in $O(n^3 \log n (\log \log n)^3)$ time in the worst case.*

Proof Follows from Theorems 2 and 3, and article [45]. □

Corollary 2 *If the maximum size of a cut is bounded by a constant, then all geometric minimal cuts in H can be found in $O(n \log n (\log \log n)^3)$-time.*

Proof If the maximum size of a defect is bounded by some constant, the size of an inducing rectangle is bounded by a constant. Furthermore, since edges in VLSI design are separated by some constant distance, thus the total number of edges in an inducing rectangle is also a constant. In this case, the secondary sweeping line L_2 needs only to sweep a constant number of edges. Thus, the running time of both plane sweep algorithms is $O(n \times MaxQU)$. Also from Theorems 2 and 3, article [45], the lemma follows. □

2.3 Generating Map of Geometric Minimal Cuts

This section shows that Problem 3 can be solved by using the Hausdorff Voronoi diagram.

2.3.1 From Geometric Minimal Cuts to Hausdorff Voronoi Diagram

Given two sets A and B, the directed Hausdorff distance from A to B is $h(A, B) = \{\max_{a \in A} \min_{b \in B} d(a, b)\}$, and the undirected Hausdorff distance between A and B is $d_h(A, B) = \max\{h(A, B), h(B, A)\}$, where $d(a, b)$ is the distance between the points a and b. In this chapter, the L_∞ metric is used to measure the distance.

Given a set $\mathcal{C}$ of geometric minimal cuts of H, the Hausdorff Voronoi diagram of $\mathcal{C}$ is a partition of the embedding plane P of H into regions (or cells) so that the Hausdorff Voronoi cell of a cut $C \in \mathcal{C}$ is the union of all points whose Hausdorff distance to C is closer than to any other cut in $\mathcal{C}$. This means that for any point $p \in P$, if an L_∞ ball is grown from p (i.e., an axis-aligned square centered at p), the ball entirely contains $R(C(p))$ earlier than the minimum inducing region of any other cut, where $C(p)$ is the cut owning the Hausdorff Voronoi cell containing p. Thus, if such a map $\mathcal{M}$ exists for the critical area extraction problem, then for each point p, one can easily determine the minimum size of a defect centered at p and disconnecting the circuit.

In the MGMC problem, two types of objects exist, the minimum inducing regions of 1-D geometric minimal cuts and the minimum inducing regions of 2-D geometric minimal cuts. For every 2-D cut C, the rectangle $R(C)$ is fixed. However, for a 1-D cut C, the location of $R(C)$ is not fixed, since there may be an infinite number of 1-D cuts cutting the same set of edges. In this case, the union of the infinite number of inducing segments $R(C)$ is used to represent the cut, which is a rectangle (denoted by $UR(C)$) bounded by the two extreme 1-D cuts C_{left} and C_{right} (or C_{top} and C_{bottom}) and the two bounding edges $B(C)$ of C. Thus, from thereafter, the objects of the Hausdorff Voronoi diagram are a set of axis-aligned rectangles.

As it is well known, the Hausdorff Voronoi diagram construction can be viewed as propagating a wave from each rectangle with unit speed. The Hausdorff distance from an arbitrary point p to an axis-aligned rectangle $R(C)$, corresponding to the minimum inducing region of a 2-D cut C, is considered to better illustrate the wave propagation. The Hausdorff distance of $d_h(p, R(C))$ is achieved at one of the four corner points, $v_1(C), v_2(C), v_3(C)$, and $v_4(C)$, of $R(C)$. Thus, $d_h(p, R(C)) = \max\{d_\infty(p, v_1(C)), d_\infty(p, v_2(C)), d_\infty(p, v_3(C)), d_\infty(p, v_4(C)))\}$. To propagate a wave $W(C)$ from $R(C)$, the initial wavefront $\partial W(C)$ is the set of points whose Hausdorff distance to $R(C)$ is the minimum. Notice that when $R(C)$ has positive size (i.e., $R(C)$ is not a point), the minimum Hausdorff distance is positive (i.e., $\max\{a, b\}/2$, where a, b are the length and width of $R(C)$ respectively) and it is achieved when $d_\infty(p, v_1(C))$, $d_\infty(p, v_2(C))$, $d_\infty(p, v_3(C))$ and $d_\infty(p, v_4(C)))$ are equal. The wavefront then expands to points having larger Hausdorff distance to $R(C)$. Since the Hausdorff distance to $R(C)$ is determined by the four corner points, an equivalent view is to propagate four distinct waves from the four corner points of $R(C)$ with each being an L_∞ ball. Let $B_1(C)$, $B_2(C)$, $B_3(C)$, and $B_4(C)$ be the

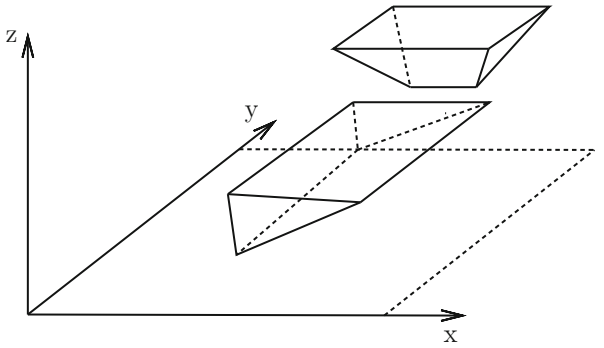

Fig. 5 Wavefronts of geometric minimal cuts

four L_∞ balls. The common intersections of the four balls constitute the wave of $R(C)$ (i.e., $W(C) = \cap_{i=1}^{4} B_i(C)$). Initially, $\partial W(C)$ is empty. Once the size of the four balls $B_i(C)$ reaches the minimum Hausdorff distance to $R(C)$, their common intersection forms a segment s_C located at the center of $R(C)$ and parallel to the shorter side of $R(C)$. As $B_i(C)$ grows, $\partial W(C)$ becomes a rectangle.

To visualize the whole growing process, the waves can be lifted to 3-D with time being the third dimension. Thus, the wavefront of $R(C)$ becomes a 4-sided facet cone in the 3-D space and apexed at s_C (i.e., the apex is not in the xy plane due to its positive minimum Hausdorff distance; see Fig. 5). Each facet of $\partial W(C)$ forms a 45° angle with the xy plane.

For a 1-D cut C, its wavefront is slightly different. Let $UR(C)$ be the rectangle of C. Since $UR(C)$ is the union of an infinite number of inducing segments $R(C)$, the Hausdorff distance to an arbitrary point p is calculated differently. For a fixed inducing segment $R(C) \in UR(C)$, let $u_1(R(C))$ and $u_2(R(C))$ be its two endpoints. The Hausdorff distance from $R(C)$ to p is achieved at one of the two endpoints (i.e., $d_h(p, R(C)) = \max\{d(p, u_1(R(C))), d(p, u_2(R(C)))\}$). Thus, the wavefront of $R(C)$ is the common intersection of two L_∞ balls centered at the two endpoints, respectively, which is also a facet cone in 3-D space.

The Hausdorff distance from p to $UR(C)$ is the minimum distance from p to one of the inducing segments in $UR(C)$, that is,

$$d_h(p, UR(C)) = \min_{R(C) \in UR(C)} d_h(p, R(C)).$$

Thus, the wavefront of $UR(C)$ is the union of an infinite number of wavefronts, which is still a facet cone with similar shape to the wavefront of a 2-D cut. The difference between the wavefront of $UR(C)$ and that of a 2-D cut with exactly same-shaped $R(C)$ is that their respective s_C may orient differently and locate at different heights.

Lemma 7 *Let C be a 1-D or 2-D geometric minimal cut. At any moment, the wavefront of C is either empty or an axis-aligned rectangle. Furthermore, the*

wavefront in 3-D is a facet cone apexed at a segment and with each facet forming a 45° *angle with the* xy *plane.*

Proof Follows from the above discussion. □

With the 3-D wavefronts of all cuts, the Hausdorff Voronoi diagram can be constructed by computing the vertical projection of the lower envelope of the set of 3-D wavefronts.

Lemma 8 *The Hausdorff Voronoi diagram can be obtained by projecting the lower envelope of the 3-D facet cones to the* xy *plane.*

Proof Follows from the definitions of the 3-D wavefronts and the Hausdorff Voronoi diagram. □

2.3.2 Plane Sweep Approach and Properties of 3-D Cones and Hausdorff Voronoi Diagram

To efficiently construct the Hausdorff Voronoi diagram $HVD(\mathcal{C})$, the approach follows the spirit of Fortune's plane sweep algorithm for points [15] and sweeps along the x axis direction a tilted plane Q in 3-D which is parallel to the y axis and forms a 45° with the xy plane (see Fig. 6a). Q intersects the xy plane at a sweep line L parallel to the y axis.

Since every facet of a 3-D facet cone forms a 45° angle with the xy plane and apexed at either a horizontal or vertical segment, the intersection of Q and a cone $\partial W(C)$ is either a V-shape curve (i.e., consisting of a 45° ray and a 135° ray on Q) or a U-shape curve (i.e., consisting of a 45° ray, a segment parallel to L, and a 135° ray; see Fig. 6b). When the cone is first encountered, it introduces either a V-shape curve or a U-shape curve to Q. When L (or Q) moves, the curve grows and its shape may change from a V-shape to a U-shape. Accordingly, the lower envelope (or beach line) of all those V- or U-shape curves forms a monotone polygonal curve, instead of parabolic arcs as in Fortune's algorithm.

With the beach line, one might think of directly applying Fortune's algorithm to sweep the objects. However, due to the special properties of this problem, quite a few significant differences exist between the MGMC problem and the ordinary Voronoi diagram problem, which fail the Fortune's algorithm. Below is a list of major differences.

1. When a cone is first encountered by Q, its corresponding initial V- or U-shape curve may not necessarily be part of the beach line.
2. The initial V- or U-shape curve may affect a number of curves in the beach line.
3. Once a V shape moves away from the beach line, it may re-appear in the beach line in a later time.

The differences indicate that it is not sufficient to maintain only the beach line in order to extract all possible event points and produce the Voronoi diagram. More information of the arrangement of the V- and U-shape curves is needed. This seemingly suggests that an algorithm with running time of the order of the

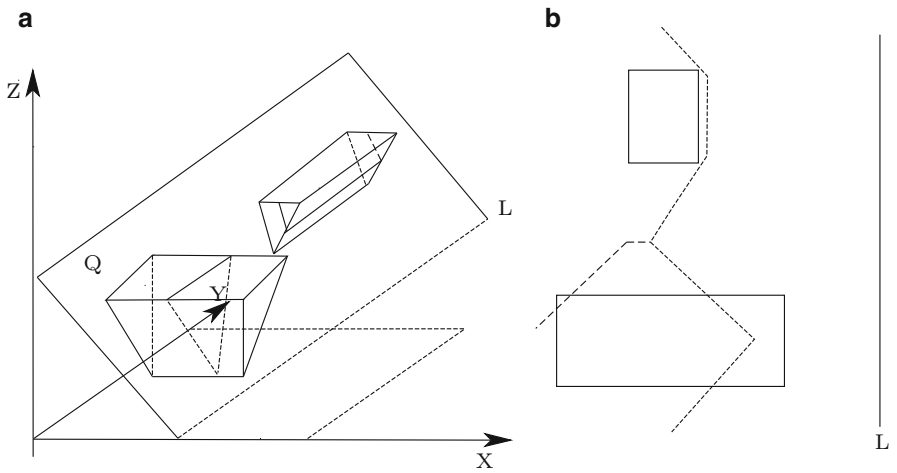

Fig. 6 (**a**) The intersections of 3-D plane Q and cones. (**b**) The Voronoi diagram with sweep line and beach line

complexity of the arrangement might be unavoidable. With several interesting observations and ideas, an output-sensitive plane sweep algorithm could be achieved. First, some basic properties of the Hausdorff Voronoi diagram of $\mathcal{C}$ are presented below.

Definition 4 A 3-D facet cone $\partial W(C)$ is a U cone (or V cone) if its apex segment s_C is parallel to the y (or x) axis.

For U cones, let $\partial W(C)$ be any U cone with apex segment s_C, and v_1 and v_2 be the two endpoints of s_C. When the sweep plane Q first encounters $\partial W(C)$, it introduces a U-shape curve C_u to Q. Let r_l, r_r, and s_m be the left and right rays and the middle segment of C_u, respectively. Initially, s_m is the apex segment s_C, and r_l and r_r are the two edges of facet cone. When Q (or L) moves, C_u grows and always maintains its U-shape.

Lemma 9 *Let $\partial W(C)$, C_u, r_l, r_r, and s_m be defined as above. When Q moves in the direction of the x axis, C_u is always a U-shape curve. The two supporting lines of r_l and r_r remain the same on Q, and the two endpoints of s_m (also the fixed points of r_l and r_r) move upwards in unit speed along the two supporting lines.*

Proof Follows from the shape and orientation of a U cone. □

For an arbitrary V cone $\partial W(C')$, let $s_{C'}$ be its apex segment and v'_1 and v'_2 be its two endpoints (or left and right endpoints). When Q first touches $\partial W(C')$ at v'_1,

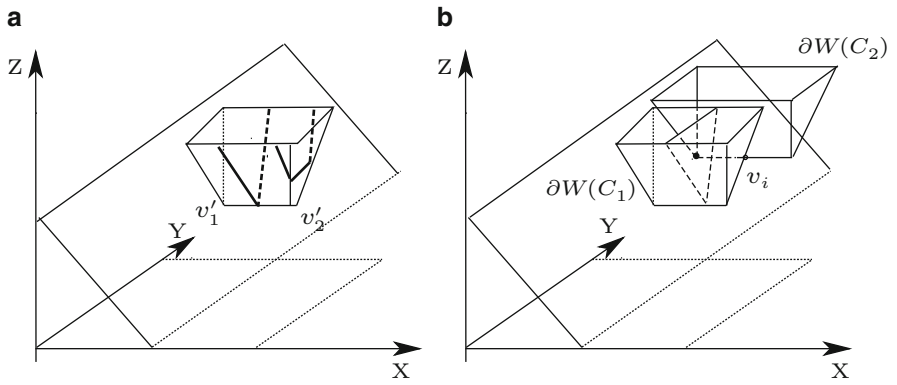

Fig. 7 (**a**) The V-shape curve changes to U-shape curve. (**b**) The V cone is hidden by another V cone

it generates a V-shape curve C'_v. C'_v remains a V-shape curve before encountering v'_2. After that, C'_v becomes a U-shape curve (see Fig. 7a).

Lemma 10 *Let r_l and r_r be the two rays of C'_v, and s_m be the middle segment of the U-shape curve C'_v after Q visits v'_2. During the whole sweeping process, the supporting lines of r_l and r_r are fixed lines on Q. C'_v remains the same V-shape curve on Q before encountering v'_2. s_m moves upwards in unit speed along the supporting lines of r_l and r_r after Q encounters v'_2.*

Proof Follows from the shape and orientation of a V cone. □

As mentioned earlier, the apex segment of each 3-D cone is located at different height (i.e., its minimum Hausdorff distance). The heights and shapes of the 3-D cones are the main reasons which cause the three differences in the MGMC problem. For example, due to the existence of height in a 3-D cone, the initial curve created by a newly encountered cone may be above the beach line (i.e., Difference (1)). In addition due to the different size of the initial curve (unlike the ordinary Voronoi diagram in which the initial curve is a vertical ray), it may intersect a number of segments or rays of the beach line (i.e., Difference (2)). More importantly, due to the existence of U- and V-shape curves, a V-shape curve which is not part of the beach line could become part of the beach line in a later time (i.e., Difference (3)). It is shown in the following lemma.

Lemma 11 *Let $\partial W(C_1)$ be either a U or V cone and $\partial W(C_2)$ be a V cone with its left endpoint v_1 of s_{C_2} being inside of $\partial W(C_1)$ and its right endpoint v_2 being outside of $\partial W(C_1)$. If $\partial W(C_2)$ is not entirely contained by the union $\cup_{C_i \in \mathcal{C}; C_i \neq C_2} \partial W(C_i)$, the V-shape curve C_2 introduced by $\partial W(C_2)$ will be hidden by the beach line at the*

beginning and will become part of the beach line later. This is the only case in which a hidden U- or V-shape curve can appear in the beach line.

Proof To simplify the proof, $\partial W(C_1)$ and $\partial W(C_2)$ are assumed to be the only two cones in $\mathcal{C}$ (see Fig. 7b). The multiple cones case can be proved similarly by induction.

By the definition of the two 3-D cones, it is known that Q first encounters $\partial W(C_1)$ and generates a curve C_1 on Q. C_1 is the only curve in the beach line. When v_1 of $\partial W(C_2)$ is first encountered by Q, it introduces a V-shape curve C_2 on Q. Since v_1 is inside $\partial W(C_1)$ and every facet of the two cones forms a 45° angle with the xy plane, the two rays of C_2 will be inside the region defined by the two rays of C_1. This means C_2 will not be part of the beach line (or lower envelope). Since $\partial W(C_2)$ is not entirely inside $\partial W(C_1)$, v_2 must be outside of $\partial W(C_1)$. Let v_i be the intersection point of s_{C_2} and the wall of $\partial W(C_1)$. When Q sweeps through v_i, the apex point of the V-shape curve C_2 will intersect the U-shape curve C_1 (note that at this moment C_1 can only be a U-shape curve even if $\partial W(C_1)$ is a V cone), since C_1 is the intersection of Q and $\partial W(C_1)$ and v_i is the intersection of $\partial W(C_1)$ and s_{C_2}. Thus, C_2 becomes part of the beach line.

To show that this is the only case where a hidden curve could appear in the beach line, first it is noted that the apex segment s_{C_2} of $\partial W(C_2)$ cannot be completely outside of $\partial W(C_1)$, otherwise the initial curve of C_2 will be part of the beach line. Second, $\partial W(C_1)$ cannot be a U cone. If this is the case, then s_{C_2} is either partially or entirely inside $\partial W(C_1)$. For the first case, the middle segment s_m of the initial U-shape curve C_2 will intersect one of the two rays of C_1, which makes C_2 be part of the beach line. For the second case, the initial U-shape curve C_2 will be hidden by C_1. When Q moves, only the middle segment s_m of C_2 moves upwards in unit speed along the two rays of C_2 (by Lemma 9). If $\partial W(C_1)$ is a U cone, then by Lemma 9, the middle segment of C_1 will also move upwards in unit speed. Thus, it will never catch up s_m. Therefore C_2 will never be part of the beach line. Similarly the same for the case $\partial W(C_1)$ is a V cone can be proved. Thus, $\partial W(C_2)$ has to be a V cone. In this case, if s_{C_2} is completely inside of $\partial W(C_1)$, then by the same argument, it can be shown that C_2 will never be part of the beach line. Thus, the lemma follows. □

In the above lemma, the point v_i indicates that when Q sweeps it, the beach line is having a topological structure change. Thus, v_i needs to be an event point for the plane sweep algorithm. However, since v_i is the intersection point of an apex segment and a 3-D cone, it has to be computed on the fly. This indicates that in the MGMC problem there is a new type of event points.

The ideas for constructing the Hausdorff Voronoi diagram $HVD(\mathcal{C})$ are discussed below. First, the bisector of two rectangles (or cuts) is considered. Let C_1 and C_2 be two axis-aligned rectangles in $\mathcal{C}$. The bisector of C_1 and C_2 is a segment with two rays as shown in Fig. 8. Each bisector contributes two vertices to the Hausdorff Voronoi diagram. Hence, the Hausdorff Voronoi diagram consists of two types of vertices: (a) The intersection points of the bisectors; and (b) The vertices of the bisectors.

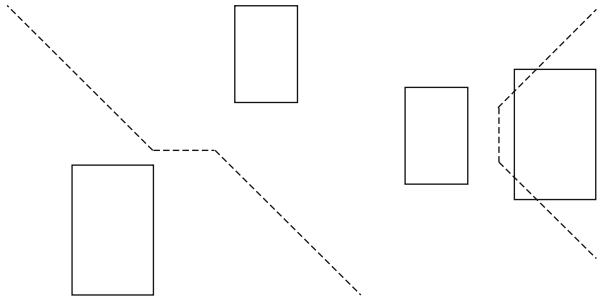

Fig. 8 Bisector is composed by a segment with two half-lines

Lemma 12 *Let $\mathcal{C}$ be a set of N rectangles. The edges of $HVD(\mathcal{C})$ are either segments or rays, and the vertices of the $HVD(\mathcal{C})$ are either the vertices of bisectors or the intersections of bisectors.*

Proof Follows from the above discussion. □

To obtain a plane sweep algorithm, it is needed to design data structures to maintain the beach line and the event points. In the MGMC problem, the beach line is the lower envelope of the set of V- and U-shape curves and is a y-monotone polygonal curve. For non-disjoint 3-D cones, the complexity of the beach line may not be linear in the number of the rectangles in $\mathcal{C}$. Figure 9b shows a newly generated U-shape curve intersecting the beach line a number of times and contributing multiple edges to it. Consequently, the complexity of $HVD(\mathcal{C})$ is not linear. The following lemma is a straightforward adaptation of Theorem 1 in [33] for the L_∞ metric.

Lemma 13 *The size of the L_∞ Hausdorff Voronoi diagram of N rectangles is $O(N + M')$, where M' is the number of intersecting pairs of rectangles. In the worst case, the bound is tight.*

2.3.3 Events

For event points, it is needed to detect all events that cause the beach line to have topological structure changes. More specifically, it is necessary to identify all the moments when a U- or V-shape curve is inserted or deleted from the beach line. There are two ways by which a curve could appear in the beach line:

(A) A newly generated U- or V-shape curve becomes part of the beach line.
(B) A hidden V-shape curve appears in the beach line.

There are also two ways for a curve or a portion of a curve to disappear from the beach line:

(C) A curve (or part of the curve) becomes hidden by a newly generated curve.
(D) A U-shape curve (or part of the U-shape curve) moves out of the beach line.

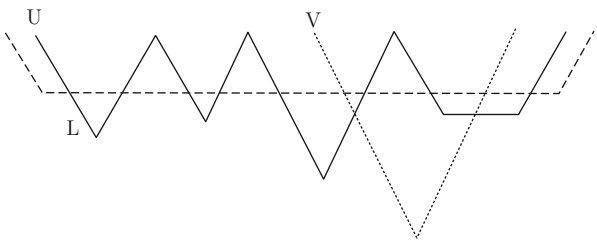

Fig. 9 The beach line L is intersected by a new U cone (*dashed line*) or V cone (*dotted line*)

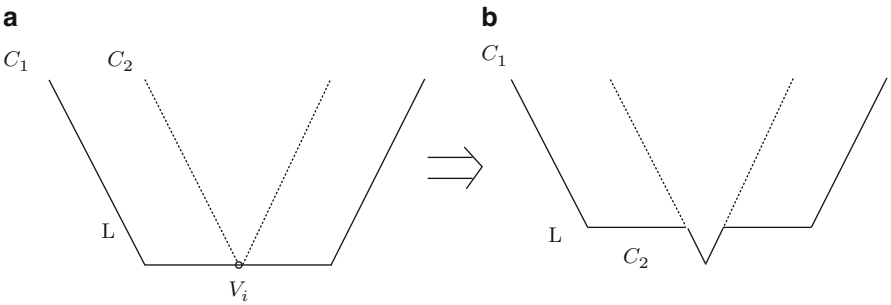

Fig. 10 (**a**) The *bottom* segment of U-shape curve C_1 reached V_i. (**b**) The hidden V-shape curve C_2 appeared on the beach line L

For (A), it is known that this is caused by the sweep plane Q encountering a new 3-D cone $\partial W(C)$. Such events can be detected in advance and are called *site events*. A site event, however, does not necessarily lead to a topological structure change to the beach line, since the new curve C can be hidden by the beach line. If $\partial W(C)$ is a U cone, the two endpoints of s_C are encountered by Q at the same time and either of them can be treated as a site event. If the corresponding U-shape curve C is not hidden, it may intersect the beach line multiple times as shown in Fig. 9. In this case, it is necessary to update the beach line for each breakpoint introduced by C. Consequently, the U-shape curve C will be partitioned into multiple pieces. Each piece is either a part of the beach line (called unhidden portion of C) or hidden by other U- or V-shape curves in the beach line (called a hidden portion of C). If $\partial W(C)$ is a V cone, then the left endpoint v_1 of s_C is encountered first and can be viewed as a site event. When Q sweeps the right endpoint v_2 of s_C, the C changes from a V-shape curve to a U-shape curve. To distinguish from the site event, v_2 is called as a U event. For unhidden V-shape curve C, it intersects the beach line at most twice (see Fig. 9).

For (B), it occurs when an unhidden portion of the bottom segment s_m of a U-shape curve C_1 moves upwards and encounters the apex point of a hidden V-shape curve C_2. This kind of events is called V events (see Figs. 10 and 7b). The V events are not known in advance and need to be computed by using the saved information in the data structures. Note that when s_m moves upwards, its hidden portions may also encounter the apex point of some hidden V-shape curve. In this case, it is not viewed as an event since the beach line has no topological structure

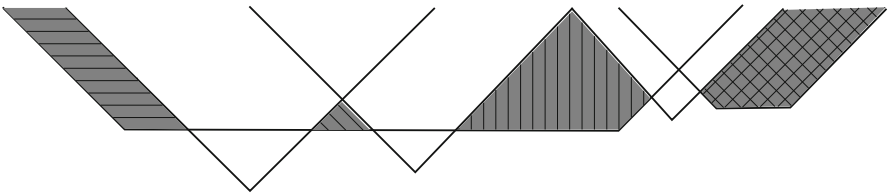

Fig. 11 Dominating regions of unhidden portions of U-shape curves

change, thus generating no new Voronoi vertex or edge. To distinguish the two cases, each unhidden portion of a U-shape curve is associated with a region, called *dominating region* (see Fig. 11), which is the region swept by the unhidden portion when it moves upwards. The dominating regions could have a few different shapes (see the shaded regions in Fig. 11), with each of them bounded by zero, one, or two 45° rays, zero, one, or two 135° rays, and an unhidden portion of the bottom segment s_m of some U-shape curve. Clearly, for a hidden V-shape curve to cause a V event, its apex point has to fall in a dominating region of an unhidden portion of a U-shape curve. Thus, to capture all V events, it is only needed to focus on those hidden V-shape curves whose apex points fall in the dominating regions.

For (C), it is obviously caused by a site event and thus can be detected in advance. For (D), the disappearing U-shape curve C (or its unhidden portion) is caused by the upward movement of the bottom segment of C. It involves three curves, C and its immediate left and right neighbors C' and C'' in the beach line. Since this event is similar to the circle event in the computation of the standard Voronoi diagram, it is also called here a circle event. The circle events cannot be detected in advance and have to be computed on the fly.

Thus, there are in total four types of events, site events, circle events, U events, and V events. The ideas on how to handle these events are discussed in the next section.

2.3.4 Data Structures and Events Handling

To construct $HVD(\mathcal{C})$, doubly connected edge lists are used to store $HVD(\mathcal{C})$. Two data structures for the plane sweep algorithm are also needed: an event queue and a sweep plane status structure representing the beach line.

The status structure for the beach line consists of three balanced binary search trees $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$. $\mathcal{T}$ stores the y-monotone polygonal curve of the beach line. Each part of the curve corresponds to a V-shape curve or an unhidden portion of a U-shape curve. The leaves of $\mathcal{T}$ correspond to the V-shape curves and the unhidden portions of the U-shape curves on the beach line sorted by their y coordinates. Each leaf also stores location information of the corresponding 3-D cone. The internal nodes of $\mathcal{T}$ adjacent to the leaves represent the breakpoints (i.e., the intersection of a pair of U or V curves) on the beach line. A breakpoint is stored at an internal node by an ordered tuple of curves $< C_i, C_j >$, where C_i is the left curve of the breakpoint and C_j is the right curve of the breakpoint. $\mathcal{T}_{\pi/4}$ is used to maintain the

orders (along the norm direction) of the 45° rays of all U- or V-shape curves which appear in the beach line. Similarly, $\mathcal{T}_{3\pi/4}$ maintains the orders of the 135° rays of U- or V-shape curves which appear in the beach line. Each leaf node in $\mathcal{T}_{\pi/4}$ or $\mathcal{T}_{3\pi/4}$ represents a 45° or 135° ray, and each ray corresponds to a U- or V-shape curve (represented by a leaf node in $\mathcal{T}$) in the beach line. For each leaf node of $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$, a pointer pointing to the corresponding leaf node in $\mathcal{T}$ is maintained. In this case, by doing binary search on $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ and the pointers between the trees, the positions in the beach line for the apex points of each newly encountered cone at a site event could be located in $O(\log N)$ time, and the beach line is updated in $O(k \log N)$ time, where k is the number of breakpoints destroyed and created after inserting the newly encountered U- or V-shape curve into the beach line.

The event queue $\mathcal{Q}$ is implemented by a priority queue, where the priority of an event is the x coordinate of the corresponding event point. If two points have the same x-coordinate, the one with larger y coordinate has the priority. All the site events and U-events are known in advance and are stored in $\mathcal{Q}$. The main challenge is to detect the V events.

For a V event, it is known that it occurs when the apex point of a hidden V-shape curve C_2 appears in the beach line. To detect such events, it is necessary to store in the data structure the information of all hidden V-shape curves. This could potentially require us to maintain the whole arrangement of all curves on Q and therefore results in unnecessarily high computational cost. To efficiently detect all possible V events, the main idea is not to maintain the arrangement, but rather to use the properties of V events to convert the problem into a query problem in 3-D. To achieve this, first it is easy to see that a V event is always caused by the upward movement of an unhidden portion of the bottom segment s_m of some U-shape curve C_1 and occurs when s_m coincides with the apex point v of a hidden V-shape curve C_2 (by Lemma 11). Thus, in order to detect all possible V events, it is needed to solve the following two difficulties: (1) Identify the next V event associated with each unhidden portion of a U-shape curve, and (2) for each newly encountered hidden V-shape curve (in a site event), find the unhidden portion of a U-shape curve which will later cause a V event involving this V-shape curve. There is another one (Difficulty (3)) related to the two difficulties: How to find the exact boundary of the dominating region for a given unhidden portion c of a U-shape curve.

First Difficulty (1) is considered. For this difficulty, it is known that the next V event associated with an unhidden portion c of a U-shape curve C_1 is the hidden V-shape curve C_2 whose apex point v lies inside the dominating region of c and is the closest to the bottom segment s_m of C_1. However, as it is noticed in the last section, the dominating region could have various shapes which seemingly suggest that it is costly to find the next V event even if the dominating region is known. To overcome this difficulty, first it is observed that the dominating region is bounded by 45° and 135° rays and the bottom segment. From Lemmas 9 and 10, it is known that the rays of any U or V curve have fixed directions (e.g., forming 45° and 135° angles with the y axis) and their supporting lines remain the same during the whole sweeping process. This suggests that the dominating regions can be orthogonalized in 3-D space. The three dimensions of the new

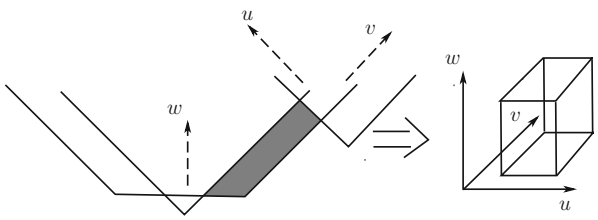

Fig. 12 A dominating region is converted to a 3-D box in the orthogonalized 3-D space for MD

space are the orthogonal directions of the 45° and 135° lines in the sweep plane Q and the orthogonal direction of the bottom segment (or the y axis). In this way, each dominating region is converted into a (possibly unbounded) box in 3-D (see Fig. 12).

To efficiently find the next V event in the orthogonalized dominating region, the apex points of all V-shape curves are processed into a 3-D dynamic range search tree data structure MD [29, 31]. In MD, the apex point of each hidden V-shape curve is mapped into a 3-D point. Thus, for a particularly dominating region R, its orthogonalized box can be used as the query range to find the closest apex point (among all hidden V-shape curves whose apex points fall in R) to the unhidden portion of the bottom segment of a U-shape curve in $O(\log^2 N \log\log N)$ time [29].

To efficiently maintain the MD, it can be built in advance for all possible V-shape curves. In each node p of the MD tree, a mark is stored to indicate whether there is any active V-shape curve in the subtree rooted at p. A V-shape curve C is active if its corresponding 3-D cone has already been encountered by the sweep plane Q, and C has not yet changed to a U-shape curve due to a U event. In this way, it is only needed to change the marks when the status of a V-shape curve changes and therefore avoid complicate updating (such as tree rotation) to the MD.

To make use of MD, it is necessary to identify all scenarios in which it is needed to either update MD or query MD for detecting potential V events. First, it is noticed that a site event or a U event could introduce (a) a new U-shape curve C to the beach line and generate a set of unhidden portions of C and (b) a hidden V-shape curve C'. Thus, for (a), in each such event, a 3-D range query in MD is performed for each unhidden portion c_i to find the closest hidden V-shape curve to c_i in its dominating region and insert a V event into the event queue $\mathcal{Q}$. For (b), it is necessary to find out the exact dominating region R which contains the apex point of the newly encountered hidden V-shape curve C' (i.e., Difficulty (2)) and then insert the hidden V event into MD, since the V event of C' might be the new next V event of the unhidden portion c of R. (The idea for this challenging problem will be discussed later.) Second, after a V event, it is also necessary to perform a 3-D range query in MD to find the closest hidden V-shape curve to the new U-shape curve converted from the V-shape curve of the V event. Third, if an unhidden portion c of a U-shape curve C disappears from the beach line (e.g., a circle event), its associated V event is deleted from $\mathcal{Q}$, since c will never appear in the beach line again by Lemma 11.

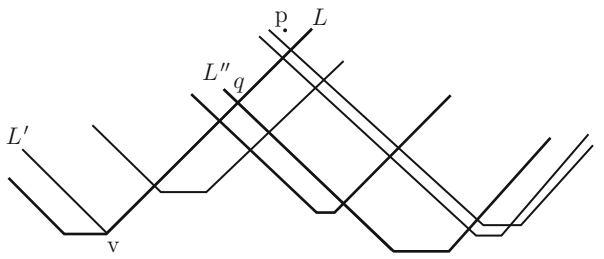

Fig. 13 Finding the dominating region of p

The ideas for Difficulty (2) (i.e., finding the unhidden portion of the dominating region containing the apex point of a hidden V-shape curve in a site event) are discussed below. Let p be the apex point of the hidden V-shape curve. As shown in Fig. 13, finding the four rays (two 45° rays and two 135° rays) bounding p does not necessarily give us the correct dominating region. This is because the rays bounding p could be stopped by the rays bounding the actual dominating region. For example, L is stopped by L'' at point q. Thus, a ray inside a dominating region may not be a ray bounding that dominating region.

Definition 5 A ray (or a portion of a ray) is active if it bounds some dominating region and inactive otherwise.

Let l be any ray in the beach line. If starting from the bottom (i.e., the endpoint) of l and walking along it, l is active until it is stopped by some other ray and thus becomes inactive. It is easy to see that once a ray becomes inactive, it will never be active again. In Fig. 13, L is active until stopped by L'' and will no longer be active. A ray has two sides and it may not be active on both sides simultaneously. Also one side of a ray may bound more than one dominating region (L'' bounds the dominating regions generated by U-shape curves with bottom segments a and c). It is easy to see that for any ray, there is always one side bounding at most one dominating region. Otherwise, there will be an intersection between two dominating regions, thus contradicting the definition of dominating regions.

With these observations, to find out the actual dominating region of p, first the four rays bounding it are found by searching the $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ trees. Further, it is needed to find out whether or not these rays are active on the sides containing p. To determine this, it is needed to know whether each of the four rays has been stopped by other rays. If the ray L which stops these rays can be found out, then it means that p belongs to the dominating region bounded by L. This means that for a given ray, it is needed to have a way to efficiently determine which ray stops it.

To achieve this, the $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ trees are augmented. For each node of the trees, the z-coordinate of the lowest bottom (segment) of all rays in the subtree rooted at that node is stored. The z-coordinate of the bottom segment of a V-shape curve is the z-coordinate of its apex point minus the length of bottom segment of the

3-D V cone. With the augmented information, the ray L'' which stops a ray L can be searched in the two trees in $O(\log N)$ time.

To better understand the idea, consider the example in Fig. 13. Let L be a 45° ray with its endpoint v. A ray L'' which stops L is a 135° ray if it exists. To find L'', another 135° ray L' with endpoint v is created. First the position of L' is searched in $\mathcal{T}_{3\pi/4}$. Then it is only needed to find the ray between L' and p (in the direction of L) with (1) lower z-coordinate than v and (2) the closet z-coordinate. This can be done by following the searching path of L' in $\mathcal{T}_{3\pi/4}$ upwards until finding a node satisfying the two conditions and then moving downwards to locate the ray L''. In Fig. 13, the dominating region with bottom segment b dominates p.

Lemma 14 *It takes $O(\log N)$ time to find out the exact dominating region of an unhidden portion c for the apex point p of the hidden V-shape curve C' generated by the site event.*

Proof Follows from the above discussion. □

It is not difficult to see that the augmented information can be maintained during the whole sweep process within the same time bound. With the augmented information, the exact boundary of a dominating region R for an unhidden portion c of a U-shape curve (i.e., Difficulty (3)) can also be found in $O(\log N)$ time following the same idea (i.e., finding the rays stopping the bounding rays of R).

Circle events can be handled in a way similar to the standard Voronoi diagram. More specifically, every new triple of consecutive U- or V-shape curves that appear in the beach line is checked. If such a new triple has converging breakpoints, the event is inserted into the event queue $\mathcal{Q}$. Furthermore, for all disappearing triples, the corresponding event is de-queued from $\mathcal{Q}$ if it has been inserted.

2.3.5 Algorithm and Analysis

The entire plane sweep algorithm is described below. The main steps of the algorithm are as follows.

Algorithm HAUSDORFF-VORONOI-DIAGRAM($\mathcal{C}$)
`Input`. A set $\mathcal{C}$ of axis-aligned rectangles (or geometric minimal cuts) in the plane.
`Output`. The Hausdorff Voronoi diagram in a doubly connected edge list $\mathcal{D}$.

1. Initialize the event queue $\mathcal{Q}$ with all site events and U events; initialize empty sweep plane status structures $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$, and an empty doubly-connected edge list $\mathcal{D}$; initialize a 3-D range search tree MD for all possible V-shape curves with all nodes marked as inactive.
2. **while** $\mathcal{Q}$ is not empty
3. **do** Remove an event with the smallest x-coordinate from Q.
4. **if** the event is a site event, **then** HANDLE-SITE-EVENT.
5. **if** the event is a circle event, **then** HANDLE-CIRCLE-EVENT.
6. **if** the event is a U-event, **then** HANDLE-U-EVENT.
7. **else** the event is a V event; HANDLE-V-EVENT.

HANDLE-SITE-EVENT

1. Let C be the new curve. If the status structure is empty, insert C into it and mark the apex point in MD as active if it is a V cone. Otherwise, continue with steps 2–8.
2. If C is a V-shape curve, mark its apex point in MD as active and continue with steps 3–5.
3. Locate the position of the apex point of C in the beach line by searching $\mathcal{T}$,$\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$.
4. If C is not hidden, insert C to the beach line by updating $\mathcal{T}$,$\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$. This includes inserting C into the beach line, computing new breakpoints, inserting possible circle events into $\mathcal{Q}$, and removing all curves hidden by C from the beach line. If an unhidden portion of a U-shape curve is removed, delete its corresponding V event from $\mathcal{Q}$. If an unhidden portion of a U-shape curve is partially blocked by C, search for its closest hidden V-shape curve in the reduced dominating region if necessary.
5. Else if (the apex of) C is inside the dominating region an unhidden portion c of a U-shape curve C', update the associated V event of c if needed.
6. Else if C is a U-shape curve, continue with steps 7–8.
7. Locate the position of C in the beach line by searching $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$.
8. If C is not fully hidden, insert C into the beach line by updating $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$. This includes computing possibly multiple breakpoints and the corresponding circle events for all its unhidden portions, removing blocked curves (similarly to Step 4), and finding the possible V event for each unhidden portion of C and partially blocked unhidden portion.

HANDLE-CIRCLE-EVENT

1. Update the beach line by updating $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$. Delete unnecessary circle events from Q involving the part disappearing from the beach line. If an unhidden portion of a U-shape curve moves out of the beach line, delete its associated V event.
2. Add the vertex to $\mathcal{D}$. Two new breakpoints of the beach line will be traced out.
3. Check the new triples involving the left or right neighbor of the disappearing part and insert the corresponding circle event into Q, if necessary.

HANDLE-U-EVENT

1. If the V-shape curve C introduced by the V cone appeared on the beach line, the corresponding part of the beach line is changed from a V-shape curve to a U-shape curve. Update $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$, and add vertex to $\mathcal{D}$. Also find the possible V event for C by querying MD, and insert it into $\mathcal{Q}$.
2. Mark the node in MD containing C as inactive, and update its ancestors if needed.
3. Detect the circle events and insert them into $\mathcal{Q}$ if necessary.

HANDLE-V-EVENT

1. Update $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$ by inserting the V-shape curve C into the beach line. Create new breakpoints, detect the possible circle event, and insert them into $\mathcal{Q}$.
2. For the corresponding unhidden portion c of a U-shape curve C', break it into two unhidden portions, c' and c'', and find possible new V event for c' and c'', respectively by querying MD.

Theorem 4 *The L_∞ Hausdorff Voronoi diagram $HVD(\mathcal{C})$ of a set $\mathcal{C}$ of geometric minimal cuts (or axis-aligned rectangles) can be constructed by a plane sweep algorithm in $O((N + K)\log^2 N \log\log N)$ time, where $N = |\mathcal{C}|$ and K is the complexity of the Hausdorff Voronoi diagram.*

Proof The discussion of the algorithm shows that $HVD(C)$ can be constructed by a plane sweep algorithm. Thus, the proof focuses on the time complexity.

First, site events are considered. Let C be a newly encountered U- or V-shape curve. If C is a V-shape curve, it is either part of the wavefront or a hidden V-shape curve. For the former case, the computation cost includes inserting C into the beach line and updating MD and $\mathcal{D}$. The cost for these is $O(\log^2 N \log\log N + k \log N)$, where k is the total number of breakpoints hidden by C. Since each breakpoint corresponds to a Voronoi vertex in $HVD(C)$, the cost of $O(k \log N)$ can be charged to the breakpoints (and their corresponding Voronoi edges). Thus, each will be charged a cost of $O(\log N)$. The cost of $(\log^2 N \log\log N)$ can be charged to each V-shape curve. For the latter case, if C is not in the dominating region of any unhidden portion of a U-shape curve, the only cost is $O(\log^2 N \log\log N)$, which can be charged to C. If C is inside the dominating region of some unhidden portion c of a U-shape curve which can be checked in $O(\log N)$ time, it is also needed to check whether C is the closest V-shape curve to c, and this can be done in $O(1)$ time. Thus, in the latter case, C is charged a cost of $O(\log^2 N \log\log N)$. If C is a U-shape curve, it could be fully hidden by the beach line, and thus, will never appear in the beach line in a later time. In this case, the total cost is $O(\log N)$, which can be charged to C. If C appears in the beach line and contributes some unhidden portions, then it is needed to update the sweep plane status structure, which takes $O(k \log N)$ time, and find the closest V-shape curve for each unhidden portion c. The cost of $O(k \log N)$ can be evenly charged to all hidden and newly created breakpoints. Thus, each of them will be charged a cost of $O(\log N)$. The closest V-shape curve to each unhidden portion c can be found by a query to MD, which takes $O(\log^2 N \log\log N)$ time and can be charged to the breakpoint bounding c. In a site event, up to two unhidden portions can be partially hidden by the newly encountered U- or V-shape curve C. In this case, each of them may need to find a new closest hidden V-shape curve in its reduced dominating region. For this, the cost of $O(\log^2 N \log\log N)$ is charged to the new breakpoint created by the two rays of C and the two unhidden portions. Thus, after processing all site events, each V-shape curve will be charged a cost of $(\log^2 N \log\log N)$, and each U-shape curve will be charged a cost of $O(\log N)$. Some breakpoints will be charged a cost of $O(\log^2 N \log\log N)$.

Clearly, each circle event can be handled in $O(\log N)$-time, and the total number of such events is bounded by $O(K)$. Thus, the total cost for all circle events is $O(K \log N)$.

For U events, there are only $O(N)$ such events. Each event takes $O(\log^2 N \log\log N)$ time to update MD, and find the closest hidden V-shape curve to the new U-shape curve. Again, all the cost is charged to the V-shape curve.

For V events, it is clear from the algorithm, that each such event takes $O(\log^2 N \log\log N)$ time. To bound the total cost of all V events, it is needed to bound their total number. For this, it is noticed that at any moment, (1) Each unhidden portion of a U-shape curve is associated with only one hidden V-shape curve; and (2) The association of a hidden V-shape curve C and an unhidden portion c of a U-shape curve can change for only two reasons: (a) The dominating region of c changes (in a site event), and (b) a new hidden V-shape curve C' is activated and C' is closer to c than C. For (a), it means that part of c is hidden by other U- or V-shape curve C'' and the cost of de-association can be charged to the edges or vertices introduced by C''. For (b), the cost of de-association can be charged to C'. Each V-shape will be charged no more than once. This means that computation cost of each V event can be charged to the Voronoi edge or vertex corresponding to the two breakpoints bounding the unhidden portion. Each Voronoi vertex and edge is charged a constant times with a total cost of $O(\log^2 N \log\log N)$. Thus, the total cost for all V events is $O(K \log^2 N \log\log N)$.

Putting all together, the total cost is $O((N + K)\log^2 N \log\log N)$. Thus, the theorem follows. □

3 The MGMC Problem via Higher-Order Voronoi Diagrams

In this section, we present the iterative approach of [34] to the MGMC problem via higher-order Voronoi diagrams. A more general version of the MGMC problem is addressed in this section, where the embedded subgraph H corresponds to arbitrary polygonal shapes. The polygonal shapes are not assumed rectilinear, but they may consist of edges in arbitrary orientation. A polygonal shape is reduced to a collection of (additively) weighted line segments by means of its L_∞ medial axis. The approach is complementary to the one presented in Sect. 2 and it does not precompute any minimal cuts. On the contrary, it discovers those geometric minimal cuts that are guaranteed to participate in the final map, on the fly, by iteratively constructing variants of higher-order Voronoi diagrams. At the end of the iteration, the derived subdivision is the Hausdorff Voronoi diagram of all geometric cuts, that is, the solution to the MGMC problem. Definitions, results, and most figures in this section are reproduced from [34].

3.1 The MGMC Problem in a VLSI Layout

In a VLSI setting, the *geometric min-cut* problem is as follows [34]: We are given a collection of geometric graphs that have portions embedded on a plane (a VLSI layer) A. The embedded portions on plane A are vulnerable to random defects that may form *cuts* on the given graphs. A defect of size r is a disk of radius r. The size of a geometric cut C at a given point t is the size of the smallest defect centered at t that overlaps all elements in C, as opposed to the number of edges in C in the classic min-cut problem. Compute, for every point t on the vulnerable plane A, the

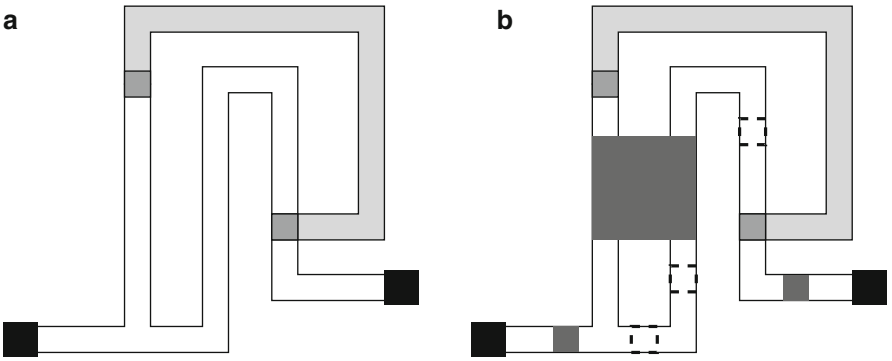

Fig. 14 (**a**) A net N spanning over two layers. (**b**) Dark defects create opens while transparent defects cause no faults

size of the minimum defect causing a geometric cut at t. The size of the minimum geometric cut at a point t is the *critical radius* for open faults at t. The resulting subdivision is referred to as the *opens Voronoi diagram*, and it corresponds to the map of the MGMC problem.

Figure 14a, reproduced from [34], illustrates an example of a simple VLSI net N spanning over two metal layers, say M1 and M2, where M2 is illustrated shaded. The two contacts illustrated as black squares are designated as *terminal shapes*. A VLSI net remains functional as long as terminal shapes remain interconnected. In Fig. 14b, defects that create open faults are illustrated as dark squares, and defects that cause no fault are illustrated hollow in dashed lines. Hollow defects do break wires of layer M1; however, they do not create opens as no terminals get disconnected.

A compact graph representation for a VLSI net N, denoted $G(N)$, can be obtained as follows: Each graph node of $G(N)$ represents a connected component of a net N on a conducting layer, and two nodes are connected by an edge if there exists at least one contact or via connecting the respective components of N. Some of the shapes constituting net N are designated as *terminal* representing the entities that the net must interconnect. A node containing terminal shapes is designated as a *terminal node*.

Given a layer A of interest, the *extended graph* of N on A, $G(N, A)$, is derived from $G(N)$ by expanding the components of N on layer A by their medial axis. Contact points between neighboring layers are approximated as points on the medial axis of the corresponding shape, called *via points*. Any via point or any portion of the medial axis corresponding to terminal shapes is identified as terminal. Figure 15a illustrates $G(N, A)$, where $A = M1$, for the net of Fig. 14; terminal points are indicated by hollow circles; dashed lines represent the original M1 polygon and they are not part of $G(N, A)$. $G(N, A)$ can be cleaned up from any trivial parts [34] by computing biconnected components, bridges, and articulation points. In the following, we assume that $G(N, A)$ is free from any trivial parts, as shown in Fig. 15b. The collection of $G(N, A)$ for all nets N involved on a layer A is the graph

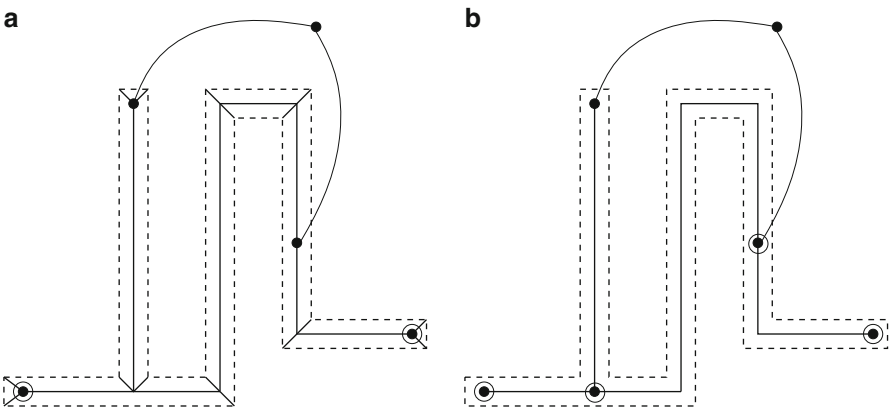

Fig. 15 The net graph of Fig. 14 before (**a**) and after (**b**) clean-up of trivial parts

of the MGMC problem, and its portion on layer A is the planar embedded subgraph H. For more details on the derivation of $G(N, A)$, see [34].

3.2 Review of Concepts on L_∞ Voronoi Diagrams

The Voronoi diagram of a set of polygonal sites in the plane is a partitioning of the plane into regions, one for each site, called *Voronoi regions*, such that the Voronoi region of a site s is the locus of points closer to s than to any other site. The Voronoi region of s is denoted as $reg(s)$ and s is called the *owner* of $reg(s)$. In the interior of a simple polygon, the Voronoi diagram is known as the *medial axis* of the polygon (a minor difference in the definition is ignored (see [26])). The boundary between two Voronoi regions is called a *Voronoi edge* and it consists of portions of *bisectors* between the owners of the neighboring regions. The bisector of two polygonal objects (such as points, segments, polygons) is the locus of points equidistant from the two objects. A point where three or more Voronoi edges meet is called a *Voronoi vertex*.

The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is $d(p, q) = \max\{|x_p - x_q|, |y_p - y_q|\}$. In the presence of additive weights, the (weighted) distance between p and q is $d_w(p, q) = d(p, q) + w(p) + w(q)$, where $w(p)$ and $w(q)$ denote the weights of points p and q, respectively. In case of a weighted line l, the (weighted) distance between a point t and l is $d_w(t, l) = \min\{d(t, q) + w(q), \forall q \in l\}$. The (weighted) bisector between two polygonal elements s_i and s_j is $b(s_i, s_j) = \{y \mid d_w(s_i, y) = d_w(s_j, y)\}$.

In L_∞, any Voronoi edge or vertex can be treated as an additively weighted line segment. For brevity and in order to differentiate from ordinary line segments, the term *core segment*, more generally *core element*, is used to denote any portion of interest along an L_∞ Voronoi edge or vertex. The term *standard*-45°s is used

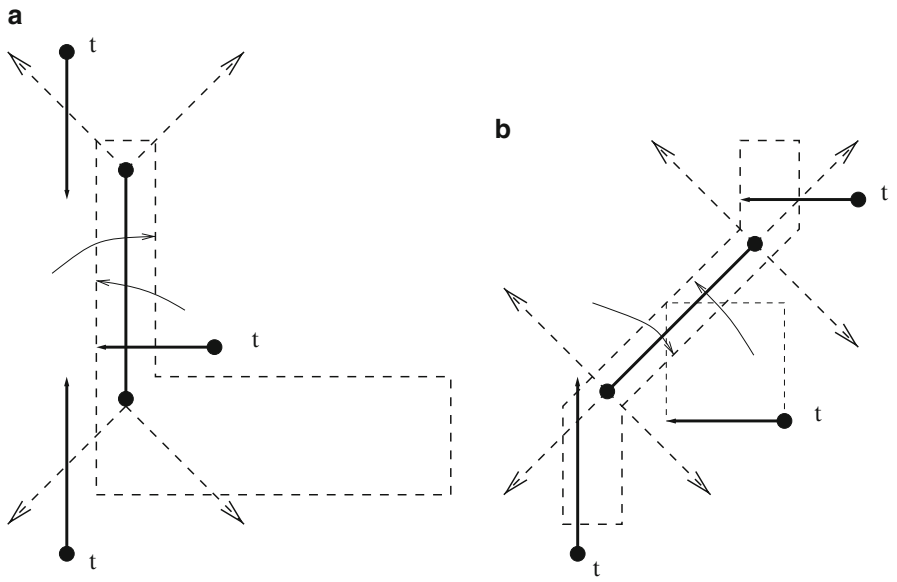

Fig. 16 The regions of influence of the core elements of a core segment: two endpoints and an open line segment. (**a**) an axis-parallel core segment, (**b**) a non-axis-parallel core segment

to refer to Voronoi edges of slope ± 1 that correspond to bisectors of axis-parallel lines. Figure 16 illustrates examples of core segments.

Let s be a core segment induced by the polygonal elements e_l, e_r. Every point p along a core segment s is weighted with $w(p) = d(p, e_l) = d(p, e_r)$, where e_l, e_r are the polygonal elements inducing s. The 45° rays[3] emanating from the endpoints of s partition the plane into the regions of influence of either the open core segment portion or the core endpoints. In Fig. 16, the L_∞ distance is indicated by straight-line arrows emanating from various points t. In the north and south (resp. east and west) regions, the L_∞ distance simplifies to a vertical (resp. horizontal) distance. In the region of influence of a non-axis-parallel core segment, it is measured by the side of a square touching e_i as shown in Fig. 16b. In the region of influence of a core point p, distance is measured in the ordinary weighted sense, that is, for any point t, $d_w(t, p) = d(t, p) + w(p)$. In the region of influence of an open core segment s, distance, in essence, is measured according to the farthest polygonal element defining s, that is, $d_w(t, s) = d(t, e_i)$, where e_i is the polygonal element at the opposite side of s than t; see, for example, the small arrows in Fig. 16. In L_∞, this is equivalent to the ordinary weighted distance between t and s. For more details see [34].

The (weighted) bisector between two core elements is defined in the ordinary way, always taking the weights of the core elements into consideration. Similarly, the (weighted) Voronoi diagram of a set of core elements is defined as above,

[3] A 45° ray is a ray of slope ± 1.

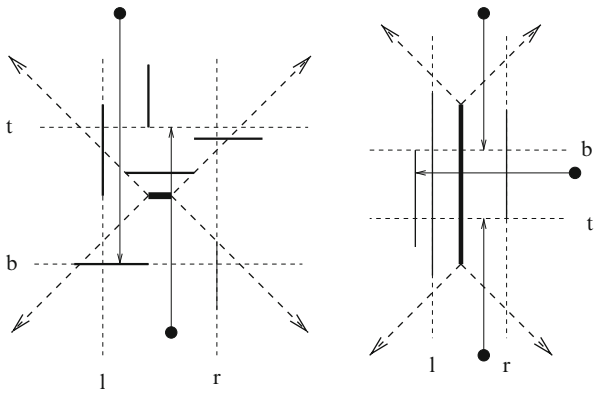

Fig. 17 The L_∞ farthest Voronoi diagram of axis parallel segments

with the difference that distance between a point t and a core element s is always measured in an additive weighted sense, $d_w(t, s)$. The (weighted) Voronoi diagram of *core* medial axis segments was first introduced in [32] as a solution to the critical area computation problem for a simpler notion of an open, called *break*, that was based solely on geometric information. For Manhattan geometries, core segments are simple (additively weighted) axis-parallel line segments and points.

The *farthest Voronoi diagram* of a set of polygonal sites is a partitioning of the plane into regions, such that the *farthest Voronoi region* of a site s is the locus of points *farther away* from s than from any other site. For typical cases (e.g., points, line segments), it is a tree-like structure consisting only of unbounded regions (see, e.g., [4, 6]). In the L_∞ metric, when sites are points or axis-parallel segments, its structure is particularly simple, consisting of exactly four regions as shown in Fig. 17. In each region, the L_∞ distance to the farthest element is measured as the vertical or horizontal distance to an axis-parallel line marked by t, b, l, r, where t (resp. b) is the horizontal line through the topmost (resp. bottommost) lower (resp. upper) endpoint of all core segments and l (resp. r) is the vertical line through the leftmost (resp. rightmost) right (resp. left) endpoint of all core segments. In Fig. 17, the thin arrows indicate the farthest L_∞ distance of selected points.

3.3 Definitions and Problem Formulation

Let $core(N, A)$ denote the collection of all core elements of a net N on a layer A, that is, the collection of all medial axis vertices and edges of $G(N, A)$ on A that are of interest. For simplicity, all standard 45° edges are excluded from $core(N, A)$. The union of $core(N, A)$ for all nets N on layer A is denoted as $core(A)$. Core segments and points in $core(A)$ represent all wire segments vulnerable to defects on layer A and correspond to weighted line segments. Core segments are assumed to consist of three distinct core elements: two endpoints and an open line segment. Open faults are determined based on the connectivity information of $G(N, A)$ and the geometry information of $core(N, A)$.

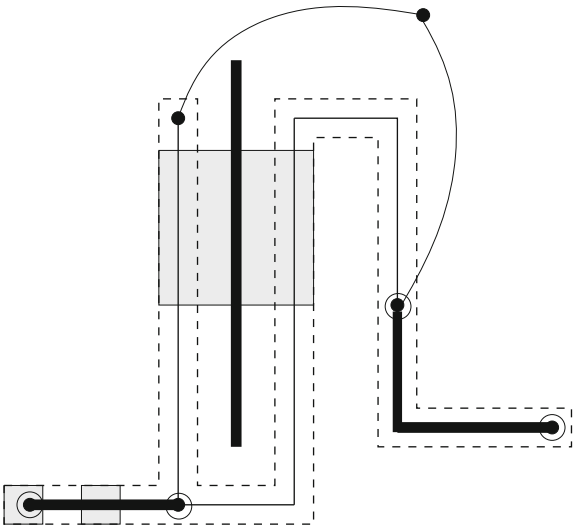

Fig. 18 Generators for strictly minimal opens

Definition 6 A *minimal open* is a defect D of minimal size that breaks a net N, that is, if D is shrunk by $\epsilon > 0$, then D no longer breaks N. An *open* is any defect that entirely overlaps a minimal open. A minimal open is called *strictly minimal* if it contains no other open in its interior.

Definition 7 The center point of an open D, is called a *generator point* for D and it is weighted with the radius of D. The generator of a strictly minimal open is called *critical*. A segment formed as the union of generator points is called a *generator segment* or simply a *generator*.

In Fig. 14, strictly minimal opens are illustrated by dark shaded disks, other than the original via and contact shapes. Figure 18 illustrates the generators for strictly minimal opens for the net of Fig. 14, thickened; the shaded squares indicate strictly minimal opens. For brevity, we say that a defect D *overlaps* a core element $c \in core(A)$, but we mean that D overlaps the entire width of the wire segment induced by c.

Definition 8 A *cut* for a net N is a collection C of core elements, $C \subset core(N, A)$, such that $G(N, A) \backslash C$ is disconnected leaving nontrivial articulation or terminal points in at least two different sides. A cut C is called minimal if $C \backslash \{c\}$ is not a cut for any element $c \in C$. A defect of minimal size that overlaps all elements of cut C is called a *cut-inducing* defect. The centerpoint p of a cut-inducing defect that encloses no other defect in its interior is called a *generator point* for cut C. If, in addition, the cut-inducing defect is a strictly minimal open, then p is called *critical*. The collection of all generator points of a cut C is referred to as the *generator(s)* of C.

The generator of a cut C can consist of *critical* and *noncritical* portions. Critical portions correspond to generators of strictly minimal opens. Noncritical portions correspond to centers of cut-inducing disks that, in addition to overlapping C, may also overlap some additional cut on a layer A, and thus, although they break C, they do not correspond to strictly minimal opens.

Definition 9 Generators of minimal cuts that consist of a single core element are called *first-order generators*. Generators of minimal cuts that consist of more than one core element are called *higher-order generators*. The set of all critical generators on layer A is denoted as $G(A)$.

In Fig. 18 first-order generators are illustrated as the thickened core segments in the interior of polygons; the vertical thick segment in the exterior of polygons is a higher-order generator that involves a pair of core elements. Given $G(N, A)$, we can detect *biconnected components,*[4] *bridges*, and *articulation points*[5] using *depth-first search* (DFS) as described in [21, 42]. By definition, we have the following property.

Lemma 15 *The set of* first-order generators *on a layer A, denoted as $G_1(A)$, consists of all the bridges, terminal edges, articulation points, and terminal points of $G(N, A) \cap core(N, A)$, for all nets N. All first-order generators are critical.*

The generator of a minimal cut C that consists of more than one core element must be a subset of the L_∞ farthest Voronoi diagram of C, derived by ignoring the standard-45° edges of the diagram. For Manhattan geometries, the generator of any cut is always a single axis-parallel segment (that can degenerate to a point); see, for example, Fig. 17. Any generator point p of a cut C is weighted with $w(p) = \max\{d_w(p, c), \forall c \in C\}$. The disk D centered at p of radius $w(p)$ is clearly an open. If in addition D is strictly minimal, then p is a critical generator.

Definition 10 The *Voronoi diagram for opens* on a layer A is a subdivision of A into regions, such that the *critical radius* of any point t in a Voronoi region is determined by the owner of the region. The critical radius of a point t, $r_c(t)$, is the size (radius) of the smallest defect centered at t and causing an open.

The following theorem is easy to see by properties of Voronoi diagrams. For a proof, see [34].

[4] A biconnected component of a graph G is a maximal set of edges, such that any two edges in the set lie on a common simple cycle.

[5] An articulation point (resp. bridge) of a graph G is a vertex (resp. edge) whose removal disconnects G.

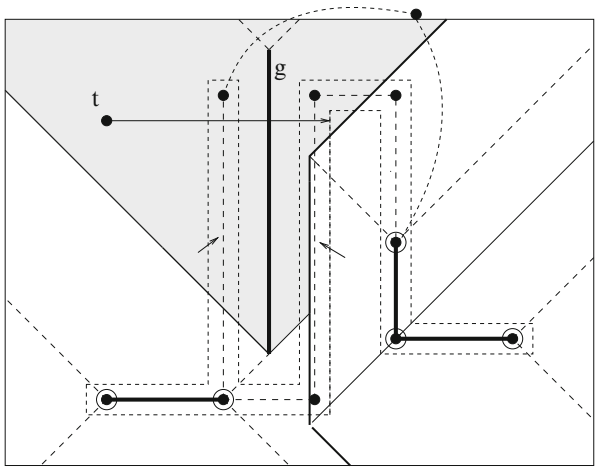

Fig. 19 The Voronoi diagram for open faults on layer A. The shaded region illustrates the Voronoi region of the higher-order generator g

Theorem 5 *The* Voronoi diagram *for open faults on a layer A corresponds to the (weighted) Voronoi diagram of the set $G(A)$ of all critical generators for strictly minimal opens on A, denoted as $\mathcal{V}(G(A))$.*

Figure 19 illustrates the opens Voronoi diagram for the net of Fig. 14. The shaded region illustrates the Voronoi region of a higher-order generator g, $reg(g)$. Generator g is the critical generator of a cut consisting of two core segments as indicated by two small arrows. The critical radius of a sample point t in $reg(g)$ is indicated by the arrow emanating from t.

Corollary 3 *Given the opens Voronoi diagram, the critical radius of any point t in the region of a generator g is $r_c(t) = d_w(t, g)$. If g is a higher-order generator of cut C, then $r_c(t) = d_w(t, g) = \max\{d_w(t, c), \forall c \in C\}$.*

In the following section, the Voronoi diagram for opens is formulated as a special higher-order Voronoi diagram of elements in $core(A)$.

3.4 A Higher-Order Voronoi Diagram Modeling Open Faults and the MGMC Problem

Let $\mathcal{V}(A)$ denote the (weighted) Voronoi diagram of the set $core(A)$ of all core elements on a plane (layer) A. If there were no loops associated with A, then $\mathcal{V}(A)$ would provide the opens Voronoi diagram on A, and $core(A)$ would be the set of all critical generators. $\mathcal{V}(A)$ for Manhattan layouts has been defined in detail in [32]. Figure 20 illustrates $\mathcal{V}(A)$ for the net graph of Fig. 14. The arrows in Fig. 20 illustrate several minimal radii of defects that break a wire segment. Given a point t in the region of generator s, $d_w(t, s)$ gives the radius of the smallest defect centered at t

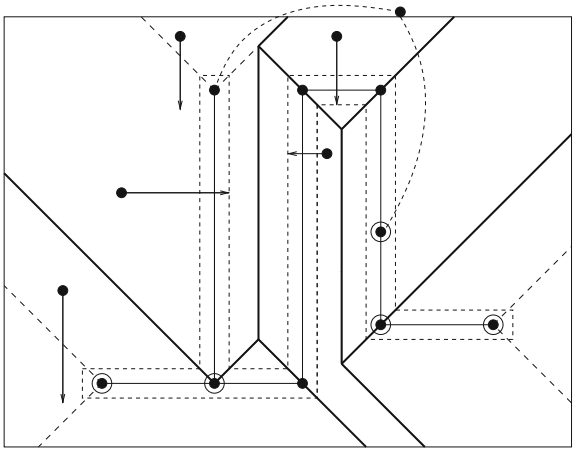

Fig. 20 The L_∞ Voronoi diagram of $core(A)$ on layer A, $\mathcal{V}(A)$

that overlaps the wire segment induced by s. Assuming no loops, $d_w(t, s)$ would be the critical radius of t.

Once loops are taken into consideration, only bridges, articulation, and terminal points, among the elements of $core(A)$, correspond to critical generators. Let us augment $\mathcal{V}(A)$ with information reflecting critical generators. In particular, the regions of first-order generators get colored red reflecting the regions of critical generators. The critical radius of point t in a red region of owner s is $r_c(t) = d_w(t, s)$. In Fig. 21, red regions are shown shaded and critical generators are shown thickened.

Let us now define the order-k Voronoi diagram on the plane A, denoted as $\mathcal{V}^k(A)$. For $k = 1$, $\mathcal{V}^k(A) = \mathcal{V}(A)$. Following the standard definition of higher-order Voronoi diagrams, a region of $\mathcal{V}^k(A)$ corresponds to a maximal locus of points with the same k nearest neighbors among the core elements in $core(A)$. The open portion of a core segment and its two endpoints count as different entities. A kth-order Voronoi region, $k > 1$, belongs to a k-tuple C representing the k nearest neighbors of any point in the region of C. The region of C is denoted $reg(C)$, and it is further subdivided into finer subregions by the farthest Voronoi diagram of C. For any point $t \in reg(C)$, $d(t, C) = \max\{d(t, c), \forall c \in C\}$. If C constitutes a cut of a net N, then the region of C is colored red.

In order to appropriately model open faults, the above standard definition is slightly modified, and in certain cases fewer than k elements are allowed to own a Voronoi region of order k. In the following, the term *k-th order Voronoi diagram* implies the modified version of the diagram as follows:

- A red region corresponds to a maximal locus of points with the same r, $1 \leq r \leq k$, nearest neighbors, C, among the core elements in $core(A)$, such that C constitutes a minimal cut for some net N.
- Any time a core segment s and one of its endpoints p participate in the same set C of nearest neighbors, s is discarded from C; this is because $d(t, p) \geq d(t, s)$

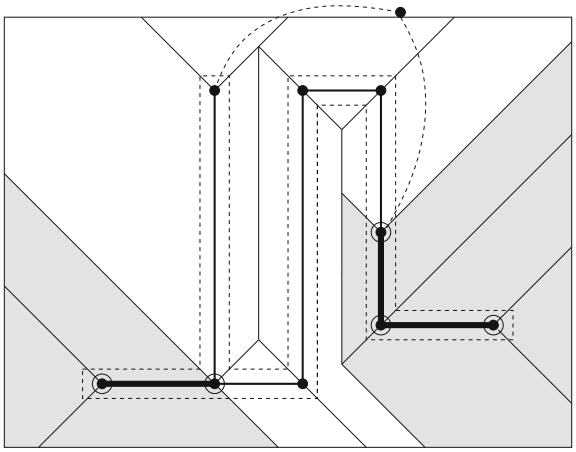

Fig. 21 The first-order opens Voronoi diagram on layer A, $\mathcal{V}^1(A)$. Shaded regions belong to first-order generators and the critical radius of any point within is determined by the region owner

$\forall t \in reg(C)$. Intuitively, a defect that destroys a core endpoint automatically also destroys all incident core segments, but not vice versa.

Figures 22 and 23 illustrate $\mathcal{V}^2(A)$ and $\mathcal{V}^3(A)$, respectively, for the net of our example. kth-order Voronoi regions are illustrated in solid lines; red regions are illustrated shaded. The darker shaded region in $\mathcal{V}^2(A)$ shows the 2nd-order red region of a pair of core segments that constitute a cut. The thick dashed lines indicate the farthest Voronoi diagram subdividing a kth-order region. In a red region, the thick dashed lines (excluding standard 45°s) correspond to critical generators. All critical generators are indicated thickened; solid ones are first-order generators and dashed ones in red regions are higher-order generators. All thin dashed lines in Figs. 21–23 can be ignored. Due to our conventions, the Voronoi region of any core endpoint p in $\mathcal{V}^1(A)$ remains present in $\mathcal{V}^2(A)$ and expands into the regions of the core segments incident to p.

Theorem 6 *The Voronoi diagram for opens on a layer A is the minimum-order m Voronoi diagram of $core(A)$, $\mathcal{V}^m(A), m \geq 1$, such that all regions of $\mathcal{V}^m(A)$ are colored red. Any region $reg(H)$, where $|H| > 1$, is subdivided into finer regions by the farthest Voronoi diagram of H. The critical radius for any point t in $reg(H)$ is $r_c(t) = d_w(t, H) = \max\{d_w(t, h), h \in H\}$.*

For a proof of Theorem 6, see [34]. Figure 24 illustrates the opens Voronoi diagram, for our example; arrows indicate the critical radius of several points; all critical generators are indicated in thick solid lines. Note that in L_∞, the Voronoi subdivision is not unique but depends on the conventions used on how to distribute regions that are equidistant from collinear elements on axis-parallel lines. Critical area calculations are immune to such differences, however, they may have an effect on the number of iterations needed to compute the opens Voronoi diagram. The adapted convention is that critical generators have priority over non-critical ones and regions equidistant from a critical and a noncritical generator are assigned to the critical one and get colored red.

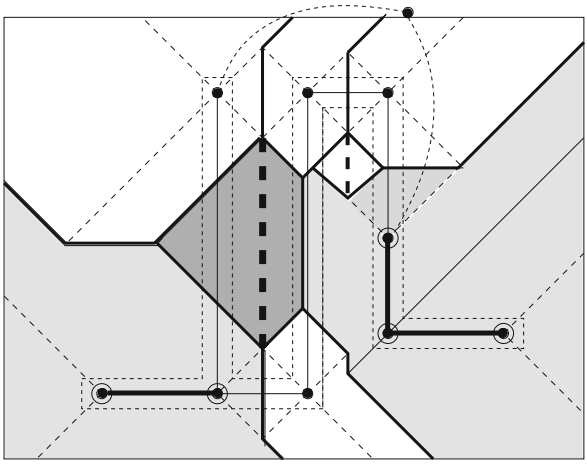

Fig. 22 The second-order opens Voronoi diagram, $\mathcal{V}^2(A)$. The darker shaded region belongs to a pair of core segments forming a cut

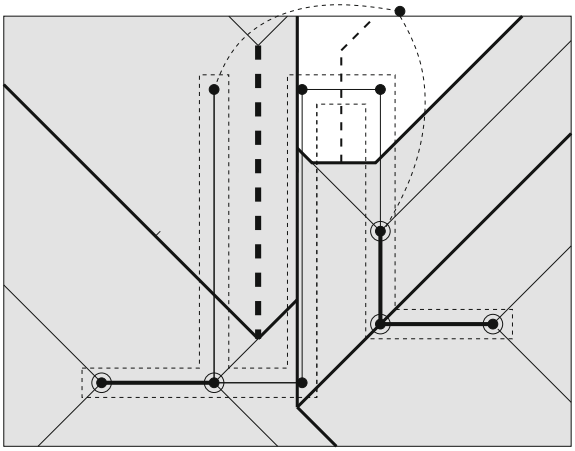

Fig. 23 The third-order opens Voronoi diagram, $\mathcal{V}^3(A)$

Corollary 4 *The higher-order critical generators on a layer A are exactly the farthest Voronoi edges and vertices, excluding the standard-45° Voronoi edges, constituting the farthest Voronoi subdivisions in the interior of each region in $\mathcal{V}^m(A)$. All higher-order critical generators are encoded in the graph structure of $\mathcal{V}^k(A)$, for some k, $1 \leq k < m$.*

Let $G(A)$ denote the set of all critical generators on layer A, including first-order and higher-order generators. Let us classify higher-order critical generators according to the minimum-order-k Voronoi diagram they first appear in. In particular,

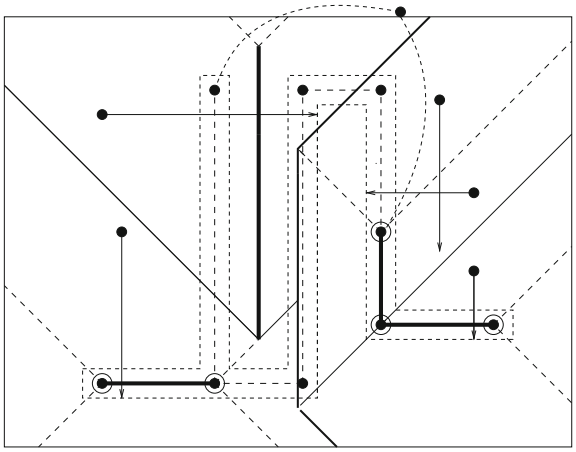

Fig. 24 The Voronoi diagram for open faults on a layer A. *Arrows* illustrate the critical radius of several points

higher-order generators encoded in $\mathcal{V}^k(A)$ are classified as $(k+1)$-*order generators* and they are denoted as $G_{k+1}(A), 1 \leq k < m$. Let $G(A) = \cup_{1 \leq i \leq m} G_i(A)$. By Theorems 5 and 6, $\mathcal{V}(G(A))$ and $\mathcal{V}^m(A)$ are identical.

3.4.1 An Approximate Opens Voronoi Diagram

Given any subset $G'(A)$ of the set of critical generators $G(A)$, the (weighted) Voronoi diagram of $G'(A)$ can be used as an approximation to $\mathcal{V}(G(A))$. Clearly, the more critical generators are included in $G'(A)$, the more accurate the result is. In practice, we can derive $G'(A)$ as $\cup_{1 \leq i \leq k} G_i(A)$, including all ith-order generators up to a small constant k. Since the significance of critical generators reduces drastically with the increase in their order, $\mathcal{V}(G'(A))$ should be sufficient for critical area computation for all practical purposes.

Corollary 5 *Let $G'(A) = \cup_{1 \leq i \leq k} G_i(A)$ be a subset of critical generators including all generators up to order k for a given constant k. The (weighted) Voronoi diagram of $G'(A)$, $\mathcal{V}(G'(A))$, can serve as an approximation to the opens Voronoi diagram. If $G'(A) = G(A)$, then the two diagrams are equivalent.*

Figure 25 illustrates the (weighted) Voronoi diagram of $G_1(A)$. $\mathcal{V}(G_1(A))$ reveals critical radii for opens under the (false) assumption that all loops are immune to open faults. In Fig. 25, solid arrows indicate selected critical radii as derived by $\mathcal{V}(G_1(A))$, while dashed arrows indicate true critical radii. Several critical radii can be overestimated in $\mathcal{V}(G_1(A))$, resulting in underestimating the total critical area for open faults. As k increases, however, $\mathcal{V}(\cup_{1 \leq i \leq k} G_i(A))$ converges fast to $\mathcal{V}(G(A))$ (see, e.g., Sect. 8 of [34] for experimental results).

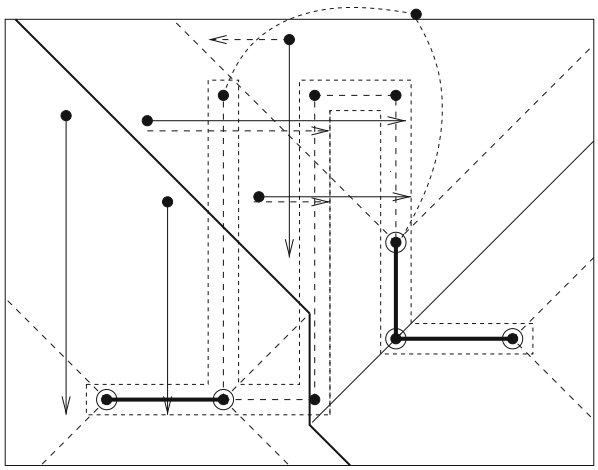

Fig. 25 $\mathcal{V}(G_1(A))$ as an approximate opens Voronoi diagram

3.5 Computing the Opens Voronoi Diagram

In this section, we give algorithmic details on how to compute $G_{k+1}(A)$ and $\mathcal{V}^{k+1}(A)$, given $\mathcal{V}^k(A)$, for $1 \leq k < m$. We also discuss how to compute $\mathcal{V}(\cup_{1\leq i\leq m} G_i(A))$ and $\mathcal{V}(G(A))$.

The iterative process to compute higher-order generators and higher-order opens Voronoi diagrams. Let's first discuss how to identify the set $G_{k+1}(A)$ of $(k+1)$-order generators, given $\mathcal{V}^k(A)$, for $k \geq 1$. The following property is shown in [34].

Lemma 16 *A Voronoi edge g that bounds two non-red Voronoi regions $reg(H)$ and $reg(J)$ in $\mathcal{V}^k(A)$ corresponds to a critical generator if and only if both the core elements $h \in H$ and $j \in J$ that induce g ($g \in b(h, j)$) are part of the same biconnected component B, and in addition, $H \cup J$ corresponds to a* cut *of B, that is, removing $H \cup J$ from B disconnects B, leaving articulation points in at least two sides. No Voronoi edge bounding a red region can be a critical generator.*

To determine if Voronoi edge g is a critical generator, we need to pose a connectivity query to biconnected component B after removing $H \cup J$. To perform connectivity queries efficiently, we can use the fully dynamic connectivity data structures of [20], which support edge insertion and deletions in $O(\log^2 n)$-time, while they can answer connectivity queries fast. For simplicity in the implementation, instead of employing dynamic connectivity data structures, reference [34] gives a simple (almost brute force) algorithm as follows: Remove the elements of H from B and determine new nontrivial bridges, articulation points, and biconnected components of $B \setminus H$. For any Voronoi edge g bounding $reg(H)$, where g is portion of $b(h, j)$, $h \in H$, $j \in J$, g is a critical generator if and only if j is a new non-trivial

bridge or articulation point of $B \setminus H$. Generator g gets associated with the tuple of core elements $H \cup J$, simplified, in case j or h are core endpoints, by removing any core segment incident to j, h.

The above process can be considerably simplified in the special case where the biconnected component B is a simple cycle. In this case, a simple coloring scheme in the DFS tree of B can efficiently identify all cuts of B that may be associated with a second-order generator. The time complexity of determining $G_{k+1}(A)$, given $\mathcal{V}^k(A)$, is summarized in the following lemma. Note that the size of $\mathcal{V}^k(A)$ is $O(k(n-k))$ (see [26]).

Lemma 17 *The $(k+1)$-order generators can be determined from $\mathcal{V}^k(A)$ in time $O(kn\log^2 n)$ using the dynamic connectivity data structures of [20] or in time $O(kn^2)$ using the simple algorithm presented above. In case of biconnected components forming simple cycles, second-order generators can be determined from $\mathcal{V}(A)$ in $O(n)$ time.*

Let us now discuss how to obtain $\mathcal{V}^{k+1}(A)$ from $\mathcal{V}^k(A)$, $k \geq 1$. The following is an adaptation of the iterative process to compute higher-order Voronoi diagrams of points [26], to the case of (weighted) segments. Let $reg(H)$ be a non-red region of $\mathcal{V}^k(A)$. Let $N(H)$ denote the set of all core elements that induce a Voronoi edge bounding $reg(H)$ in $\mathcal{V}^k(A)$.

1. Compute the (weighted) L_∞ Voronoi diagram of $N(H)$ and truncate it within the interior of $reg(H)$; this gives the $(k+1)$-order subdivision within $reg(H)$. Each $(k+1)$-order subregion of $reg(H)$ is attributed to a tuple $J = H \cup \{c\}$, $c \in N(H)$. In case c is a core endpoint incident to a core segment s in H, J simplifies to $J = H \setminus \{s\} \cup \{c\}$. In case c is part of a cut C owning a neighboring red region of $\mathcal{V}^k(A)$, the subregion of J gets colored red and gets as owner the cut C.
2. Once the $(k+1)$-order subdivision within all non-red regions neighboring $reg(H)$ has been performed, merge any incident $(k+1)$-order subregions that belong to the same tuple of owners J into a maximal $(k+1)$-order region, $reg(J)$. The edges of $\mathcal{V}^k(A)$ included within $reg(J)$ constitute the finer subdivision of $reg(J)$ by its farthest Voronoi diagram. All $(k+1)$-order red subregions are merged into the neighboring red regions of $\mathcal{V}^k(A)$ forming the maximal red regions of $\mathcal{V}^{k+1}(A)$.

Using established bounds for higher-order Voronoi diagrams of points (see, e.g., [26]) we conclude the following.

Lemma 18 *$\mathcal{V}^{k+1}(A)$ can be computed from $\mathcal{V}^k(A)$ in time $O(k(n-k)\log n)$, plus the time $T(k,n)$ to determine the $(k+1)$-order generators, where $T(k,n)$ is as given in Lemma 17.*

3.5.1 Computing the Opens Voronoi Diagram from Critical Generators

The iterative process of Sect. 3.5 can continue until all regions are colored red and the complete opens Voronoi diagram (i.e., the map of the MGMC problem) is guaranteed to be available. By Lemmas 17 and 18, this results in

an $O(n^3 \log n(\log\log n)^3)$-time algorithm. In practice, however, this would be unnecessarily inefficient. Note that the iterative process may continue for several rounds without any new critical generators being identified, only the regions of existing critical generators keep enlarging into neighboring non-red regions. Note also that as the number of iterations k increases, the weight of order-k critical generators (if any) increases as well, and their contribution to the total critical area drastically reduces. In practice, we can restrict the number of iterations to a small predetermined constant k, or to a small number determined adaptively, and compute only a sufficient set of critical generators $G'(A) = \cup_{1\le i\le k} G_i(A)$. We can then use Theorem 5 to report $\mathcal{V}(G'(A))$ as an approximate opens Voronoi diagram. The overall algorithm can be broken into two independent parts:

– Part I: Compute the set of critical generators $G'(A) = \cup_{1\le i\le k} G_i(A)$, up to a given (or adaptively determined) order k.
– Part II: Compute the (weighted) Voronoi diagram of $G'(A)$, $\mathcal{V}(G'(A))$, as the opens Voronoi diagram.

Part I can be performed using the iterative process of Sect. 3.5. Experimental results in [34] suggest that $k = 2$ is often adequate and no $k > 4$ is ever needed. Alternatively, k can be determined adaptively, for example, to the first round such that no new critical generators are determined. Part II can be performed using a plane sweep algorithm for computing $\mathcal{V}(A)$. Critical generators have similar properties to the core elements of $core(A)$, and the same plane sweep algorithm can be used to compute either (see [32, 36]). The computations of Parts I and II can be synchronized: once a generator is discovered in Part I, it can be immediately scheduled for processing in Part II. For more details on the plane sweep construction and the synchronization of Parts I and II that is important in maintaining the locality of the computation see [34]. We thus conclude:

Theorem 7 *Assuming that $G(N)$ is available for all nets under consideration and given a small constant k, the approximate opens Voronoi diagram $\mathcal{V}(G'(A))$, $G'(A) = \cup_{1\le i\le k} G_i(A)$ can be computed in time $O(n \log n)$ plus the time needed to answer connectivity queries. The latter can be done in time $O(n \log^2 n)$.*

The original implementation of this method, whose experimental results are reported in [34], used a slightly different approach in order to guarantee accuracy while the locality property was preserved. Namely, the iterative process of Sect. 3.5 was applied to each biconnected component independently. The advantages of considering each biconnected component independently were locality as well as the ability to run the process on each individual component to completion and guarantee the accuracy. The disadvantage was that generators produced in this manner, $G''(A)$, did not need all be critical. Including noncritical generators in the set $G''(A)$ complicates the algorithm of Part II because it amounts to computing the *Hausdorff Voronoi diagram* of corresponding geometric minimal cuts on layer A. Note that critical generators can be treated as simple additively weighted segments, having special weights, such that Voronoi regions remain connected and the entire generator is always enclosed in its Voronoi cell. This property considerably simplifies the construction of their (weighted) Voronoi diagram, which is, the Hausdorff Voronoi

diagram, which remains similar to the construction of ordinary Voronoi diagrams of line segments. Once non-critical generators are present, however, this property no longer holds and the algorithms is equivalent to constructing the Hausdorff Voronoi diagram of the corresponding cuts as presented in Sects. 2.3 and 4.

For information on critical area computation once the solution to the MGMC problem is available, see, for example, [32, 34, 36].

4 The L_∞ Hausdorff Voronoi Diagram

In this section, we review structural properties of the L_∞ Hausdorff Voronoi diagram of clusters of points, or equivalently, the L_∞ Hausdorff Voronoi diagram of rectangles, presented in [39]. The L_∞ Hausdorff Voronoi diagram of rectangles provides a solution to the MGMC problem after geometric minimal cuts have been identified. Results in this section are reproduced from [39].

Given a set S of point clusters in the plane, the *Hausdorff Voronoi diagram* of S, denoted *HVD(S)*, is a subdivision of the plane into regions, such that the *Hausdorff Voronoi region* of a cluster P, denoted *hreg(P)*, is the locus of points *closer* to P than to any other cluster in S, where distance between a point t and a cluster P is measured as the *farthest distance* between t and any point in P, $d_f(t, P) = \max\{d(t, p), p \in P\}$, $hreg(P) = \{x \mid d_f(x, P) < d_f(x, Q), \forall Q \in S\}$. It is subdivided into finer regions by the *farthest Voronoi diagram* of P, *FVD(P)*. The farthest distance $d_f(t, P)$ is equivalent to the *Hausdorff distance*[6] $d_h(t, P)$ between t and P. In the L_∞ metric, $d_f(t, P)$ (equiv. $d_h(t, P)$) is equivalent to $d_f(t, P')$ (equiv. $d_h(t, P')$), where P' is the minimum enclosing axis-aligned rectangle of P. Thus, the L_∞ Hausdorff Voronoi diagram of S is equivalent to the L_∞ Hausdorff Voronoi diagram of the set S' of the minimum enclosing rectangles of all clusters in S. In the following, the terms cluster and rectangle are used interchangeably.

In this section, we review the tight bound on the structural complexity of the L_∞ Hausdorff Voronoi diagram given in [39]. It is shown that the structural complexity of the L_∞ Hausdorff Voronoi diagram is $\Theta(n + m)$, where n is the number of input clusters (equiv. rectangles) and m is the number of *essential pairs* of *crossing* clusters (see Definition 11). We also review a simple plane sweep construction in two dimensions given initially in [32] and improved in [39]. The improved algorithm consists of an $O((n+M)\log n)$-time preprocessing step, based on *point dominance* in $\mathbb{R}^3$, followed by the main plain sweep algorithm that runs in $O((n + M)\log n)$-time and $O(n + M)$-space, where M reflects special crossings that are *potentially essential* (see Definition 12); m, M are $O(n^2)$, $m \leq M$, but $m = M$, in the worst case. In practice, typically, $m, M << n^2$. For non-crossing rectangles the algorithm simplifies to optimal $O(n \log n)$-time and $O(n)$-space.

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

An $O(n \log n)$-expected time algorithm can be derived in the case of non-crossing rectangles using the randomized incremental construction for abstract Voronoi diagrams [24] (see [1]). For arbitrary rectangles, however, the L_∞ Hausdorff Voronoi diagram does not fall under the umbrella of *abstract Voronoi diagrams* [23] (see, e.g., [38]). This algorithm is efficient for a relatively small number of crossing rectangles as motivated by the critical area application. For a large number of crossings, the $O(n^2)$-time approach of [11] can be preferable. Section 2.3 presented an output-sensitive version of the plane sweep construction in three dimensions of increased space complexity.

4.1 Definitions and Structural Complexity

Let S be a set of n rectangles, or a set of n point clusters in the plane, where each cluster has been substituted by its minimum enclosing axis aligned rectangle. A pair of rectangles (P, Q) is called *crossing* if P and Q intersect in the shape of a cross. Given a crossing pair (P, Q), P is assumed to be at least as long as Q. For a rectangle P, let P^n, P^s, P^e, and P^w denote the north, south, east, and west edges of P, respectively. P is called horizontal (resp. vertical) if P^n is longer (resp. shorter) than P^e. The axis-parallel line through edge $P^i, i = n, s, e, w$, is denoted as $l(P^i)$. The term P^i is also used to denote the main coordinate of edge P^i. The *core segment* of P is the locus of centers of all minimum enclosing squares of P, and it is given by the axis-parallel line segment of the L_∞ *farthest Voronoi diagram* of P. It can be viewed as an ordinary line segment s additively weighted with $w(s) = d_f(s, P)$. In Fig. 26, *FVD(P)* is illustrated in dashed lines and the core segment is indicated by s. The L_∞ Hausdorff Voronoi diagram of S is equivalent to the (weighted) Voronoi diagram of the set of *core segments* of all clusters in S (see [32]).

The Hausdorff bisector between two clusters P, Q is $b_h(P, Q) = \{y \mid d_f(y, P) = d_f(y, Q)\}$. As shown in [38], $b_h(P, Q)$ is a subgraph of $FVD(P \cup Q)$. For a rectangle Q strictly enclosed in the interior of a minimum enclosing square of P, $b_h(P, Q)$ consists of either one (if P and Q are non-crossing) or two (if P and Q are crossing) chains, each one forming a V-shape out of the ± 1-slope rays of $FVD(P \cup Q)$; the apex of each chain is called a *V-vertex*. A V-vertex v is incident to the core segment of P and its 90°-angle faces the portion of the plane closer to P. It is characterized as *up*, *down, right*, or *left*, depending on whether its 90°-angle is facing north, south, east, or west, respectively. In addition, it is characterized as *crossing*, if Q is crossing P, and *non-crossing*, otherwise. The minimum enclosing square of P centered at V-vertex v is also enclosing Q and it is denoted as $square(P, v)$. It is also denoted as $square(P, Q^i)$, where Q^i is the non-crossing edge of Q that delimits one of its edges. Figure 26 illustrates $b_h(P, Q)$ consisting of two crossing V-vertices, one right and one left; $square(P, Q^w)$ is illustrated dashed. $square(P, Q^i)$ is referred to as an *extremal minimum enclosing square* of P and Q. The V-vertices of *HVD(S)* are referred to as *Voronoi V-vertices*.

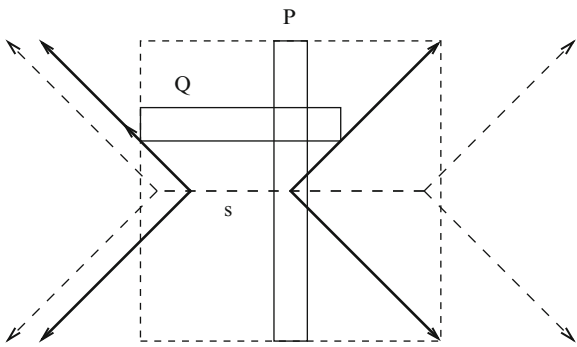

Fig. 26 The L_∞ Hausdorff bisector of crossing rectangles

Definition 11 A pair of crossing rectangles (P, Q) is called *essential* if there is an extremal minimum enclosing square of P and Q, $square(P, Q^i)$, that is empty of any other rectangle.

The following lemma is easy to see.

Lemma 19 *A pair of crossing rectangles (P, Q) induces a Voronoi V-vertex v in HVD(S) if and only if (P, Q) is an essential crossing. Assuming that P is a vertical rectangle, v is a right (resp. left) V-vertex if and only if $square(P, Q_i^w)$ (resp. $square(P, Q_i^e)$) is empty of other rectangles. Similarly for a horizontal rectangle.*

Combining Lemma 19 with the structural complexity results of [38], the following bound can be derived (see [39] for details).

Theorem 8 *The structural complexity of the L_∞ Hausdorff Voronoi diagram of a set S of n point clusters, equivalently n rectangles, is $\Theta(n + m)$, where m is the total number of essential crossings.*

Definition 12 A collection of crossings for a vertical rectangle P, (P, Q_i), $i = 1, \ldots, k$, is called a *staircase*, if $Q_i^w < Q_{i+1}^w$ and $Q_i^e < Q_{i+1}^e$, $i = 1, \ldots, k$. If in addition, $square(P, Q_i^w)$ is empty of $Q_j \neq Q_i$, the staircase and its crossings are called *potentially essential*. The maximum size of a potentially essential staircase for P is the *number of potentially essential crossings* for P. Let M denote the total number of potentially essential crossings for all vertical rectangles in S, plus the number of essential crossings for all horizontal rectangles in S.

Figure 27 shows a potentially essential staircase for a vertical rectangle P. In the absence of additional rectangles, all crossings are essential, that is, they all induce Voronoi V-vertices in the Hausdorff Voronoi diagram. The shaded regions in Fig. 27 belong to $hreg(P)$.

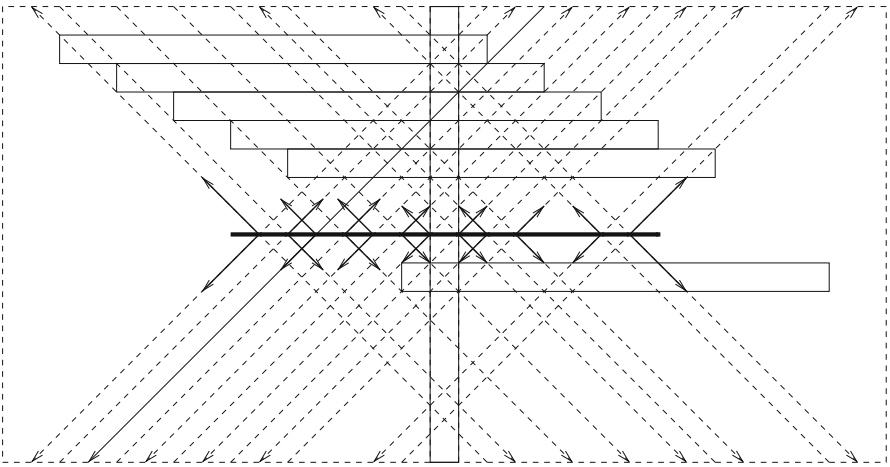

Fig. 27 Collection of essential crossings. Shaded regions belong to P

4.2 A Refined Plane Sweep Construction

In this section we present the plane sweep construction of [32, 39]. It is based on the standard plane sweep paradigm for Voronoi diagrams [8, 15], its adaptation for line segments in L_∞ [36], and the generalization to handle the special features of Hausdorff Voronoi diagrams introduced in [32, 33]. In the Hausdorff Voronoi diagram, sites need not be enclosed in their Voronoi regions and Voronoi regions may be disconnected, which are features not addressed by the standard plane sweep paradigm for Voronoi diagrams.

Assume a vertical sweep-line l_t sweeping the entire plane from left to right. At any instant t of the sweeping process, $HVD(S_t \cup l_t)$ is computed, for $S_t = \{P \in S \mid l(P^e) < t\}$. The boundary of the Voronoi region of l_t is the *wavefront* at time t. Voronoi edges and core segments incident to the wavefront are called *spike bisectors* and *spike core segments*, respectively. The combinatorial structure of the wavefront changes at specific *events* organized in a priority queue. We have four types of *site events*: *start-vertical-rectangle*, *end-vertical-rectangle*, *V-vertex* events (for brevity *V events*), and *horizontal-rectangle* events. Site events are partially similar to those for ordinary line segments [36] enhanced with additional functions to handle V-vertices and disconnected Voronoi regions. *Spike events* are caused by the intersection of incident spike bisectors, and they remain the same as in the ordinary plane sweep paradigm.

The *wave-curve* of a rectangle R is the bisector between R and the sweep line l_t, at time t, $b(R, l_t) = \{y \mid d_f(y, R) = d(y, l_t)\}$, where $d(y, l_t)$ is the ordinary distance between y and l_t. In L_∞, it consists of two or three *waves*: a ray of slope -1, corresponding to $b(R^s, l_t)$, a ray of slope $+1$, corresponding to $b(R^n, l_t)$, and possibly a vertical line segment corresponding to $b(R^w, l_t)$, if appropriate. The wave-curve of R can be seen equivalently as the (weighted) bisector between the

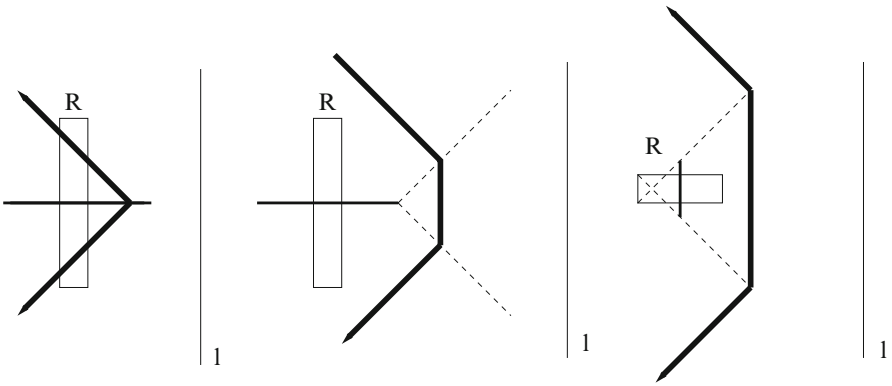

Fig. 28 The wave-curve of a rectangle R

core segment s of R and l_t, that is, $b(R, l_t) = \{y \mid d_w(y, s) = d(y, s) + w(s) = d(y, l_t)\}$. In Fig. 28, the wave-curve of several instances of rectangles is illustrated. The bold axis-aligned segment illustrates the core segment of R. The *wavefront* (equiv. the *beach line*) at time t is the lower envelope with respect to the sweep-line, of the *wave-curves* of all rectangles in S_t. Note that the term *beach line* of Sect. 2.3 is equivalent to the term *wavefront* of this section, which follows the terminology of [39]. In L_∞, the wavefront is monotone with respect to any line of slope ± 1. The wavefront is typically maintained as a height balanced binary tree, $\mathcal{T}$, ordered from bottom to top. Leaf nodes correspond to waves, while internal nodes correspond to spike bisectors and spike core segments revealing *breakpoints* between incident waves.

In [39], $\mathcal{T}$ gets augmented with nonstandard additional information in order to efficiently answer queries regarding V-vertices. This augmentation is essential for the time complexity bound of the main plane sweep algorithm. Each node x is augmented with a *w-max* value representing the rightmost west edge of all rectangles contributing a wave to the portion of the wavefront rooted at x, and two *x-min* values (*x-min-I* and *x-min-II*) that in combination represent the minimum x-coordinate of the portion of the wavefront rooted at x. In particular, for a leaf node representing a wave of rectangle R, the *w-max* value is R^w and both *x-min* values are $+\infty$. For an internal node x, *w-max* is the maximum between the *w-max* values of its children. If node x corresponds to a horizontal (resp. ± 1-slope) bisector, then *x-min-I* (resp. *x-min-II*) points to the breakpoint of minimum x-coordinate among its own breakpoint and the *x-min-I* (resp. *x-min-II*) values of its children; otherwise *x-min-I* (resp. *x-min-II*) points to the breakpoint of minimum x-coordinate among its children only. The minimum x-coordinate of the portion of the wavefront rooted at node x is *x-min* $= \min\{$*x-min-I*, *x-min-II*$\}$. The augmentation values *w-max, x-min-I, x-min-II* remain the same unless a combinatorial change in the wavefront (i.e., an event) takes place.

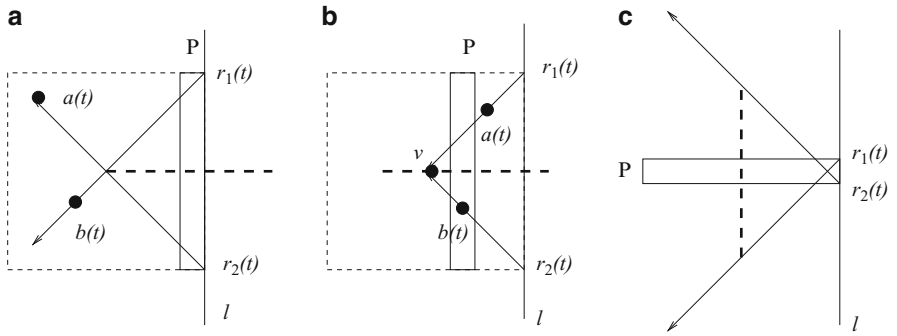

Fig. 29 (**a**) At $t = priority(P)$ the wavefront has not reached s. (**b**) An invalid event at time $t = priority(v)$. (**c**) A *horizontal* rectangle event

Any Voronoi point in *HVD(S)* enters the wavefront at the time of its *priority*. The *priority* of a point v is the rightmost x-coordinate of the smallest square centered at v that is entirely enclosing the rectangle P that induces v. The priority of a rectangle P is denoted as *priority(P)* and corresponds to the x-coordinate of P^e.

Let us now discuss the handling of various types of events of a rectangle P of core segment s (see Fig. 29). At time t, let $r_1(t)$ and $r_2(t)$ be the rays of slope $+1$ and -1, respectively, emanating from $l(P^n) \cap l_t$, and $l(P^s) \cap l_t$, respectively, extending towards the left of l_t. Let $a(t)$ and $b(t)$ be the intersection points of $r_1(t)$ and $r_2(t)$ with the wavefront, respectively, at time t. Since the wavefront is ± 1 monotone, $a(t)$ and $b(t)$ can be determined by binary search in $O(\log n)$ time. In case of a wave collinear with $r_1(t)$ or $r_2(t)$, the rightmost endpoint is assigned to $a(t), b(t)$, adopting the convention that an equidistant region is assigned to the rectangle preceding P. Because the wavefront is monotone with respect to any line of slope ± 1, in case of a vertical rectangle, the entire portion of the wavefront between $a(t)$ and $b(t)$ must be either to the left (Fig. 29a) or the right (Fig. 29b) of the intersection point of $r_1(t)$ and $r_2(t)$, and thus, it may intersect $r_1(t), r_2(t)$, or the core segment of s at most once. For a horizontal rectangle (Fig. 29c), the wavefront can intersect the vertical core segment of P a number of times.

Consider time $t = priority(P)$. There are three cases: (1) The wavefront has not reached core segment s yet (either at a start-vertical-rectangle or a horizontal-rectangle event); (2) The wavefront has already covered portion of s, where s is horizontal (start-vertical-rectangle event); and (3) The wavefront has already covered a portion of s but s is not horizontal (horizontal-rectangle event).

In case 1, the handling of the corresponding event (a start-vertical-rectangle or a horizontal-rectangle event) is similar to processing an ordinary line-segment event [32, 36]: The portion of the wavefront between $a(t)$ and $b(t)$ is finalized and gets substituted by the wave-curve of P. There is one new action to take: For any crossing V-vertex on the finalized portion of the wavefront, induced by a rectangle Q, generate a V event for the right V-vertex of $b_h(P, Q)$ and insert it to the event queue.

In case 2 (start-vertical rectangle event), a V event is generated to predict the first right V-vertex along s (if any). Given the wavefront, perform a binary search in the augmented wavefront to determine the wave between $a(t)$ and $b(t)$ with the rightmost w-max value as induced by a rectangle Q. If no other information is available, generate a V event for the right V-vertex of $b_h(P, Q)$. Note that at a start-rectangle event, Q must be non-crossing with P. Case 3 will be discussed later at a horizontal-rectangle event.

A V event v is processed similarly. If at time $t = priority(v)$ the event is *invalid*, i.e., the wavefront has already covered v, generate a new V event, given the wavefront, as described above. In particular, let Q be the rectangle inducing the wave with the rightmost w-max value among the waves between $a(t)$ and $b(t)$. If no additional information is available and assuming that Q is crossing P, generate a V event for the right V-vertex of $b_h(P, Q)$.

End-rectangle events are similar to right events of [32]. For a proof of correctness in handling vertical-rectangle events, see Lemma 2 in [39].

A horizontal-rectangle event is processed at time $t = priority(P)$. The problem is to identify the intersections (V-vertices) of the wavefront with the vertical core segment s (if any). Note that at time t, the wavefront may intersect s a number of times, each intersection corresponding to a V-vertex, where only the first and the last may be non-crossing V-vertices. If there are no intersections because the wavefront has not reached any portion of s yet, then we have Case 1 of $t = priority(P)$ discussed earlier. Otherwise, this is Case 3. Any portion of the wavefront to the left of s is finalized and gets substituted by the wave-curve of P as fragmented by the V-vertices and their incident spike bisectors.

To identify V-vertices efficiently, the x-min value of the augmentation is used. Let r be the breakpoint of minimum *x-min* value between $a(t)$ and $b(t)$. If r is to the right of s, then s must be entirely covered by the wavefront and there can be no intersections, that is, $hreg(P) = \emptyset$. Otherwise, trace the wavefront sequentially, starting at r, until the first intersections above and below r are determined. The intersection above (resp. below) corresponds to a *down* (resp. *up*) V-vertex v (resp. u). Repeat the process for the portions of the wavefront above v and below u until all intersections are determined. Any time the *x-min* value of a portion of the wavefront is to the right of s, this portion can be eliminated as it contains no intersections with s. The correctness of handling of a horizontal-rectangle event follows easily. A horizontal rectangle event covers also squares.

The time complexity of the plane sweep algorithm, as presented, is $O((n + m + E) \log n)$, where E is the number of invalid V events. There are two reasons for invalid V events: (1) Potentially essential staircases of vertical rectangles whose crossings are not all essential; and (2) Sequences of *strongly dominated* vertical rectangles, even in the case of non-crossing rectangles. Given a pair of vertical rectangles (P, Q), P is said to *dominate* Q if $Q^w < P^w$ and $Q^n < P^n$, $Q^s > P^s$. If, in addition, there is a minimum enclosing square of Q that is crossing P, P is said to *strongly dominate* Q.

To eliminate source-2 of invalid V-vertex events, a preprocessing step can be added to the algorithm that precomputes *dominating sequences* of vertical

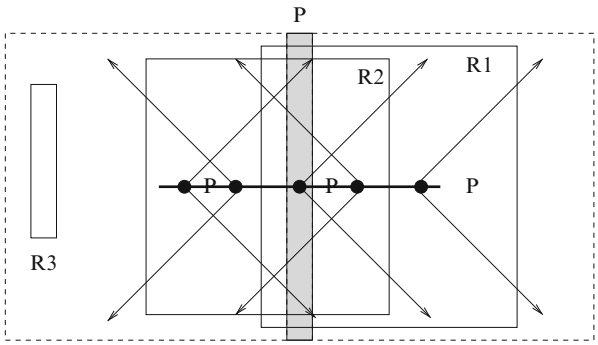

Fig. 30 The dominating sequence of rectangle P consisting of rectangles R_1, R_2, R_3

rectangles, which is described in [39]. Given the preprocessing information, the number of all V events generated is bounded by $O(n + M)$ as shown in [39].

Definition 13 The *dominating sequence* of a vertical rectangle P, denoted *ds(P)*, is a maximal collection of vertical rectangles $R_i, i = 1, \ldots, k$, such that every R_i is entirely enclosed in a minimum enclosing square of P, R_1 is the rectangle with the rightmost west edge among all rectangles dominated by P, and if R_i is crossing P, $i \geq 1$, then R_{i+1} is the vertical rectangle with the rightmost west edge dominated by $square(P, R^e_{i-1})$.

Every R_i in the *dominating sequence* of P, except possibly the last rectangle R_k, is crossing P. Rectangle R_1 in the dominating sequence of P is referred to as the *rightmost* rectangle dominated by P. In the case of non-crossing rectangles, the dominating sequence of P is R_1 (if not empty). Figure 30 shows the dominating sequence of a vertical rectangle P consisting of three rectangles R_1, R_2, R_3. Considering only those four rectangles, the region of P is alternating with regions of R_i on the core segment of P, as shown in Fig. 30. Lemma 3 in [39] shows that the dominating sequence of P (except the last rectangle R_k, if R_k is non-crossing with P) forms a maximal staircase of vertical rectangles crossing P that is potentially essential. No vertical rectangle, other than the dominating sequence of P, may induce a right Voronoi V-vertex on the core segment of P, even when only $ds(P)$ is considered. Given the dominating sequences of vertical rectangles, every time a V event is generated, source-2 of invalid V events can be completely eliminated. The exact algorithmic details are given in [39].

Using the information of dominating sequences of vertical rectangles, the total number of V events that may be produced throughout the algorithm is shown in [39] to be $O(n + M)$. The problem of computing the dominating sequence of every vertical rectangle can be transformed into a *point dominance* problem in $\mathbb{R}^3$ which can be solved by plane sweep in $O((n + M) \log n)$-time, as described in Sect. 4 of [39]. Combining with Lemma 3 of [39] we conclude.

Theorem 9 *HVD(S) can be computed by plane sweep in* $O((n+M)\log n)$*-time and* $O(n+M)$*-space. In the case of non-crossing rectangles, this results in optimal* $O(n \log n)$*-time and* $O(n)$*-space.*

For the plane-sweep algorithm to compute dominating sequences of vertical rectangles, see [39] (Sect. 4). It is based on a simple transformation to a variant of the standard *point dominance* problem in R^3 (see, e.g., [40]) and the use of an auxiliary standard *priority search tree* [28].

5 Conclusion

Given a graph $G = (V, E)$ with a subgraph H embedded in the plane, this chapter presented the problem of computing the *map of geometric minimal cuts* (MGMC) of H, as induced by axis-aligned rectangles in the embedding plane. It surveyed two different approaches for the solution of the MGMC problem [34, 47], that were based on a mix of geometric and graph-algorithm techniques. The chapter also surveyed results on the related Hausdorff Voronoi diagram, which provides a solution to the MGMC problem.

Recommended Reading

1. M. Abellanas, G. Hernandez, R. Klein, V. Neumann-Lara, J. Urrutia, A combinatorial property of convex sets. Discret. Comput. Geom. **17**, 307–318 (1997)
2. E. Aurenhammer, Voronoi diagrams – a survey of a fundamental data structure. ACM Comput. Surv. **23**(3), 345–405 (1991)
3. F. Aurenhammer, R. Klein, Voronoi diagrams, in *Handbook of Computational Geometry*, ed. by J. Sack, G. Urrutia (Elsevier, Amsterdam, 2000), pp. 201–290
4. F. Aurenhammer, R. Drysdale, H. Krasser, Farthest line segment Voronoi diagrams. Inf. Process. Lett. **100**, 220–225 (2006)
5. S.C. Braasch, J. Hibbeler, D. Maynard, M. Koshy, R. Ruehl, D. White, Model-based verification and analysis for 65/45nm physical design, CDNLive! September 2007
6. M. de Berg, O. Schwarzkopf, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, 2nd edn. (Springer, Berlin/New York, 2000)
7. J.P. de Gyvez, C. Di, IC defect sensitivity for footprint-type spot defects. IEEE Trans. Comput. Aided Des. **11**, 638–658 (1992)
8. F. Dehne, R. Klein, "The big sweep": on the power of the wavefront approach to Voronoi diagrams. Algorithmica **17**, 19–32 (1997)
9. F. Dehne, A. Maheshwari, R. Taylor, A coarse grained parallel algorithm for Hausdorff Voronoi diagrams, in *Proceedings of the 2006 International Conference on Parallel Processing*, Columbus (2006), pp. 497–504
10. J.R. Driscoll, N. Sarnak, D. Sleator, R. Tarjan, Making data structures persistent, in *STOC*, Berkeley, CA (1986), pp. 109–121
11. H. Edelsbrunner, L.J. Guibas, M. Sharir, The upper envelope of piecewise linear functions: algorithms and applications. Discret. Comput. Geom. **4**, 311–336 (1989)
12. D. Eppstein, Z. Galil, G.F. Italiano, A. Nissenzweig, Sparsification – a technique for speeding up dynamic graph algorithms. J. ACM **44**(5), 669–696 (1997)
13. G.N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985)

14. M. Fredman, M. Henzinger, Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998)
15. S. Fortune, A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174 (1987)
16. P. Gupta, E. Papadopoulou, Yield analysis and optimization, in *The Handbook of Algorithms for VLSI Physical Design Automation*, ed. by C.J. Alpert, D.P. Mehta, S.S. Sapatnekar (Taylor & Francis/CRC, London, 2008)
17. M.R. Henzinger, M. Thorup, Sampling to provide or to bound: with applications to fully dynamic graph algorithms. Random Struct. Algorithm **11**, 369–379 (1997)
18. M.R. Henzinger, V. King, Randomized dynamic graph algorithms with polylogarithmic time per operation, in *Proceedings of the 27th STOC*, Las Vegas (1995), pp. 519–527
19. J. Holm, K. de Lichtenberg, M. Thorup, Polylogarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity, in *Proceedings of the 30th STOC*, Dallas, TX (1998), pp. 79–89
20. J. Holm, K. Lichtenberg, M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**(4), 723–760 (2001)
21. J. Hopcroft, R. Tarjan, Efficient algorithms for graph manipulation. Commun. ACM **16**(6), 372–378 (1973)
22. A.B. Kahng, B. Liu, I.I. Mandoiu, Non-tree routing for reliability and yield improvement. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **23**(1), 148–156 (2004)
23. R. Klein, *Concrete and Abstract Voronoi Diagrams*. Lecture Notes in Computer Science, vol. 400 (Springer, Berlin/New York, 1989)
24. R. Klein, K. Mehlhorn, S. Meiser, Randomized incremental construction of abstract Voronoi diagram. Comput. Geom. **3**, 157–184 (1993)
25. R. Klein, E. Langetepe, Z. Nilforoushan, Abstract Voronoi diagrams revisited. Comput. Geom. **42**(9), 885–902 (2009)
26. D.T. Lee, On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput. **C-31**, 478–487 (1982)
27. W. Maly, J. Deszczka, Yield estimation model for VLSI artwork evaluation. Electron Lett. **19**(6), 226–227 (1983)
28. E.M. McCreight, Priority search trees. SIAM J. Comput. **14**, 257–276 (1985)
29. K. Mehlhorn, S. N$\ddot{a}$her, Dynamic fractional cascading. Algorithmica **5**, 215–241 (1990)
30. P.B. Miltersen, S. Subramanian, J.S. Vitter, R. Tamassia, Complexity models for incremental computation. Theor. Comput. Sci. **130**(1), 203–236 (1994)
31. Y. Nakamura, S. ABE, Y. Ohsawa, M. Sakauchi, MD-tree: a balanced hierarchical data structure for multi-dimensional data with highly efficient dynamic characteristics. IEEE Trans. Knowl. Data Eng. **5**(4), 682–694 (1993)
32. E. Papadopoulou, Critical area computation for missing material defects in VLSI circuits. IEEE Trans. Comput. Aided Des. **20**(5), 583–597 (2001)
33. E. Papadopoulou, The Hausdorff Voronoi diagram of point clusters in the plane. Algorithmica **40**, 63–82 (2004)
34. E. Papadopoulou, Net-aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. IEEE Trans. Comput. Aided Des. **30**(5), 704–716 (2011). Preliminary version in *ISAAC'07*. Lecture Notes in Computer Science, vol. 4835 (2007), pp. 716–727
35. E. Papadopoulou, D.T. Lee, Critical area computation via Voronoi diagrams. IEEE Trans. Comput. Aided Des. **18**(4), 463–474 (1999)
36. E. Papadopoulou, D.T. Lee, The L_∞ Voronoi diagram of segments and VLSI applications. Int. J. Comput. Geom. Appl. **11**, 503–528 (2001)
37. E. Papadopoulou, D.T. Lee, The min-max Voronoi diagram of polygonal objects and applications in VLSI manufacturing, in *Proceedings of the 13th International Symposium on Algorithms and Computation*, Vancouver, BC. Lecture Notes in Computer Science, vol. 2518, (2002), pp. 511–522
38. E. Papadopoulou, D.T. Lee, The Hausdorff Voronoi diagram of polygonal objects: a divide and conquer approach. Int. J. Comput. Geom. Appl. **14**(6), 421–452 (2004)

39. E. Papadopoulou, J. Xu, The L_∞ Hausdorff Voronoi diagram revisited, in *IEEE-CS Proceedings, ISVD 2011, International Symposium on Voronoi Diagrams in Science and Engineering*, Qingdao (2011)
40. F.P. Preparata, M.I. Shamos, *Computational Geometry: An Introduction* (Springer, New York, 1985)
41. D. Sleator, R. Tarjan, A data structure for dynamic trees. J. Comput. Syst. Sce. **26**(3), 362–391 (1983)
42. R. Tarjan, Depth-first search and linear graph algorithms. SIAM J. Comput. **1**, 146–160 (1972)
43. R.E. Tarjan, Efficiency of a good but not linear set union algorithms. J. ACM **22**, 215–225 (1975)
44. M. Thorup, Decremental dynamic connectivity, in *Proceedings of the 8th SODA*, New Orleans (1997), pp. 305–313
45. M. Thorup, Near-optimal fully-dynamic graph connectivity, in *STOC*, Portland (2000), pp. 343–350
46. "Voronoi CAA: Voronoi Critical Area Analysis", IBM CAD Tool, Department of Electronic Design Automation, IBM Microelectronics, Burlington, VT. Initial patents: US6178539, US6317859. Distributed by Cadence
47. J. Xu, L. Xu, E. Papadopoulou, Computing the map of geometric minimal cuts, in *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, Hawaii, USA. Lecture Notes in Computer Science, vol. 5878, 16–18 Dec 2009, pp. 244–254