Fast Yield-Driven Fracture for Variable Shaped-Beam Mask Writing

Andrew B. Kahng[†], Xu Xu[‡] and Alex Zelikovsky[¶]
[†] CSE and ECE Departments, UCSD, La Jolla, CA 92093
[‡] CSE Department, UCSD, La Jolla, CA 92093
[¶] CS Department, Georgia State University, GA 30303
{abk,xxu}@cs.ucsd.edu, alexz@cs.gsu.edu

ABSTRACT

Increasing transistor densities, smaller feature sizes, and the aggressive use of RET techniques with each successive process generation have collectively presented new challenges for current fracture tools, which are at the heart of layout data preparation. One main challenge is to reduce the number of small dimension trapezoids (*slivers*) to improve mask yield since the sliver count reflects the risk of mask critical-dimension errors.

Some commercial tools are available for handling the sliver minimization problem in fracture, such as CATS from Synopsys and Fracturem from Mentor Graphics. However, the number of slivers in the existing fracture solutions can be significantly reduced. The integer linear programming (ILP) method has been previously applied to find the optimal fracture[1] but has not explored potential benefits from additional ray-segments. Unfortunately, the ILP becomes prohibitively slow for polygons with the large number of vertices and heuristic partitioning of large polygons may severely degrade the solution quality.

In this paper, we propose a new ray-segment selection heuristic which can find a near-optimal fracture solution in practical time while being flexible enough to take into account all specified requirements. We fist divide the rectilinear region with all rays from the concave points and formulate the fracture problem as a sequential ray-segment selection problem. Each ray segment is assigned a weight based on its probability to form a sliver. All ray segments to be selected are placed in a candidate pool. An iterative "gain" based process is used for fast and efficient selecting ray segments from the candidate pool and dynamic update of ray segments and their gains. Further reduction of the number of slivers is achieved by auxiliary ray-segments. The resulted runtime overhead is reduced by a rule-based auxiliary ray-segments addition method which achieves a tradeoff between the sliver number reduction and runtime overhead. Compared with state-of-art sliver-driven fracturing tools, the proposed method reduces the number of slivers in the fractures of two industry testcases by 76.7% and 58.6%, respectively, without inflating the runtime and shot count. Similarly, compared with the previous ILP based fracture methods, the new method reduces the number of slivers by 56.1% and 2.2%, respectively, with more than 60X speedup and insignificant shot count overhead. The reduction in the sliver number is primarily due to the introduction of additional ray-segments. The proposed method can also solve the reverse-tone fracture problem in practical time for large industry testcases.

Keywords: Fracture, Variable Shaped-Beam Mask Writing, Mask data preparation, Yield-driven

1. INTRODUCTION

The onset of the 90nm and 65nm technology nodes is accompanied by a sharp increase in mask manufacturing cost, which becomes prohibitive for low-volume designs. The mask cost increase is directly in line with increased write time and data volume, which is caused by highly complex *reticle enhancement technology* (RET) used to manufacture deeply sub-wavelength features. To decrease the turnaround time of mask manufacturing and to improve mask quality for highly complex layouts, variable shaped beam (VSB) e-beam mask writing becomes more attractive than increasingly expensive raster mask writing. Today's VSB mask writing machines offer a plethora of rapidly advancing technologies, with beam currents reaching 50kV, support for slanted apertures, and a host of data formats for pattern generation from "fractured" layout information. This paper addresses the problem of optimizing VSB mask writing to take into account the constraints imposed by mask writing equipment as well as manufacturability of optically corrected layouts.

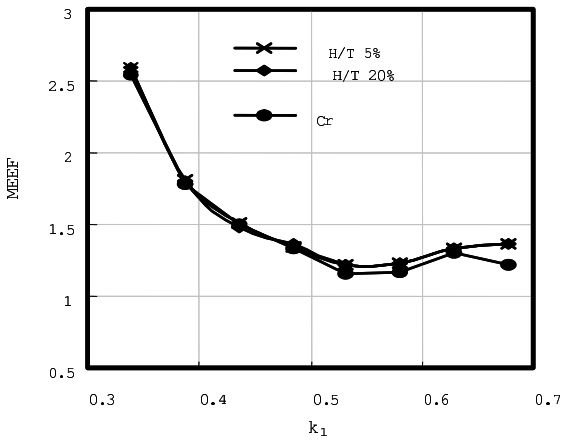

Figure 1. Relationship between MEEF and k_1 for different mask types, where k_1 is proportional to feature size.

1.1. Definitions and Problem Statement

In the following, we refer to dimensions of various parameters on the *mask*. Corresponding dimensions on the wafer can be obtained by scaling down these values by the *stepper reduction ratio* = the ratio of image size on the reticle to that on wafer (in each of the x and y dimensions), which is usually between 4 and 5. Exposure data for VSB writing is described as a set of single-exposure units, commonly referred to as *shots*. There are several limitations on the shape and size of a single shot:

(a) each shot should be either a rectangle or, more generally, an axis-parallel trapezoid;

(b) the side size of each shot cannot exceed a certain maximum threshold value M;

(c) shots cannot overlap, i.e., each feature is partitioned into disjoint shots; and

(d) the minimum width of each shot should be above a certain minimum threshold value ε.

The first two constraints are hard since the VSB writing technology cannot reproduce arbitrary shapes, and since exposure quality can be guaranteed only up to a certain extent. Currently, the value of M is between $2\mu m$ and $3\mu m$ (e.g., $2.55\mu m$ for a recent Toshiba VSB writing tool). This corresponds to an image size of $0.5\mu m$ to $0.75\mu m$ on the wafer with a $4\times$ reduction stepper. The third constraint is also hard since any overlap between shots would be comparatively overexposed with respect to non-overlapped shots. The fourth constraint is "soft" – narrow trapezoids having width below the critical value ε (which is typically around 100nm on the mask scale), henceforth referred to as *slivers*, can still be reproduced. However, as shown in Figure 1, small feature size proportional to k_1 * will lead to an increase in the mask error enhancement factor (MEEF), which is formally defined as the ratio of changes in the pattern printed on the wafer to the corresponding changes in the pattern written on the reticle[9,10,11] . An increase of MEEF will cause larger CD variation and more manufacturing defects, likely reducing mask and / or wafer manufacturing yield. In general, either the number of slivers or the total length of slivers should be minimized.† Figure 2 shows one example of the shots obtained from the fracturing of post-RET layout data.[16]

The design-through-mask data flow is shown in Figure 3.[16] In this process, the number of shots is roughly proportional to the mask writing time, which in turn affects the mask cost. Therefore, the traditional optimization objective for fracture, i.e., partitioning of polygons into shots, is to minimize the number of shots. Some slivers can be avoided, but a significantly reduced number or total length of slivers may be achievable only at the cost of an increased number of shots.

Fracture Problem. Given a simple polygon P with axis-parallel and 45-degree slant edges, along with specified critical dimensions, partition P into axis-parallel trapezoidal shots subject to constraints (a),(b),(c) minimizing

$$\#(shots) + W_C \#(slivers) \tag{1}$$

*$W_{min} = \frac{k_1 \lambda}{NA}$, where W_{min} is the minimum feature size, λ is the exposure wavelength and NA is numerical aperture.

†Nakao et al.[2] have suggested that slivers be counted only if they share a boundary with the polygon to be fractured.

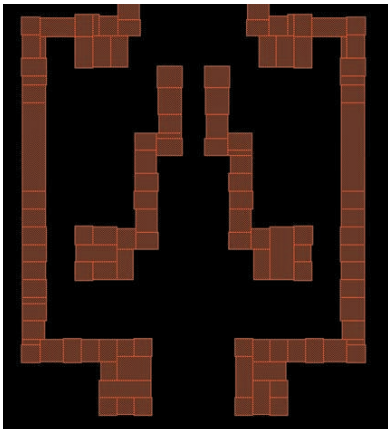

Figure 2. An example of fractured polygons.

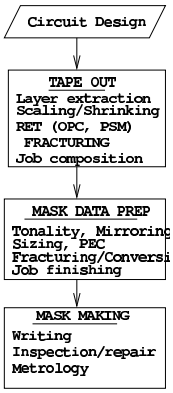

Figure 3. Mask data process flow.

where #(*shots*) and #(*slivers*) are the numbers of shots and slivers, respectively, and W_C is the scaling coefficient for the number of slivers.

1.2. Previous Work

Fracture of a polygon into basic shapes (rectangles, trapezoids) is a well-studied problem. The standard formulation is to minimize the number of shots subject to constraints (a) and (d) disregarding all other constraints specified above. Ohtzuki[6] has given an exact $O(n^{5/2})$ algorithm for polygon fracture into rectangles where n is the number of vertices of a polygon. The algorithm is based on finding a maximum independent set in a bipartite graph where vertices correspond to certain lines slicing the given polygon. Certain ideas of this algorithm are described in Section 2.1. Imai and Asano[5] have further sped up this algorithm to $O(n^{3/2}\log n)$ and also generalized it to the optimal partition into trapezoids.[4] Unfortunately, these theoretically nice algorithms are not flexible enough to take into account additional important constraints such as sliver minimization.

Nakao et al.[2] have developed a fairly complicated ad hoc heuristic based on the generalization of the same bipartite graph which takes in account all other constraints except the constraint (b). In fact, they have introduced a different objective – minimize the weighted length of slivers and slices cutting through critical features – while minimizing shot number over all obtained solutions that are (sub)optimal with respect to the new objective. Their heuristic is fast but does not guarantee optimum fracture and ignore the potential sliver saving with auxiliary rays. Recently, Kahng et al.[1] proposed an optimal integer linear algorithm based on a grid graph. The method did not explored potential benefits from additional ray-segments. Unfortunately, the ILP becomes prohibitively slow for polygons with the large number of vertices and heuristic partitioning of large polygons may severely degrade the solution quality.

A different technological aspect of the Fracture Problem has been addressed in a recent series of papers by Shulze et al.[3] and Cobb et al.[7,8] In the standard data preparation flow for VSB mask writing, a post-RET layout

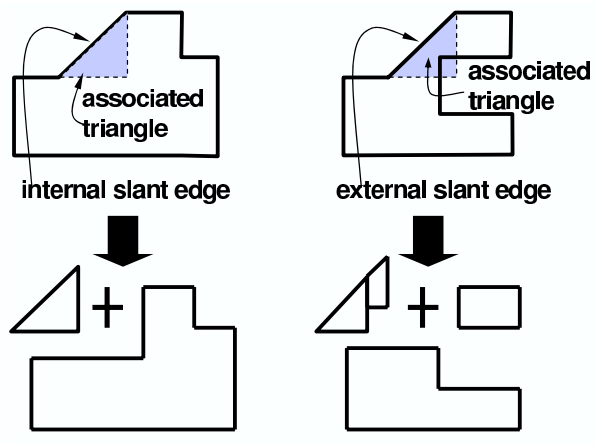

Figure 4. Internal and external slant edges.

in GDSII format is transferred into the MEBES mask writing format, which is then further transferred into VSB formats supported by various VSB mask writing machines. The drawback of this flow is that GDSII and VSB formats are hierarchical, while MEBES does not support hierarchy. The cited works suggest ways to avoid layout flattening (e.g., to exclude MEBES format from the flow), which result in drastic reduction of data volumes as well as processing times. The improvements that we develop in the present work are complementary to such methods.

1.3. Contributions

In this paper, we apply a fast heuristic to the fracture problem. Our contributions include:

- We propose a new ray-segment selection based fast heuristic which can find a near-optimal fracturing solution in practical time with good scalability while being flexible enough to take into account all specified requirements.
- We introduce auxiliary ray segments to further reduce the number of slivers. One rule is proposed for efficient auxiliary rays addition.

2. RAY-SEGMENT SELECTION BASED HEURISTICS

As in,[1] we generally consider a simple polygon P with axis-parallel and slant edges. With each slant edge we associate a triangle which internally intersects P, as shown in Figure 4. If this triangle is completely inside P, the slant edge is called *internal*; otherwise, it is called *external*. In order to simplify exposition of the Fracture Problem without compromising rigor we will further exclude from consideration slant edges since they can be replaced by two or more rectilinear edges as shown in Figure 4.

In order to partition P into trapezoids, which are convex quadrangles, it is necessary to start at least one *partitioning line*, further referred to as a *ray*, from each *rectilinear concave point* on the boundary of P, i.e., a point with internal angle 270° (see Figure 5). The rays should be axis-parallel since only axis-parallel trapezoids can be made in a single shot. There are two rays from each rectilinear concave point.

2.1. Polygon Partitioning with Coincident Rays

Since one ray should be sent from each concave point, the total number of rays is equal to the number of concave points N. Whenever a ray is sent it can stop as soon as it reaches the opposite side of P or another orthogonal ray which has been sent earlier. Two rays may coincide if they reach each other's origin, which is defined as a *coincident ray*. See Figure 6 for an example. If I denotes the number of pairs of coincident rays, then the total number of different rays that should be sent in order to partition P into trapezoids is $N-I$. Note that each time we send a ray, we increase the number of faces of the resulting planar graph by 1, and hence the total number of trapezoids in the polygon partitioning is $\#(trapezoids) = N-I+1$.

Thus, in order to minimize the number of trapezoids, one should maximize the set of coincident rays. Note that all such rays cannot intersect as well as cannot come from the same concave point. The last constraint can be expressed in a graph B with vertex set $R = RV \cup RH \cup S$, where RV (respectively, RH) is the set of coincident

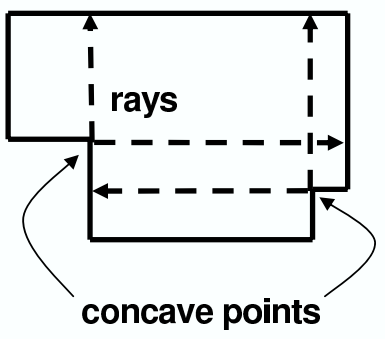

Figure 5. A polygon P with concave boundary points. The dashed vertical and horizontal rays are originated from concave points.

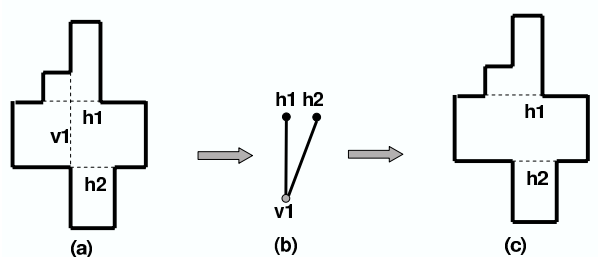

Figure 6. a) A polygon with five concave points and three rays between them. (b) The corresponding bipartite graph B, in which the vertices $\{h1, h2\}$ form the maximum independent set. (c) The corresponding partitioning into sub-polygons without coincide rays.

vertical (respectively, horizontal) rays and S is the set of rays sent from concave points which are simultaneously the endpoints of slant edges. Two vertices u and v of the graph B are adjacent if the corresponding rays are in conflict. This may happen in the following two cases (see Figure 6):

(a) $u \in RV$ and $v \in RH$ and the corresponding rays intersect;

(b) $u \in RV \cup RH$ and $v \in S$ and the corresponding rays are orthogonal and sent from the same concave point.

The maximum set of coincident rays that we seek corresponds to a maximum independent set in the graph B. In general, finding a maximum independent set is NP-hard, but in our case the graph B is bipartite since all edges are between vertices corresponding to orthogonal rays. According to König's theorem, finding the maximum independent set in a bipartite graph can be reduced to maximum matching and, therefore, can be done efficiently. Then the polygon will be divided by the selected set of coincident rays into some small polygons without coincident rays.

2.2. Ray-Segment Selection Formulation of the Fracture Problem

Our fracturing method is based on the following directed grid graph $G(V,E)$ (see Figure 7). For each concave point and the endpoint of slant edges, there are rays directed inside the polygon P and drawn to the opposite side of P. V are all intersection points of the rays and the concave vertices of the polygon P. E include all segments of the directed rays from one vertex to a neighboring vertex on the same ray or boundary segment. We enumerate all vertical rays with x-coordinates $X_i,\ i = 1,\ldots,hr$ and all horizontal rays with y-coordinates $Y_j,\ j = 1,\ldots,vr$. Then, the vertex of G that lies on the i-th vertical ray and j-th horizontal ray is denoted $v_{i,j}$. From each vertex $v_{i,j} \in V$, there are exactly one horizontal directed edge denoted as $e^h_{i,j}$ and one vertical directed edge denoted as $e^v_{i,j}$. In G, every ray becomes a linked list of ray segments. For a ray segment e, the next ray segment in the list is denoted as $Next(e)$, which is null if e is the end of the list. The number of ray segments will be $O(n^2)$, where n is the number of concave points. The fracture of a polygon P is denoted as F, which is a subset of $E - P$.

Fracture can be viewed as a process of "eliminating" all concave points by partitioning a polygon into rectangles since any rectilinear polygon without concave vertices is a rectangle.

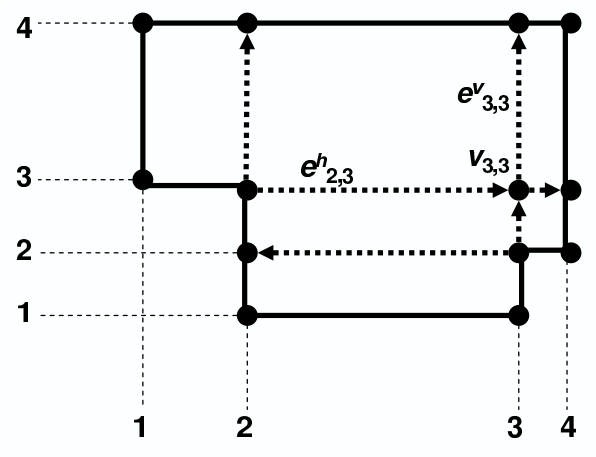

Figure 7. The grid graph G.

DEFINITION 2.1. *Two ray segments form a* **conflict pair** *if they are from the same vertex.* For a ray segment e, the ray segment form a conflict pair with e is denoted as $C(e)$.

THEOREM 2.2. *F is a fracture solution of the rectilinear polygon P with N concave points and the number of rectangles in the fracturing solution is $N+1$, if the following conditions are satisfied: (1) at least one of ray segments from each concave point of P is in F; (2) if $e \in F$ and $Next(e) \neq$ null, either $Next(e) \in F$ or $C(Next(e)) \in F$; and (3) at most one ray from each conflict pair is in F.*

Proof: We only need to prove that every vertex in G becomes convex with the edges of $P \bigcup F$. The convexity of any rectilinear concave vertex v of P is guaranteed with the first constraint since the internal angles of v are 90° and 180° with one ray segment from v. As shown in Figure 8, for any internal vertex v:

1. If there is no ray segment point to v (case (a)), it is a convex point;
2. If there is a ray segment $e \in F$ point to v and $Next(e) \in F$ (case (b)), it is a convex point since both internal angles are 180°;
3. If there is a ray segment $e \in F$ point to v and $C(Next(e)) \in F$ (case (c)), it is a convex point since the internal angles are 90°, 90° and 180°.

Therefore, every vertex in G is convex and hence all fractured polygons are rectangles.

Next, we want to prove that the number of rectangles is $N+1$. From the Eulerian formula relating the numbers of vertices, edges and faces of a planar graph:

$$\#(rectangles) = |E(H)| - |V(H)| + 1 \tag{2}$$

For the polygon P, the edge number is equal to the vertex number. Also, for each internal point v, there is exactly one ray segment from v according to the third constraint if one ray segment from v is in F. Therefore, the edge number is equal to the vertex number for each internal point. For every concave vertex v of P, there is a ray segment from v. There are N such edges and $\#(rectangles) = N+1$. □

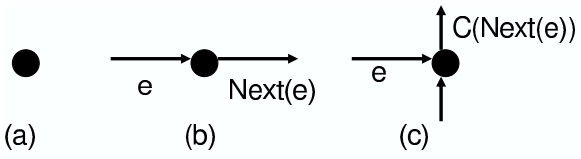

Figure 8. The cases of internal vertices v: a) no ray segments point to v; (b) a ray segment $e \in F$ point to v and $Next(e) \in F$; (c) a ray segment $e \in F$ point to v and $C(Next(e)) \in F$.

Input: Rectilinear polygon P and its corresponding grid graph $G(V,E)$
Output: $F \subset E$ to form a legal fracture solution with a minimize sliver number
1. $F \leftarrow \emptyset$; $S \leftarrow \{e \mid e$ from a concave point of $P\}$, calculate the gains of edges in S
2. **While** ($S \neq \emptyset$)
3. Select the edge e with the largest gain, $F \leftarrow F \cup \{e\}$
4. $S \leftarrow S - \{e, C(e)\}$
5. **If** ($Next(e) \neq$ null)
6. $S \leftarrow S \cup \{Next(e)\}$
7. Update gains of the edges in S

Figure 9. Gain-based ray segment selection algorithm.

2.3. Gain-Based Ray Segment Selection Heuristics

Based on Theorem 1, we propose the "Gain-Based Ray Segment Selection" or **GRSS** heuristics for the fracture problem.

DEFINITION 2.3. *A rectangle is a* **sliver** *if its minimum width* $< \varepsilon$*, where* ε *is a given threshold value.* As shown in,[1] slivers will decrease the mask manufacturing yield. The main objective of our method is to reduce the sliver number of the fracture solution.

DEFINITION 2.4. *For any edge* $e \in G$*, its* weight $W(e)$ *is equal to the number of parallel edges in* $F \cup P$ *which are within distance of* ε *and their projects in the orthogonal direction overlap with the projection of* e*. In other words, the weight of* e *is equal to the increased sliver number with the selection of* e*.*

DEFINITION 2.5. *For any edge* $e \in G$*, its* gain $G(e) = W(C(e)) - W(e)$*. In other words, the gain of* e *is equal to the number of "saved" slivers with the selection of* e*.*[‡]

Our proposed gain-based ray segment selection algorithm is shown in Figure 9. The algorithm starts with a candidate pool S which includes all ray segments from a concave point of P. We calculate the gains of all edges in S. Then in each iteration, one ray segment e is selected and its conflict pair ($\{e, C(e)\}$) is deleted from S (Line 3-4). If e is not the end ray segment in the ray list, the next ray segment of the list $Next(e)$ will be added to S. The gains of the edges in S are updated in Line 6. The iterative selection process continues until S is empty. The intuition behind this gain based algorithm is to greedily select the ray segment which leads to the largest save on the sliver number.

To speed up the gain update and ray segment selection process, we adopt a "linked bucket" structure to store the candidate pool S as shown in Figure 10. There are five linked lists or buckets, each store the edges in S with a gain between -2 and 2. The edge deletion, addition and gain update operations take constant runtime with this data structure.

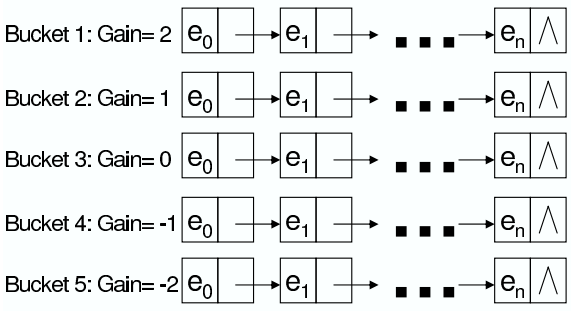

Figure 10. Bucket structure for candidate pool S.

3. AUXILIARY RAY SEGMENTS FOR SLIVER MINIMIZATION

DEFINITION 3.1. *An* auxiliary ray *is a ray whose starting point is not a concave vertex of* P*.*

[‡]Unlike the algorithm presented in[2] which use weights to choose rays, we adopt gains to advise ray segment selection.

From the Eulerian formula, the number of rectangles will be increased with auxiliary rays. However, the sliver number may be reduced with the additional rays at the expense of rectangle number increase. For the example shown in Figure 11, the sliver number is one in the fracture solution without auxiliary ray (case (a)); while the sliver number is zero with one auxiliary ray (case (b)). To the best of our knowledge, no previous methods are proposed to construct the auxiliary rays to minimize the sliver number. Since adding too many rays may significantly increase the complexity and runtime, it is crucial to limit the number of auxiliary rays being added.

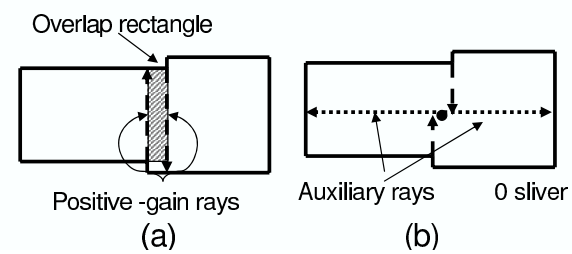

Figure 11. Fracture of a polygon: (a) without addition ray and (b) with auxiliary ray.

We propose the rule-based auxiliary rays addition method for efficient tradeoff between runtime and sliver number.

DEFINITION 3.2. *If a ray segment e is from a concave point of P and $G(e) > 0$, the ray containing e is a* positive-gain ray.[§]

DEFINITION 3.3. *For two parallel rays of different directions, let one ray from vertex v_0 located at (x_0, y_0) and the other ray from vertex v_1 located at (x_1, y_1), their* overlap rectangle *is a rectangle whose four corners are (x_0, y_0), (x_0, y_1), (x_1, y_1) and (x_1, y_0). Its width is the distance between the two rays and its length is the other dimension of the rectangle.*

Auxiliary Ray Addition Rule: Two auxiliary rays are added if two parallel positive-gain rays of opposite directions satisfy the following constraints: (1) the overlap rectangle located inside *P*; (2) the length of the overlap rectangle is $> 2\varepsilon$ and the width of the overlap rectangle is $< \varepsilon$; and (3) no existing rays intersect with the two rays and partition the overlap rectangle into two rectangles whose lengths are $> \varepsilon$. Two auxiliary rays from the center of the overlap rectangle are added to *G* as shown in Figure 11 (b) and the two ray segments from the center of the overlap rectangle are added to *S* in Line 1 of the algorithm shown in Figure 9.

4. EXPERIMENTAL RESULTS

We use two industry testcases to evaluate the performance of our fracture approach. Design A and Design B are from Photronics, Inc.[13] The basic properties of the two designs are listed in Table 1. For each testcase, the minimum number of fractured trapezoids is calculated using the method given in Section 6. We have implemented our algorithm in ANSI *C*. The previous ILP method specified in[1] uses ILP CPLEX 8.100 Mixed Integer Optimizer[12] to solve all Integer Linear Programming instances.

Testcase	# Polygons	Min # trapezoids	slants	# vertices
Design A	602	9613	21	24807
Design B	676	16273	17	38305

Table 1. Properties of testcases.

We have experimentally compared our fracture code against state-of-art commercial fracture tools. Two specific examples of leading commercial tools are Mentor Calibre v9.3_2.10 Fracturem[14] and Synopsys CATS v2501.[15] In our experiments, we set the maximum shot size as 2.55μm and the slivering size as 100nm, following parameters for a recent Toshiba VSB writing tool. The stepper reduction ratio is four. All tests are run on an Intel Xeon 2.4GHz CPU.[¶]

[§]There are two positive-gain rays in Figure 11 (a).

[¶]The results in Table 2 are anonymized to satisfy no-benchmarking restrictions in the vendor tool licenses. Per the license terms, we do not ascribe any specific results to any specific vendors or tools. However, we believe that our results demonstrate the magnitude of the solution quality gap in current tools.

	Design A				Design B			
Method	shots	slants	slivers	CPU(s)	shots	slants	slivers	CPU(s)
Tool A	10754	22	6111	0	17335	17	11572	0
Tool B	10455	23	4451	0	17130	17	10797	0
Tool C	9755	26	786	2	17195	21	6502	3
ILP	9750	22	417	134	17684	17	2750	222
GRSS+AR	9786	22	183	1	17656	17	2691	4

Table 2. Fracture results with sliver size $\varepsilon = 100nm$ and maximum shot size $M = 2.55\mu m$. ILP is the previous method specified in[1] and "GRSS+AR" is our proposed gain-based ray segment selection method with auxiliary rays.

The fracturing results in Table 2 show that our method can reduce the number of slivers by 83.7%, 81.1% and 60.5% compared to Tool A, Tool B, and Tool C respectively, while also reducing the number of shots by 5.5%, 0.6% and -2.5%. Compared to the previous Integer Linear Programming based method, our method reduces the sliver count by 56.1% and 2.2% for the two testcases, with more than 60X speedup and insignificant shot count overhead.

5. CONCLUSIONS

We have suggested a new gain-based ray segment selection method with auxiliary rays for the fracture problem in VSB mask writing. Our new approach substantially reduces sliver count in comparison to leading commercial mask data prep tools. Our method is much faster than the previous Integer Linear Programming based method (with more than 60X speedup). as well as reduces the sliver count by 56.1% and 2.2% for the two testcases, and insignificant shot count overhead.

From an IC design automation perspective, our work offers the possibility of directly considering yield loss mechanisms such as MEEF into existing layout and RET insertion flows. This would lead, for example, to fracturing-friendly "smart OPC". More generally, our results reveal significant headroom in existing tool solution quality; we believe that this can be exploited by future design-to-mask tools to reduce manufacturing variability and cost of IC designs. Directions for our ongoing and future work include:

- taking into account any unavoidable partitioning of slant edges and CDs, and, e.g., incorporating into the ILP objective the minimization of slant and CD slicing;
- fast heuristics for reverse-tone fracturing; and
- adjusting our approach to target non-rectilinear (e.g., X-style[17]) layouts with multiple slant edges.

REFERENCES

1. A. B. Kahng, X. Xu and A. Zelikovsky, "Yield- and Cost-Driven Fracturing for Variable Shaped-Beam Mask Writing", Proc. 24th BACUS Symposium on Photomask Technology and Management, 2004, pp. 360–371.
2. H. Nakao, M. Terai and K. Moriizumi, "A new figure fracturing algorithm for variable-shaped EB exposure-data generation," Electronics and Communication in Japan, Part 3, **83**, 2000, pp. 87–102.
3. S. Shulze, E. Sahouria and E. Miloslavsky, "High Performance Fracturing for Variable Shaped Beam Mask Writing Machines," Proc. of SPIE 5130, pp. 648–659.
4. T. Asano, T. Asano and H. Imai, "Partitioning a polygonal region into trapezoids," J. ACM, **33**, 1986, pp. 290–312.
5. H. Imai and T. Asano, "Efficient algorithms for geometric graph search problems," SIAM J. Comput., **15**, 1986, pp. 478–494.
6. T. Ohtsuki, "Minimum dissection of rectilinear regions," Proc. ISCS, 1982, pp. 1210–1213.
7. N. Cobb and E. Sahouria, "Hierarchical GDSII based fracturing and jobdeck system,", Proc. of SPIE 4562, pp. 734–762.
8. N. Cobb and W. Zhang, "High performance hierarchical fracturing," Proc. of SPIE 4754, pp. 91–96.
9. Y. Granik and N.B. Cobb, "MEEF as a Matrix", *Proc. of SPIE*, 2002, Vol. 4562, pp. 980-991.
10. W. Maurer, "Mask error enhancement factor", *Proc. of SPIE*, 2000, Vol. 3996, pp. 2-7.
11. F.M. Schellenberg and C.A. Mack, "MEEF in theory and practice", *Proc. of SPIE*, 1999, Vol. 3873, pp. 189-202.
12. CPLEX Mixed Integer Optimizer, ILOG. `http://www.cplex.com/`.
13. Photronics Inc. `http://www.photronics.com/`.
14. Calibre Fracturem, Mentor Graphics. http://www.mentor.com/calibre/datasheets/mdp/html/.
15. CATS, Synopsys. `http://www.synopsys.com/products/ntimrg/`.
16. Mask EDA workshop.`http://www.sematech.org/public/resources/mask/mask/INTRO.pdf`.
17. `http://www.xinitiative.org/`