Yield- and Cost-Driven Fracturing for Variable Shaped-Beam Mask Writing

Andrew B. Kahng,[a] Xu Xu[b] and Alex Zelikovsky[c]

[a]CSE and ECE Departments, University of California at San Diego
[b]CSE Department, University of California at San Diego
[c]CS Department, Georgia State University

ABSTRACT

Mask manufacturing for the approaching 90nm and 65nm nodes increasingly deploys variable shaped beam (VSB) mask writing machines. This has led to high interest in the *fracturing* methods which are at the heart of layout data preparation for VSB mask writing. In this paper, we set out the main requirements for fracturing and suggest a new solution approach based on integer linear programming (ILP). The main advantage of the new method is that the ILP finds optimal solutions while being flexible enough to take into account all specified requirements. We also suggest several decomposition (polygon partitioning) heuristics which speed up the ILP approach. Experimental comparisons with leading industry tools show significant improvement in quality, as well as acceptable scalability, of the proposed methods. In particular, our fracturing solutions reduce *shot count* (which reflects write time and mask cost) and dramatically reduce *sliver count* (which reflects the risk of mask critical-dimension errors). Our results reveal significant headroom that can be exploited by future design-to-mask tools to reduce the manufacturing variability and cost of IC designs.

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 subwavelength features. To decrease the turnaround time of mask manufacturing and to improve mask quality for highly complex layouts, the trend is away from increasingly expensive raster mask writing* and toward variable shaped beam (VSB) e-beam 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.

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;

The authors are partially supported by the MARCO Gigascale Systems Research Center and the NSF grant CCF 0429735.

*The mask writing grid size - roughly 1/50 of the minimum feature size - continues to decrease with technology scaling. Thus, raster writing methods take increasingly large amounts of time to write the mask, since the size of the die and mask remain roughly constant across technology nodes. With their fixed spot size, the raster tools do not achieve the shape resolution of the variable-shaped beam tools.

24th Annual BACUS Symposium on Photomask Technology,
edited by Wolfgang Staud, J. Tracy Weed, Proceedings of SPIE Vol. 5567
(SPIE, Bellingham, WA, 2004) 0277-786X/04/$15 · doi: 10.1117/12.568526

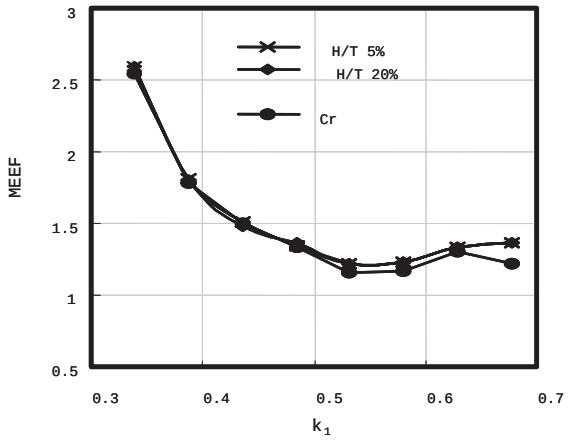

Figure 1. Relationship between MEEF and k_1 for different mask types, where k_1 is proportional to feature size.

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

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

The first two constraints are "hard" – the VSB writing technology cannot reproduce arbitrary shapes, and the 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 "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[8910] . 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.[‡]

Besides the above constraints on shot shape and dimension, there are also constraints on how a general layout geometry (polygon) can be represented as a *set* of shots:

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

(e) shots should not slice critical features, e.g., minimum-width poly gates, for which special fabrication accuracy is required; and

(f) slant edges should not be partitioned.

These constraints (d-f) are, in general, hard: any overlap between shots would be comparatively overexposed with respect to nonoverlapped shots; Critical features are already narrow and their width even without partitioning is difficult to control; and slant edges cannot be controlled with the same accuracy as axis-parallel edges. Figure 2 shows one example of the shots obtained from the fracturing of post-RET layout data.[15]

The design-through-mask data flow is shown in Figure 3.[15] In this process, the number of shots is roughly proportional to the mask writing time, which in turn affects the mask cost. Therefore, the main optimization

[†] $W_{min} = \frac{k_1 \lambda}{NA}$, where W_{min} is the minimum feature size, λ is the exposure wavelength and NA is numerical aperture.
[‡] Nakao et al.[1] have suggested to count slivers only if they share a boundary with the fractured polygon.

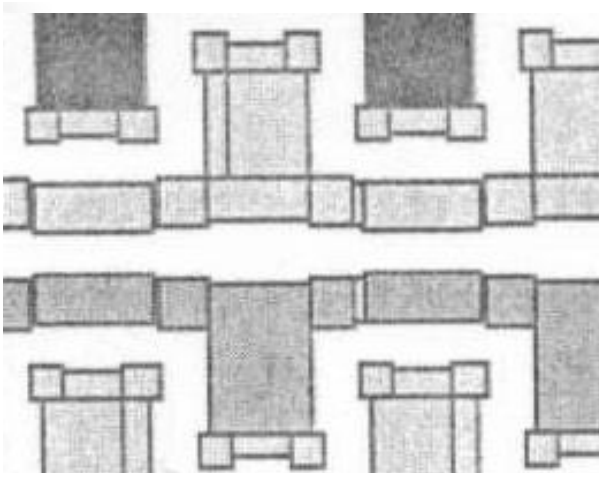

Figure 2. An example of fractured polygons.

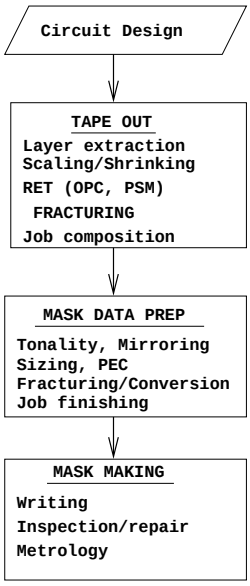

Figure 3. Mask data process flow.

objective for fracturing, 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. Therefore, we suggest an integrated objective function that seeks to minimize a linear combination of the number of shots and the number (or the total length) of slivers, with empirically chosen scaling coefficients.

Fracturing 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),(d),(e),(f) minimizing either

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

or

$$\#(shots) + W_L L(slivers) \tag{2}$$

where $\#(shots)$ and $\#(slivers)$ are, respectively, the numbers of shots and slivers, $L(slivers)$ is the total sliver length, and W_C and W_L are respectively scaling coefficients for the number of slivers and the total sliver length.

1.2. Previous Work

Fracturing 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[5] has given an exact $O(n^{5/2})$ algorithm for polygon fracturing 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 3. Imai and Asano[4] have further sped up this algorithm to $O(n^{3/2} \log n)$ and also generalized it to the optimal partition into trapezoids.[3] Unfortunately, these theoretically nice algorithms are not flexible enough to take into account additional important constraints.

Nakao et al.[1] 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 essentially disregards slant edges during fracturing (rather, slant edges are integrated after rectilinear fracturing); moreover, the method does not guarantee optimum fracturing and does not appear to allow any way of incorporating the maximum shot size constraint (b). Of course, a standard way to fracture polygons into bounded-size shots is to first partition the polygon into a small number of trapezoids, and then partition these large trapezoids into maximum-size shots. Such an approach is obviously suboptimal; our investigations show that it gives a significant increase in shot count, and we do not pursue it further.

A different technological aspect of the Fracturing Problem has been addressed in a recent series of papers by Shulze et al.[2] and Cobb et al.[6,7] In the standard data preparation flow for VSB mask writing, a post-RET layout 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 an integer linear programming (ILP) approach to the Fracturing Problem. Our contributions include:

- a more adequate formulation of the Fracturing Problem for VSB mask writing machines including maximum shot size constraint (b),
- new ILP formulation for the Fracturing Problem capturing all constraints (a-f),
- fast heuristics based on ILP formulation (see Section 3), and
- validation of the proposed heuristic with available industry tools.

In the next section we describe the ILP formulations for the Fracturing Problem. Section 3 is devoted to heuristic enhancements to speed up the ILP-based solution. Section 4 compares our results to those of leading commercial fracturing tools, and shows that the proposed approach offers significant improvement in quality (shot count and sliver count), as well as acceptable scalability. Finally, Section 5 gives conclusions and future research directions.

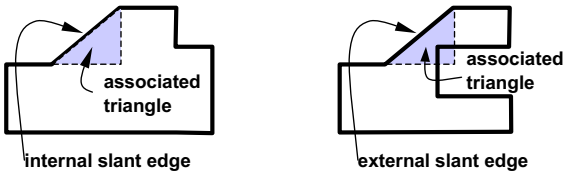

Figure 4. Internal and external slant edges.

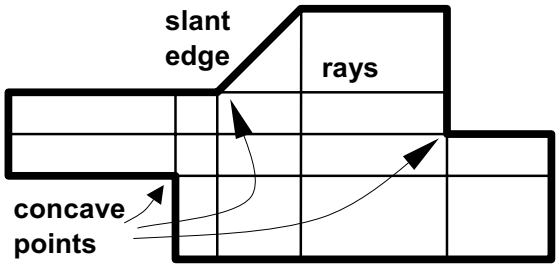

Figure 5. A polygon P with concave boundary points. The dashed vertical and horizontal rays are originated from concave points. The internal triangle associated with the slant edge is shaded.

2. ILP FORMULATION FOR THE FRACTURING PROBLEM

In this section, we describe an integer linear program formulation to optimally solve the Fracturing Problem. We generally consider a simple polygon P with axis-parallel and 45-degree-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 Fracturing Problem without compromising rigor we will further exclude from consideration external slant edges.[§] Note that partitioning of the internal slants can always be avoided.

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 *concave point* on the boundary of P, i.e., a point with internal angle greater than π (see Figure 5). The rays should be axis-parallel since only axis-parallel trapezoids can be made in a single shot. Besides rays from concave points, there are also rays from the (two) endpoints of slant edges, which form the associated internal triangles. At least one of the two rays from a slant edge should be used since the slant edge can serve only as a side edge of a trapezoid.

Our ILP formulation is based on the following grid graph G (see Figure 7). For each concave point, there are two rays directed inside the polygon P and drawn to the opposite side of P. For each end point of a slant edge, a single ray is drawn to the opposite side of P. The vertices of the graph G are all intersection points of the rays between themselves and with the boundary of the polygon P. The edges of G are all segments of the rays connecting neighboring vertices 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 $v_{i,j}$. Each horizontal edge of G between vertices $v_{i,j}$ and $v_{i+1,j}$ is denoted $e^h_{i,j}$, and each vertical edge between $v_{i,j}$ and $v_{i,j+1}$ is denoted $e^v_{i,j}$.

The number of variables and constraints in our ILP will be $O(n^2)$, where $n = vr + hr$ is the number of concave points plus the number of slant edges on the boundary of the given polygon P. As noted in[1] (see Figure 6), sometimes it may be necessary to use *auxiliary rays* initiated on the boundary at a convex point. These

[§]In fact, our ILP-based solution can take external slant edges into account and our implementation does not exclude them, but the exposition will be unnecessarily overcomplicated.

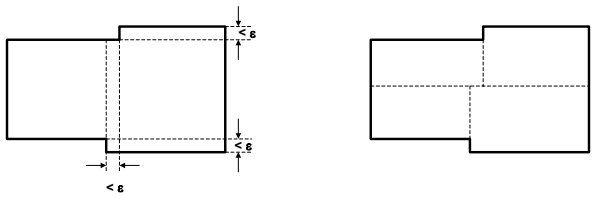

Figure 6. On the left, a polygon for which all possible partitions with minimum number of shots will have trapezoids of width less than ϵ. On the right, an additional horizontal line makes possible a partition without slivers, but having an extra shot.

auxiliary rays will induce additional vertices and edges in the grid graph G. However, in general there can be no more than $n-1$ auxiliary vertical lines and $n-1$ auxiliary horizontal lines: this is because two auxiliary rays originating between the same two rays initiated at concave points cannot be simultaneously useful.

We consider a fracturing of polygon P as a subgraph of the grid graph G. Each fracturing should choose certain edges of G as edges of its trapezoids. Let us introduce a Boolean variable $x^d(i,j)$, $d = v, h$, $i = 1, \ldots, hr$ and $j = 1, \ldots, vr$, which is set to 1 if the edge $e^d_{i,j}$ belongs to the chosen fracturing, and 0 otherwise. Note that variables corresponding to the boundary edges are always set to 1 since they belong to any fracturing.

In the rest of this section we show that the ILP objectives (1) and (2) can be computed based on Formulas (9), (11) and (13), and that the ILP constraints are expressed in (3), (4), (5), (6), (8), (10) and (12). We will first describe several types of constraints to ensure:

- convexity of rectilinear fracturing elements;
- convexity of trapezoids with slants;
- keeping intact slant edges and CDs;
- maximum shot size; and
- counting of shots and slivers.

2.1. Convexity constraints

The following constraints force any point $v_{i,j}$ (regardless of whether its position is on the boundary or inside P) to be convex with respect to fracturing edges. In other words, among the four edges incident to an internal point, there could be zero fracturing edges, two fracturing edges along the same ray, or three fracturing edges forming a "T"-shape.

$$\begin{aligned} x^h(i-1,j) + x^v(i,j-1) &\leq 2x^h(i,j) + 2x^v(i,j) \\ x^h(i,j) + x^v(i,j-1) &\leq 2x^h(i-1,j) + 2x^v(i,j) \\ x^h(i,j) + x^v(i,j) &\leq 2x^h(i-1,j) + 2x^v(i,j-1) \\ x^h(i-1,j) + x^v(i,j) &\leq 2x^h(i,j) + 2x^v(i,j-1) \end{aligned} \tag{3}$$

Indeed, let us consider the first constraint. If neither the bottom nor the left edge incident to the point is chosen, then the constraint trivially holds. If only one of the bottom and left edges is chosen, the constraint ensures that at least one more edge should be chosen, since no vertex can have degree 1 in the fracturing. If both bottom and left edge are chosen, then at least one more edge should also be chosen, i.e., the node $v_{i,j}$ cannot be concave since it should have degree at least 3. The next three constraints similarly ensure the same property for the bottom and right edges, top and right edges, and top and left edges.

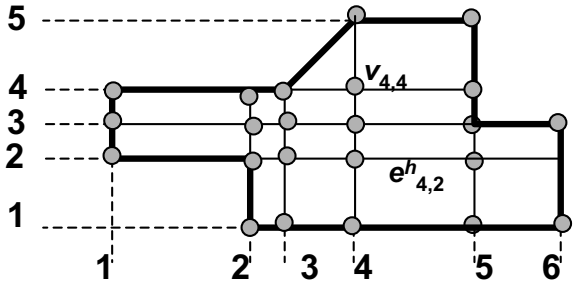

Figure 7. The grid graph G.

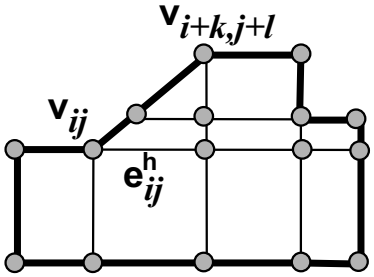

Figure 8. Treating a slant edge. Since the point point v_{ij} is concave, either the edge e^h_{ij} or the edge $e^v_{i,j-1}$ should be used in any fracturing. Since v_{ij} and $v_{i+k,j+l}$ are endpoints of a slant boundary edge, either the edge e^h_{ij} or $e^v_{i+k,j+l-1}$ should be in any fracturing.

2.2. Constraints for slant edges and critical features

The endpoints of slant edges are treated in a different manner, as follows. Let a slant boundary edge connect points $v_{i,j}$ and $v_{i+k,j+l}$ (see Figure 8). If $v_{i,j}$ is concave, then the following constraint ensures that at least one of the two non-boundary incident edges belongs to the fracturing solution.

$$x^v(i, j-1) + x^h(i, j) \geq 1 \tag{4}$$

A similar constraint is added if the other endpoint is also concave. To ensure that the slant edge is a part of an axis-parallel trapezoid, the following constraint is introduced:

$$x^v(i, j-1) + x^h(i+k, j+l-1) \geq 1 \tag{5}$$

Since the constraints (e-f) on fracturing are hard, we forbid slicing of slant edges and critical features by simply setting to 0 the variables corresponding to slicing edges.

2.3. Maximum shot size constraints

No previous fracturing method can directly minimize shot count in the presence of maximum shot size constraints. The ILP approach can smoothly incorporate such constraints, as follows. Let M be the upper bound on horizontal/vertical shot dimension. If the length of any edge in the graph G exceeds M, we add new rays partitioning such edges accordingly and modify the graph G. When combining a chain of edges into the boundary of a single shot we should ensure that it never exceeds M. Let X_i, $i = 1, \ldots, vr$, be the x-coordinates of the vertical rays. For each pair (i, i'), $i,\ i' \in \{1, \ldots, vr\}$, satisfying

$$X_{i'+1} - X_{i-1} \geq M$$

and

$$X_{i'} - X_{i-1} < M$$

we introduce the following constraint for each j, $j = 1, \ldots, hr$.

$$x^h(i,j) + x^h(i+1,j) + \ldots + x^h(i',j) \geq 1 \tag{6}$$

Similar constraints are also introduced for the vertical dimension.

2.4. Counting shots

The objective of the Fracturing Problem is to minimize the number of shots and the number of slivers, so it is necessary to accurately count these numbers in the ILP. The subgraph H of G corresponding to a fracturing is obviously planar. Each shot corresponds to a face of the graph H, and we can apply the Eulerian formula relating the numbers of vertices, edges and faces of a planar graph:

$$\#(shots) = |E(H)| - |V(H)| + 1 \tag{7}$$

The number of edges $|E(H)|$ is easily computed, since it is equal to the sum of all variables corresponding to edges of G. To find the number of vertices of H we must exclude from all vertices of G the vertices which become isolated in H. This is done by introducing a variable $y(i,j)$ for each vertex of G which is set to 0 if v_{ij} is isolated, and to 1 otherwise. Since the objective is to minimize the number of shots, (7) implies that it is sufficient to introduce an upper bound on $y(i,j)$:

$$y(i,j) \leq x^h(i-1,j) + x^v(i,j-1) + x^v(i,j) + x^v(i,j) \tag{8}$$

Finally, the number of shots is counted as:

$$\#(shots) = 1 + \sum_{d,i,j} x^d(i,j) - \sum_{i,j} y(i,j) \tag{9}$$

2.5. Counting and finding length of slivers

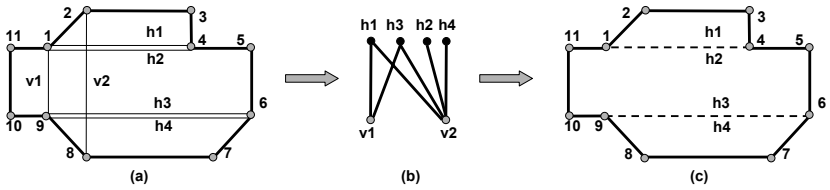

Figure 9. a) A polygon with three concave points 1, 4, and 9 and three slants $(1,2)$, $(6,7)$ and $(8,9)$. The pairs of rays reaching each other's origin are $v1 = ((1,9),(9,1))$, $h1 = ((1,4),(4,1))$, $v2 = ((2,8),(8,2))$, $h3 = ((6,9),(9,6))$. There are two cases of two rays being copies of each other: a ray $h1 = (1,4)$ sent horizontally from the concave point 1 and its copy $h2 = (1,4)$ sent from the endpoint 1 of the slant edge (1,2) and, similarly, a ray $h3 = (9,6)$ sent horizontally from the concave point 9 and its copy $h4 = (9,6)$ sent from the endpoint 9 of the slant edge (8,9). (b) The corresponding bipartite graph B, in which the vertices $h1 - h4$ form the maximum independent set. (c) The corresponding partitioning into the minimum number of trapezoids.

For each possible sliver, i.e., a pair of vertical (or horizontal) rays X_i and $X_{i'}$ such that $|X_i - X_{i'}| < \epsilon$, we introduce a sliver variable $sl(i,i')$ which is set to 1 if there is a sliver in the fracturing, and to 0 otherwise. The corresponding constraints are as follows:

$$sl(i,i') \geq x^v(i,j) + x^h(i',j) - 1, \quad j = 1, \ldots, hr \tag{10}$$

The resulting number of slivers is computed as

$$\#(slivers) = \sum_{|X_i - X_{i'}|<\epsilon} sl(i,i') + \sum_{|Y_j - Y_{j'}|<\epsilon} sl(j,j') \tag{11}$$

If, instead of counting the number of slivers, we wish to take into account the total length of slivers, we can use variables $sl(i,i',j)$ such that

$$sl(i,i',j) \geq x^v(i,j) + x^h(i',j) - 1 \tag{12}$$

The total length of slivers would then be computed as

$$\begin{aligned} L(slivers) \quad = \quad & \sum_{|X_i - X_{i'}|<\epsilon} \sum_j sl(i,i',j)(X_j - X_{j-1}) + \\ & \sum_{|Y_j - Y_{j'}|<\epsilon} \sum_i sl(i,j,j')(Y_i - Y_{i-1}) \end{aligned} \tag{13}$$

3. HEURISTICS TO SPEED UP THE ILP SOLUTION

The ILP described in the previous section, while producing high-quality solutions for the Fracturing Problem, cannot be solved sufficiently fast for large polygons. In this section, we discuss a general approach of partitioning the polygon P into sufficiently small sub-polygons, after which a standard ILP solver (e.g., CPLEX) can be applied. Of course, using such partitioning before applying ILP will increase the number of shots, since there are boundary and packing effects with respect to the maximum shot size and the partitioned sub-polygons. The main goal is to identify rays, either present or not present in the graph G, along which one can cut the polygon P without significantly compromising solution quality.

Plausible candidates for such rays are those that can be simultaneously used for two concave points. If we use a maximum number of such rays that do not intersect each other, we will minimize the total number of resulting trapezoids (see[5]). Our *matching* heuristic finds a maximum number of such rays, then uses them to partition the original polygon P, and finally solves ILP for each part separately. In the following, we describe our efficient algorithm for finding a maximum number of such rays, and justify the fact that it leads to polygon partitioning into the minimum number of trapezoids.

Since one ray should be sent from each concave point and each slant edge, the total number of rays is equal to the number of concave points C plus the number of slant edges S. 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 in the following two cases: (i) if they reach each other's origin, or (ii) if they are copies of each other - one of them has been sent because its origin is a concave point and another has been sent because its origin is an endpoint of a slant edge. See Figure 9 for an example illustrating each of these possibilities. 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 $C + S - I$. 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

$$\#(trapezoids) = C + S - I + 1$$

Thus, in order to minimize the number of trapezoids, one should maximize the set of pairs of coincident rays. Note that all such pairs of rays cannot intersect as well as cannot come from the same concave point or slant edge. 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 pairs of coincident 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 three cases (see Figure 9):

(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; or

(c) $u, v \in S$ and the rays correspond to the same slant edge.

The maximum set of pairs 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.

Testcase	# Polygons	Min # trapezoids	slants	# vertices
Design A	602	9613	21	24807
Design B	676	16273	17	38305
Design C	104	476	0	1580

Table 1. Properties of testcases.

	Design A				Design B				Design C		
Method	shots	slants	slivers	CPU(s)	shots	slants	slivers	CPU(s)	shots	slivers	CPU(s)
Tool A	10754	22	6111	0	17335	17	11572	0	589	318	0
Tool B	10455	23	4451	0	17130	17	10797	0	566	147	0
Tool C	9755	26	786	2	17195	21	6502	3	592	66	0
ILP+matching	9750	22	417	134	17684	17	2750	222	518	83	8

Table 2. Fracturing results with slivering size $\epsilon = 100nm$ and maximum shot size $M = 2.55\mu m$. We set $W_C = 100$ and $P_L = 30$ for ILP+matching.

		Design A				Design B				Design C		
W_C	P_L	shots	slants	slivers	CPU(s)	shots	slants	slivers	CPU(s)	shots	slivers	CPU(s)
100	30	9750	22	417	134	17684	17	2750	222	518	83	8
100	20	9801	22	527	12	17704	17	4280	46	523	94	2
100	15	9941	22	1049	5	18641	17	6219	21	564	123	1
1	30	9727	22	430	1329	17594	17	3361	16342	499	116	13
1	20	9882	22	563	52	17691	17	3953	614	523	143	5
1	15	10047	22	992	14	18417	17	7402	205	548	171	2

Table 3. Fracturing results with different values of W_C and P_L for ILP+matching. $\epsilon = 100nm$ and $M = 2.55\mu m$.

For any polygons which remain large after applying matching heuristics, we apply a heuristic "forced cut". We set a maximum vertex count P_L and divide any polygon whose number of vertices is greater than P_L into two parts such that the part with smaller number of vertices has at least $P_L/3$ vertices. The lower bound on the number of vertices in the resulting polygons ensures that optimization of the shot number remains non-trivial; this also limits the increase shot count caused by the forced cut.

4. EXPERIMENTAL RESULTS

We use three industry testcases to evaluate the performance of our fracturing approach. Design A and Design B are from Photronics, Inc..[12] Design C is a post-RET cell layout from a leading foundry 130nm standard-cell library; RET was inserted by Mentor Calibre. The basic properties of the three designs are listed in Table 1. For

each testcase, the minimum number of fractured trapezoids is calculated using the method given in Section 9. We have implemented our algorithm in ANSI C, and use the CPLEX 8.100 Mixed Integer Optimizer[11] to solve all Integer Linear Programming instances. In all runs, we set the runtime limit for CPLEX to 10 CPU seconds.

We have experimentally compared our fracturing code against state-of-art commercial fracturing tools. Two specific examples of leading commercial tools are Mentor Calibre v9.3_2.10 Fracturem[13] and Synopsys CATS v2501.[14] In our experiments, we set the maximum shot size as $2.55\mu m$ 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.¶

The fracturing results in Table 2 show that our method can reduce the number of slivers by 82%, 79% and 29% 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%.‖ The runtime of our method is much larger than that of the commercial tools due to the large runtime of the Integer Linear Programming solver. However, the runtime can be substantially reduced by reducing P_L at the cost of (acceptable) degradation of solution quality, as shown in Table 3.** We may also vary the scaling coefficient W_C that weights the number of slivers in the objective of the Fracturing Problem. Runtime increases with any decrease in W_C, since large W_C effectively forbid slivers and effectively reduces the solution space for the ILP.

Last, we note that the runtimes given in Tables 2 and 3 are for flattened layouts. When using hierarchical layout representation, the number of different polygons will be drastically reduced. Hence, the CPU cost of ILP solver can also be amortized in exact correspondence to any amortization of RET insertion costs: as soon as optical correction of a feature is fixed, fracturing can also be decided.

5. CONCLUSIONS

We have suggested a new ILP approach and fast heuristics based on ILP formulations for the fracturing problem in VSB mask writing. Our new approach improves both shot count and, very substantially, sliver count, in comparison to leading commercial mask data prep tools. Although the processing of layouts is much slower than in commercial products, we note that no effort has been spent on code optimizations. We believe that the ILP-based heuristic framework is scalable to full-chip layout processing, particularly when fracturing is done hierarchically (cf., e.g.,[2]) and/or in a distributed fashion.

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-aware "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:

- developing faster heuristics to speed up the ILP-based approach without compromising solution quality - specifically, by introducing new constraints and by reducing the size of ILP with additional rays from the polygon boundary;
- 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;
- further improving the quality of the ILP solution, e.g., by using unavoidable extra cuts that result from the maximum shot size constraint in the definition of the grid graph G; and

¶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.

‖We have also made a comparison when the slivering size is set to $50nm$. In that experiment, our method ILP+matching1 reduces the number of slivers by 96%, 97% and 81% compared with Tool A, Tool B, and Tool C respectively, while reducing the number of shots by 6.3%, 4.7% and 1.4%.

**Thus, while the commercial tools no doubt represent a heuristic tradeoff point between solution quality and runtime, we believe that the ILP-based approach can be made very competitive even with respect to runtime by caching results, exploiting hierarchy, using distributed computation platforms, etc.

- adjusting our approach to target non-rectilinear (e.g., X-style[16]) layouts with multiple slant edges.

REFERENCES

1. 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.
2. S. Shulze, E. Sahouria and E. Miloslavsky, "High Performance Fracturing for Variable Shaped Beam Mask Writing Machines," Proc. of SPIE 5130, pp. 648–659.
3. T. Asano, T. Asano and H. Imai, "Partitioning a polygonal region into trapezoids," J. ACM, **33**, 1986, pp. 290–312.
4. H. Imai and T. Asano, "Efficient algorithms for geometric graph search problems," SIAM J. Comput., **15**, 1986, pp. 478–494.
5. T. Ohtsuki, "Minimum dissection of rectilinear regions," Proc. ISCS, 1982, pp. 1210–1213.
6. N. Cobb and E. Sahouria, "Hierarchical GDSII based fracturing and jobdeck system,", Proc. of SPIE 4562, pp. 734–762.
7. N. Cobb and W. Zhang, "High performance hierarchical fracturing," Proc. of SPIE 4754, pp. 91–96.
8. Y. Granik and N.B. Cobb, "MEEF as a Matrix", *Proc. of SPIE*, 2002, Vol. 4562, pp. 980-991
9. W. Maurer, "Mask error enhancement factor", *Proc. of SPIE*, 2000, Vol. 3996, pp. 2-7
10. F.M. Schellenberg and C.A. Mack, "MEEF in theory and practice", *Proc. of SPIE*, 1999, Vol. 3873, pp. 189-202
11. CPLEX Mixed Integer Optimizer, ILOG.
    `http://www.cplex.com/.`
12. Photronics Inc.
    `http://www.photronics.com/.`
13. Calibre Fracturem, Mentor Graphics.
    http://www.mentor.com/calibre/datasheets/mdp/html/.
14. CATS, Synopsys.
    `http://www.synopsys.com/products/ntimrg/.`
15. Mask EDA workshop.
    `http://www.sematech.org/public/resources/litho/mask/maskeda/A_INTRO.pdf.`
16. `http://www.xinitiative.org/`