Benchmarking of Mask Fracturing Heuristics

Tuck-Boon Chan, *Student Member, IEEE*, Puneet Gupta, *Senior Member, IEEE*, Kwangsoo Han, *Student Member, IEEE*, Abde Ali Kagalwalla, *Student Member, IEEE*, and Andrew B. Kahng, *Fellow, IEEE*

***Abstract*—Aggressive resolution enhancement techniques such as inverse lithography (ILT) often lead to complex, nonrectilinear mask shapes which make mask writing extremely slow and expensive. To reduce shot count of complex mask shapes, mask writers allow overlapping shots, due to which the problem of fracturing mask shapes with minimum shot count is NP-hard. The need to account for e-beam proximity effect makes mask fracturing even more challenging. Although a number of fracturing heuristics have been proposed, there has been no systematic study to analyze the quality of their solutions. In this paper, we first propose a method to generate tight upper and lower bounds for actual ILT mask shapes by formulating mask fracturing as an integer linear program and solving it using branch and price. Since the integer program requires significant computational resources to compute reasonable bounds, we propose a new method to generate benchmarks with *known optimal* solutions, that can be used to evaluate the suboptimality of mask fracturing heuristics. To make the generated benchmark shapes realistic, we further propose a novel automated benchmark generation method that takes any ILT shape as input and returns a benchmark shape which looks similar to the input shape and for which the optimal fracturing solution is known. Using these methods, we compare the suboptimality of four mask fracturing heuristics. Our results show that even a state-of-the-art prototype (version of) capability within a commercial EDA tool for e-beam mask shot decomposition can be suboptimal by as much as 2.6× for real ILT shapes and by 6.0× for generated benchmarks.**

***Index Terms*—Benchmark testing, design for manufacture, integer linear programming, layout, relaxation method.**

I. Introduction

PHOTOMASKS are one of the most significant contributors to semiconductor manufacturing cost. The use of aggressive resolution enhancement techniques (RETs) has made mask manufacturing extremely expensive and challenging. Moreover, the number of critical masks required for a particular design has increased due to the use of multiple patterning. As a result, controlling the cost of mask manufacturing is urgently needed to sustain benefits derived from Moore's-Law scaling of patterning technologies.

Masks are fabricated using variable-shaped electron beam (VSB) writing tools. These tools directly expose *shots*, i.e., axis-parallel rectangles of different sizes. Mask *fracturing* is used to obtain a set of shots from the mask pattern, which can then be input to a VSB tool. Since the total shot count strongly affects mask fabrication time, the key objective of mask fracturing tools is to minimize the number of shots. This has been traditionally formulated as the well-studied rectilinear polygon partitioning problem. Imai and Asano [20] gave an $O(n^{1.5} \log(n))$ algorithm to partition a polygon into a minimum number of rectangles. Since such approaches are unable to handle additional manufacturing constraints such as minimization of slivers, Kahng *et al.* [24] proposed an ILP-based fracturing method, and a faster heuristic based on selection of rays from concave corners [25].

Due to aggressive RET such as inverse lithography (ILT), mask shapes are curved and nonrectilinear [6]. Fracturing these polygons using traditional methods with acceptable fidelity can dramatically increase the shot count [34]. To manage the shot count of such complex patterns, [7] proposes model-based fracturing, which is also often referred to as *model-based mask data preparation* (MB-MDP). Two key features of model-based fracturing distinguish it from traditional mask fracturing.

1) Shots may overlap, that allows more flexibility in determining shot locations and hence lower shot count.
2) E-beam proximity effects in VSB mask writers are simulated during the mask fracturing itself to ensure that the final mask pattern matches the intended target.

The model-based mask fracturing problem allowing overlapping shots becomes similar to the rectilinear covering problem, which is NP-hard [10]. In fact, there is no known constant-factor approximation algorithm for rectilinear covering [2]. For polygons which are convex in the vertical or horizontal direction, [12] proposes a quadratic-time algorithm to solve the covering problem. But, this convexity property rarely holds for ILT shapes. Franzblau [13] proposed an approximation algorithm with worst-case performance bounds for any rectilinear polygon. However, the need to correct for proximity effects means that these methods cannot be used for mask fracturing.

In addition to overlapping shots and proximity effect correction, [16] proposes to adjust the dose of each shot independently. The use of L-shaped shots to reduce shot count has been suggested by [33]. The use of circular shots [15], [34] or shots with 45° edges [9] has also been proposed. Elayat *et al.* [11] analyzed the benefits and disadvantages of different mask fracturing strategies. They conclude that, among the alternatives studied, using axis-parallel rectangle shots with fixed dose is the most viable candidate for improvement of shot count without significant changes in mask

Manuscript received March 5, 2015; revised June 5, 2015 and September 23, 2015; accepted November 16, 2015. Date of publication October 24, 2016; date of current version December 20, 2016. This work was supported by the IMPACT+ Program. This paper was recommended by Associate Editor C. C.-N. Chu.

T.-B. Chan and K. Han are with the Department of Electrical and Computer Engineering, University of California at San Diego, San Diego, CA 92093-0407 USA.

P. Gupta and A. A. Kagalwalla are with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA.

A. B. Kahng is with the Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA, and also with the Department of Electrical and Computer Engineering, University of California at San Diego, San Diego, CA, USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2620902

TABLE I
GLOSSARY OF TERMINOLOGY

Term	Meaning	Term	Meaning
S	Set of all possible candidate shots (i.e., with different sizes and locations) for fracturing of target shape	$\zeta(\theta)$	Distance between two parallel lines of the same θ. One line is across a corner point of a shot and the other line is the closest tangent line of the shot image
P_d	Set of pixels lying within distance γ of boundary of t_{ori}	$P_1(P_0)$	Set of pixels within (outside) the boundary of t_{ori} not belonging to P_d
s	Particular candidate shot under consideration	$I(x,y,s)$	Intensity at pixel $p(x,y)$ due to shot s
$W(s),H(s)$	Width and height of shot s	ρ	Threshold value for e-beam resist
t_{ori}	Target mask shape to fracture	γ	CD tolerance limit for fracturing
$p(x,y)$	Pixel of a mask shape at coordinate (x,y)	(x,y)	Coordinates of particular point on the mask
$S_{min}(t_{ori})$	Minimal set of shots used to fracture t_{ori} (depends on γ,σ,ρ)	Δ	Shot size granularity
$W_{min}(W_{max})$	Minimum (maximum) shot size allowed	Δ_{MP}	Shot size granularity used in matching pursuit heuristic
$I(x,y)$	Total intensity at $p(x,y)$ due to all shots in current solution	z_s	0-1 variable indicating whether candidate shot s is part of solution
$W(t_{ori}),H(t_{ori})$	Width and height of bounding box of t_{ori}	$B_v(B_h)$	Vertical (horizontal) boundary segments of t_{ori}
b_i	i^{th} boundary segment	$t_{ori,i}$	i^{th} split-shape of t_{ori}
λ_p^*	Value of dual variable at pixel $p(x,y)$	P_{neg}	Set of pixels for which $\lambda_p^* \leq 0$
$(x_{bl}(s),y_{bl}(s))$	Bottom-left coordinate of candidate shot s	N_C	Maximum number of candidate shots inserted in one pricing round
$(x_{tr}(s),y_{tr}(s))$	Top-right coordinate of candidate shot s	B^n	Set of all boundary segments that require at least n shots to construct
β	Maximum distance outside shot at which the intensity $\geq 10^{-6}$	σ	Parameter characterizing the spreading of the e-beam
b^n	Particular boundary segment that requires at least n shots	θ	Angle between straight-line boundary segment and x-axis
$L^{\theta}_{lin}(W,H)$	Longest straight-line boundary segment having angle θ with the x-axis that can be covered with a shot of width W and height H	α	Maximum distance outside target boundary where candidate shot corner can lie without exposing any pixel in P_0
$b_{seg\_in}(b_{seg\_out})$	Inner (outer) segment parallel to b_{seg} obtained by shifting b_{seg} by γ	L_t	Defined in Figure 7(a)
$L^{\theta}_{con}(W,H)$	Length of concave boundary segment, having angle θ with the x-axis, made with shot of size $W \times H$	L^{θ}_{max}	Maximum length of linear boundary segment, having angle θ with the x-axis, that can be made with any rectangular shot
$b_{main}(b_{cri})$	Main (critical) boundary segment	$t_{gen}(t_{gen,i})$	Generated benchmark shape which resembles t_{ori} $(t_{ori,i})$
$V(t)$	Set of vertices of shape t	$v_k(t)$	k^{th} vertex of shape t
L^{θ}_{th}	Minimum distance between any two shot corner points that generate the boundary segment b^2	V_{main}	Ordered sublist of $V(t_{ori,i})$ such that the straight-line segments connecting them approximate b_{main} of $t_{gen,i}$
$v_{main,i}$	i^{th} vertex of V_{main}	b_{seg}	Boundary segment under consideration
V^{sft}_{main}	Set of points obtained by shifting V_{main}	C_{main}	Set of shot corner points lying on line segments connecting V_{main}
C^{opp}_{main}	Set of opposite corner points of C_{main}	$d(t_a,t_b)$	Number of error pixels between shapes t_a and t_b
$L(c_a,c_b)$	Distance between the points c_a and c_b	$b_{gen,i}$	Boundary segment of generated shape $t_{gen,i}$

writing tools. Hence, in this paper we only use axis-parallel rectangle shots with fixed dose.

Jiang and Zakhor [21] proposed an algorithm based on matching pursuit (MP) to solve the fracturing problem, and a greedy approximate covering algorithm that grows rectangles from convex vertices of a target polygon [22]. Both of these heuristics assume an adjustable shot dosage, but can be extended to solve the fixed-dose problem. Lin *et al.* [26] compared several heuristics to solve the model-based fracturing problem. While these recent works on model-based mask fracturing have demonstrated improvements in shot count over traditional partitioning-based approaches, the gap between existing methods and optimal solutions remains unclear.

Benchmarking of heuristics used to solve NP-hard EDA problems such as gate sizing [17] enables the development of better methods for solving these problems. The goal of this paper is to enable the benchmarking of model-based fracturing as a foundation for further research toward more effective heuristics. To the best of our knowledge, this is the first work that attempts to benchmark model-based mask fracturing. The key contributions of this paper are as follows.

1) We propose an ILP formulation to optimally solve the model-based mask fracturing problem. We then develop a branch and price (B&P) method that, in practice, generates strong upper bound (UB) and lower bound (LB) for benchmarking.
2) To deal with the slow runtime of ILP-based benchmarking, we propose a systematic method to generate benchmarks with known optimal shot count.
3) To generate more realistic benchmarks, we propose an automated benchmark generation method that takes a real ILT shape as input and creates a benchmark with known optimal shot count that looks similar to the input shape.
4) Using the above methods, we evaluate the suboptimality of four mask fracturing heuristics: greedy set cover (GSC), MP, graph coloring (GC) and a state-of-the-art prototype (version of) capability within a commercial EDA tool for e-beam mask shot decomposition (PROTO-EDA).

In the following, Section II defines the mask fracturing problem, and Section III describes mask fracturing heuristics that we benchmark. Section IV proposes an ILP-based method to obtain tight UB and LB on optimal shot count. Section V gives our method for benchmark generation with known minimum shot count. Section VI presents our method to automatically generate benchmarks with known minimum shot count which are similar to input mask shapes. Table I summarizes our notations.

II. MASK FRACTURING PROBLEM

The goal of mask fracturing is to find the minimum number of *rectangular* shots required to construct a mask *target shape*. Although each shot is rectangular, the *e-beam proximity effect* blurs its boundary [7]. Hence, the developed mask pattern differs from the union of rectangular shots. Also, since the blurring due to the e-beam proximity effect is smaller than the spacing between different shapes, each shape in the mask can be fractured independently. Moreover, to better understand which target shapes are more challenging, the suboptimality of mask fracturing heuristics should be evaluated for individual mask target shapes rather than for the entire mask.

We define S as the set of all possible candidate shots that could be used to reconstruct the target shape t_{ori}, i.e., the dictionary of candidate shots. S consists of all shifted copies of all different shot sizes ranging from $W_{\min}$ to $W_{\max}$, with shot size granularity of Δ.[1] E-beam proximity effect is modeled using a low-pass filter, typically a Gaussian or sum of Gaussians [30]. In this paper, we model the proximity effect by a single 2-D Gaussian low-pass filter, described by (1). However, our proposed methods for benchmarking can be easily extended to handle other proximity effect models

$$K(x,y) = \begin{cases} \dfrac{1}{F}\exp^{-\frac{x^2+y^2}{\sigma^2}} & \text{if } -3\sigma \leq \sqrt{x^2+y^2} \leq 3\sigma \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

[1]The step size of shifting is also Δ, i.e., all shots are rectangles in a discrete grid.

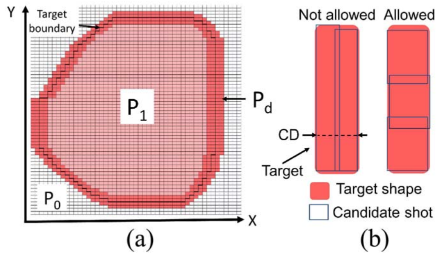


Fig. 1. (a) Each grid is a pixel $p(x, y)$. The thick black line is the target boundary. CD tolerance is $\gamma = 2$ nm and the grid size is 1 nm × 1 nm. $p(x, y) \in P_d$ if $p(x, y)$ is within 2 nm of the target boundary. (b) Illustration of CD control constraint for candidate shots for a horizontal critical region.

$K(x, y)$ is the kernel function of the Gaussian filter, F is a normalization factor [i.e., sum of $K(x, y)$ across all values of x and y] and σ is a parameter which characterizes the spreading of the e-beam. For any rectangular shot s, the intensity at a pixel can be computed by convolving the *ideal rectangular function* [$\psi(\hat{x}, \hat{y})$] [4] with the kernel function. That is

$$I(x, y, s) = K(x, y) \otimes \psi\left(\frac{(x - x_{c,s})}{W(s)}, \frac{(y - y_{c,s})}{H(s)}\right)$$
$$\psi(\hat{x}, \hat{y}) = \begin{cases} 1 & \text{if } |\hat{x}| < 0.5 \text{ and } |\hat{y}| < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $x_{c,s}$ and $y_{c,s}$ are the x and y coordinates of the center of the shot. In this paper, all dimensions are in wafer scale.[2]

We model the e-beam resist using a constant-threshold model with threshold value of ρ. A given pixel [$p(x, y)$] on the mask will be *exposed* if and only if the total intensity at that pixel resulting from all shots is greater than or equal to the resist threshold ρ.[3] As shown in Fig. 1(a), we divide the set of pixels on the mask into three disjoint sets: 1) P_1; 2) P_0; and 3) P_d. have intensity $\geq \rho$. Similarly, we define P_0 as the set of the pixels outside the target shape which do not belong to P_d. The pixels in P_0 must have intensity $< \rho$.

The mask fracturing problem is formally defined as follows.

1) *Goal:* Minimize total #mask shots $N = |S_{\min}(t_{\text{ori}})|$.
2) *Inputs:* Mask target shape, set of candidate shots S, ρ, σ, γ.
3) *Outputs:* Set of rectangular shots, $S_{\min}(t_{\text{ori}})$.
4) *Constraints:*

$$\sum_{s \in S_{\min}} I(x, y, s) \geq \rho \text{ if } p(x, y) \in P_1$$
$$\sum_{s \in S_{\min}} I(x, y, s) < \rho \text{ if } p(x, y) \in P_0. \tag{3}$$

CD control of the target pattern is a key concern for mask manufacturing. To minimize CD variation, any critical vertical or horizontal segment of the target shape boundary should not be constructed with more than one shot [32] [see Fig. 1(b)]. To check if a particular candidate shot satisfies the CD control constraint, we first identify critical vertical/horizontal regions of a given target shape (i.e., long and narrow part of a target shape). Any candidate shot in S that overlaps with these horizontal (vertical) critical regions must have both of its vertical (horizontal) edges touching the target boundary.

The mask writing process may also require additional constraints to avoid resist over-heating. In this paper, we do not consider the imposition of maximum intensity constraints to model resist over-heating, since the over-heating is an effect at length scales on the order of microns [14].

III. Fracturing Heuristics

To evaluate the suboptimality of different mask fracturing heuristics, we have implemented two simple methods to fracture mask shapes, based on prior work, that we describe in this section. The fracturing solutions created by both these heuristics tend to have CD violations, i.e., pixels that violate constraint (3). Hence, we use an additional step, *shot refinement*, to fix the CD violations. In addition to the two simple heuristics, we evaluate the suboptimality of a PROTO-EDA. We further evaluate the recent heuristic of Kagalwalla and Gupta [23], which uses a combination of GC-based approximate fracturing and shot refinement.

The first heuristic, GSC, is inspired by the well-known greedy approximation algorithm for the NP-complete set cover problem [8]. Note that the fracturing problem (in the absence of proximity effect) is a geometric set cover problem, so we choose this heuristic for comparison. We first construct a Hanan grid [18] by constructing x- and y-axis-parallel lines from each vertex of the target polygon. Every grid element that lies inside the polygon and contains at least one pixel $p(x, y) \in P_1$ is considered an element to be covered in the set cover problem. We then find all the maximal rectangles lying inside the polygon.[4] Each maximal rectangle is treated as a "set" that covers some of the grid elements. The fracturing problem reduces to the set cover problem to which we apply the greedy approximation algorithm. Note that the e-beam proximity model is not considered in constructing this fracturing solution but is handled during the shot refinement.

The second heuristic, MP, is a well-known technique to represent a signal sparsely for an over-complete basis set [28].[5] Jiang and Zakhor [21] proposed a technique to apply this method to the mask fracturing problem. A dictionary of different shot sizes is first constructed. To keep the dictionary size tractable, some step size $\Delta_{\text{MP}} \geq \Delta$ is used to discretize the width/height range between $W_{\min}$ and $W_{\max}$.[6] The proximity model is applied to each shot in the dictionary. We then iterate over the dictionary and over all potential positions of the candidate shot to pick the shot that maximally reduces the *residual error* (RE). This procedure is repeated until no shot is found that can reduce the RE, defined to be the sum of $|I(x, y) - \rho|$ over all pixels which violate the CD constraint.

The third heuristic, GC, proposed recently by Kagalwalla and Gupta [23], traverses the boundary of

[2] Typically, mask scale is 4× wafer scale.

[3] The exposed pixels will form the mask shape.

[4] A rectangle is maximal if all four edges touch the boundary of the target polygon.

[5] Although there are several papers on model-based mask fracturing (see [7]), there is a dearth of papers that talk about the specific method being used. The only specific method we found was MP, so we have implemented that for comparison.

[6] We use a large shot size granularity Δ_{MP} only during the first step of generating an approximate fracturing solution that is then refined to obtain the final valid solution. The refinement step moves shots in steps of size Δ. Consequently the final solution of MP uses shots with the same granularity as B&P. We note that keeping the granularity of MP large during the first step is necessary to ensure reasonable runtime.

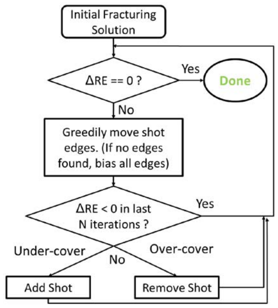


Fig. 2. Flowchart of shot refinement step for GSC and MP heuristics.

the target shape to identify candidate shot corner point locations. Treating every shot corner point as a graph vertex, mask fracturing is mapped to a GC problem such that each color corresponds to a shot. The NP-hardness of GC motivates use of a sequential greedy heuristic [29] to find an approximate fracturing solution.

After obtaining initial fracturing solution from GSC, MP, and GC heuristics, we perform *shot refinement* step as shown in Fig. 2. The shot refinement step moves edges of shots greedily to minimize the RE. Once this procedure stops reducing the RE, we bias all the shots of the current solution by a small value. If the number of pixels in set P_0 that violate the CD constraint (over-cover) is greater than the number of pixels in P_1 that violate the CD constraint (under-cover), then we shrink all shots; otherwise, we expand all shots during this bias step. After biasing, we continue with the greedy shot edge adjustment. If this iterative procedure fails to reduce the RE for more than N iterations, we add (remove) one shot if more pixels are under-covered (over-covered). We terminate when the RE is zero, i.e., there are no CD violations. We note that this procedure does not guarantee a feasible (CD error-free) fracturing solution, and that some results in our experiments have CD errors.

IV. ILP-Based Benchmarking

To evaluate the suboptimality of fracturing heuristics on any given mask shape, we apply an optimal ILP formulation. The straightforward ILP formulation requires a large number of binary variables, even for small target shapes. As a result, even commercial ILP solvers can run out of memory on high-performance computers. To circumvent this, we propose three strategies, described in this section: 1) pruning the set of candidate shots; 2) splitting large target shapes; and 3) solving the ILP using B&P. With these strategies, we can obtain strong UB and LB on the optimal solution within feasible runtime. Note that although the proposed ILP can be used to inspire effective mask fracturing heuristics, the goal of this paper is benchmarking. Hence, runtime is important only in the context of making the method tractable.

A. Optimal ILP Formulation

Inspired by the ILP formulation of Heinrich-Litan and Lübbecke [19], we proposed a simple ILP formulation for the model-based mask fracturing problem. We define a binary selection variable z_s for each candidate shot $s \in S$, where $z_s = 1$ if shot s is used and $z_s = 0$ otherwise. Then, based on the problem description in Section II, we may formulate an optimal ILP to solve the fracturing problem as

$$\begin{aligned} \text{Minimize} \quad & \sum_s z_s \\ \text{subject to} \quad & \sum_s \{z_s \cdot I(x,y,s)\} \geq \rho, p(x,y) \in P_1 \\ & \sum_s \{z_s \cdot I(x,y,s)\} < \rho, p(x,y) \in P_0. \end{aligned} \tag{4}$$

The problem with this ILP formulation is that $|S|$ can be very large even for small target shapes. For a target shape t_{ori} with a bounding box of $W(t_{\text{ori}}) \times H(t_{\text{ori}})$, if the shot size granularity is Δ and no shots are disallowed due to the CD control constraint, then the size of the set of candidate shots would be $[((W_{\max} - W_{\min})/\Delta)]^2 \cdot [W(t_{\text{ori}}) - (W_{\max} + W_{\min}/2)] \cdot [H(t_{\text{ori}}) - (W_{\max} + W_{\min}/2)]$, where $W_{\min}$ and $W_{\max}$ are the minimum and maximum allowed shot sizes. Even for small mask shapes, the corresponding ILP is too large for commercial solvers to handle due to runtime and excessive memory usage.[7] In fact, the *CPLEX v12.5* solver [36] runs out of memory when we attempt to solve an instance on an Intel Xeon L5420 with 128GB RAM.[8]

B. Pruning the Candidate Shot Dictionary

Reducing $|S|$ can significantly help in making the above ILP tractable for benchmarking. Here we highlight two simple rules that can be used to reduce $|S|$.

1) For any candidate shot s, if there exists a pixel $p(x, y) \in P_0$ such that $I(x, y, s) \geq \rho$, then s can be removed from the set S. This pruning condition obviously does not affect optimality because any candidate shot that satisfies this condition cannot be a part of a feasible solution of the ILP. Depending on the specific target shape, this pruning strategy can significantly reduce $|S|$.
2) If a candidate shot s is inside the target shape and none of its four edges are less than the distance γ from the target boundary, then we remove s from set S. If s is a part of the optimal solution, then we can replace s with a larger shot that covers s and has at least one boundary close to the edge of the target shape, without affecting the optimality of the solution.

The reduction in $|S|$ due to these pruning rules depends strongly on the specific target shape and the e-beam proximity effect model. For certain target shapes, the number of variables even after pruning could be $> 10^6$, making it difficult to solve the problem efficiently with commercial ILP solvers.

C. Splitting Target Shapes

The size of the mask fracturing ILP depends on the size of the bounding box of the target shape [i.e., the number of variables (candidate shots) and the number of CD constraints (pixels)]. The ILP can become intractable for large target shapes. In this section, we propose a simple strategy that can be used to split large target shapes into two or more *split-shapes*. This allows us to solve a separate smaller ILP for each smaller split-shape. The fracturing solutions of the smaller ILP instances can then be aggregated to obtain a solution of the

[7]We have formulated alternative ILPs with fewer variables, but these turned out to be even harder for the ILP solver, for numerical reasons. We have assiduously explored the possibility of ILP speedups and, to our current understanding, currently apply "best known methods" in expressing our ILP to the CPLEX solver.

[8]The details of the instance are described in [5].

Algorithm 1 Determine Horizontal Locations to Split Target

Input: Target shape t_{ori} and length threshold L_{th}
Output: Locations where t_{ori} is split
1: $B_v \leftarrow$ all vertical boundary segments longer than L_{th}, sorted by x-coordinate
2: **for all** $b_i \in B_v$ **do**
3: **for all** $b_j \in B_v$ **do**
4: $low \leftarrow max(b_i.low, b_j.low)$
5: $high \leftarrow min(b_i.high, b_j.high)$
6: **if** $b_i \neq b_j \&\& (high - low) > L_{th}$ **then**
7: $(x_{bl}(rect), y_{bl}(rect)) \leftarrow (b_i.val, low)$
8: $(x_{tr}(rect), y_{tr}(rect)) \leftarrow (b_j.val, high)$
9: **if** *rect* lies inside t_{ori} **then**
10: Split location $\leftarrow$ Line segment from $(b_i.val, \frac{low+high}{2})$ to $(b_j.val, \frac{low+high}{2})$
11: **end if**
12: **end if**
13: **end for**
14: **end for**

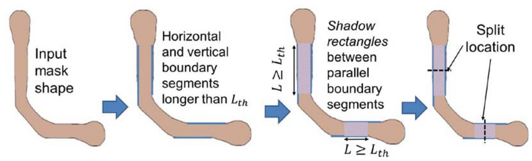


Fig. 3. Splitting a target polygon into smaller polygons.

full target shape; this can then be used as the initial solution of the larger ILP corresponding to the full target shape.

When splitting any target shape, the key step is to determine locations where the shape should be split. We find horizontal and vertical line segments which serve as *split locations* (see Fig. 3). Algorithm 1 describes the procedure we use to obtain horizontal split locations. An analogous procedure can be used to find vertical split locations.

In Algorithm 1, we first identify vertical *boundary segments*[9] of t_{ori} which are longer than L_{th}. Each such vertical boundary segment b_i is characterized by three parameters: the x-coordinate of the orthogonal segment (b_i.val), and the respective y-coordinates of the two end points of the segment (b_i.low and b_i.high). Using these, we then find pairs of parallel vertical segments which satisfy the following two conditions: 1) the length of the *shadow rectangle* between the two parallel segments is longer than L_{th} (line 6) and 2) the shadow rectangle lies inside the target polygon t_{ori} (line 9). Then, we can split t_{ori} at the center of the shadow rectangle.

Suppose that we split a target shape into two shapes $t_{\mathrm{ori},i}$ and $t_{\mathrm{ori},j}$ with a horizontal split location using Algorithm 1. Let $L_{\mathrm{th}} \geq 2\beta$, where β is the maximum distance outside the shot for which the shot intensity is nonzero. Then, we can bound minimum shot counts of t_{ori} as a function of minimum shot count of $t_{\mathrm{ori},i}$ and $t_{\mathrm{ori},j}$ based on the following two lemmas.

Lemma 1:

$$|S_{\min}(t_{\mathrm{ori}})| \leq \left|S_{\min}\left(t_{\mathrm{ori},i}\right)\right| + \left|S_{\min}\left(t_{\mathrm{ori},j}\right)\right|.$$

Proof: If we consider all the shots of an optimal fracturing solution of the two split-shapes $t_{\mathrm{ori},i}$ and $t_{\mathrm{ori},j}$, we vertically extend all the shots that touch the split location by one pixel, so that these shots from $t_{\mathrm{ori},i}$ and $t_{\mathrm{ori},j}$ overlap at the split location. This modified set of shots is a feasible solution that satisfies all the CD constraints. The size of this feasible solution is an UB on the size of the optimal solution of the full shape. Note that combining solutions of the smaller split-shapes is valid if and only if the distance from the horizontal split line to the top (bottom) of the shadow rectangle is large enough that extending the shot from the lower (upper) split-shape by one pixel above (below) the horizontal split line does not add intensity to any pixel above (below) the shadow rectangle. This is guaranteed when we set $L_{\mathrm{th}} = 2\beta$. ■

Lemma 2:

$$|S_{\min}(t_{\mathrm{ori}})| \geq \max\left(\left|S_{\min}\left(t_{\mathrm{ori},i}\right)\right|, \left|S_{\min}\left(t_{\mathrm{ori},j}\right)\right|\right).$$

Proof: Without loss of generality, assume that $|S_{\min}(t_{\mathrm{ori},i})| \geq |S_{\min}(t_{\mathrm{ori},j})|$, and that $t_{\mathrm{ori},i}$ lies below a horizontal split location. Suppose toward a contradiction that there exists an optimal fracturing solution $S^*_{\min}(t_{\mathrm{ori}})$ for target shape t_{ori}, such that $|S^*_{\min}(t_{\mathrm{ori}})| < |S_{\min}(t_{\mathrm{ori},i})|$. From this solution, we can obtain a feasible fracturing solution for $t_{\mathrm{ori},i}$ by taking all the shots that lie below the split location and splitting all shots that overlap with the split location. Clearly, the number of shots in this fracturing solution satisfies $|S^*_{\min}(t_{\mathrm{ori},i})| \leq |S^*_{\min}(t_{\mathrm{ori}})|$ since only a subset of the shots in the fracturing solution of t_{ori} are used. This implies $|S^*_{\min}(t_{\mathrm{ori},i})| < |S_{\min}(t_{\mathrm{ori},i})|$, which is impossible. Consequently our original assumption must be incorrect, and $|S_{\min}(t_{\mathrm{ori}})| \geq |S_{\min}(t_{\mathrm{ori},i})|$. ■

After splitting t_{ori} using the method described above, we solve a separate ILP for each split-shape. We then combine the fracturing solution of the separate ILPs to obtain a feasible fracturing solution for t_{ori}, which we use as a starting solution for the larger ILP for t_{ori}. Note that this splitting technique is effective only if the target shape boundary contains long vertical and/or horizontal boundary segments.

D. Branch and Price Method

B&P is a well-known method for solving large ILPs [3]. The key feature that distinguishes B&P from typical ILP solvers is that the LP relaxation at each node of the branch and bound tree is solved using column generation. To solve the LP relaxation, which contains too many variables to handle efficiently, a reduced master problem (RMP) containing only a small subset of the variables is solved first. To confirm the optimality of this RMP, a separate pricing subproblem is solved to find any new variables that must be inserted back into the RMP. If no variable is found by the pricing subproblem, then the LP relaxation is optimal and branching can be done to obtain the integral solution to the original ILP.

The selective insertion of variables based on the pricing subproblem in B&P means that most variables are never inserted into the LP relaxation. As a result, the LP relaxation solver does not consume too much memory. This is the main reason why we choose to apply this technique to solve the ILP described in (4). The runtime of B&P is known to be limited by the pricing subproblem for most problems [3]. Hence, we propose a novel pricing mechanism comprising a fast, approximate pricer, and a slower, optimal pricer.

The goal of pricing subproblem is to identify additional variables that must be inserted into the RMP. For mask fracturing problem, let λ^*_p be the optimal value of the dual variable corresponding to CD constraint (4) at pixel $p(x, y) \in P_1 \cup P_0$, obtained after an iteration of the RMP. The pricing subproblem (derived from the dual of the RMP) reduces to finding a new candidate shot s such that $\sum_p I(x, y, s) \cdot \lambda^*_p \leq -1$. This candidate shot must satisfy the pruning rules discussed above. Moreover, additional constraints imposed by the branching rules of branch and bound tree must be met. The *reduced cost* of any candidate shot s is given by $R_s = 1 + \sum_p \{I(x, y, s) \cdot \lambda^*_p\}$. In short, we refer to any candidate shot that has $R_s \leq 0$ and satisfies all pruning and branching constraints as an insertable candidate shot (ICS).

[9] A boundary segment is a contiguous part of the boundary of a target shape.

Algorithm 2 Fast Pricer Heuristic

Input: Target shape t, and list of pixels with negative dual values P_{neg}
Output: Set of inserted candidate shots to be added into the RMP
1: **for all** $p(x, y) \in P_{neg}$ **do**
2: Draw vertical/horizontal line from (x, y) to find y_{low}, y_{high}, x_{low} and x_{high} (iillustrated in [5, Fig. 9])
3: Find all candidate shots in vicinity of (x, y) that satisfy Equation (5) below
4: Insert (up to $\frac{N_C}{|P_{neg}|}$) candidate shots that satisfy reduced cost, pruning and branching constraints into the RMP
5: **end for**

To ensure that the LP relaxation is solved optimally, pricing subproblem must guarantee that no ICS exists. If there are several ICSs, the pricing subproblem only needs to find a subset of all the ICSs in an iteration. To improve the convergence of B&P, we set the maximum number of candidate shots that are inserted in each pricing iteration as $N_C = 500$.

One strategy to solve the pricing problem is to enumerate all possible sizes and locations of candidate shots and insert any shot that has a negative reduced cost and satisfies pruning and branching rules. To improve efficiency of this naive pricing strategy, we analyze the dual variables of the RMP. Based on the Karush-Kuhn-Tucker conditions, the following holds for the dual variables.

1) Due to complementary slackness, $\lambda_p^* \neq 0$ if and only if $\sum_s \{z_s \cdot I(x, y, s)\} = \rho$. Since this is likely to occur only close to the boundary of the target shape, λ_p^* is nonzero only for a small number of pixels that lie very close to the target boundary. We shall refer to the set of pixels with nonzero dual values as *dual points*.
2) To ensure dual feasibility, $\lambda_p^* > 0$ for $p(x, y) \in P_0$ and $\lambda_p^* \leq 0$ for $p(x, y) \in P_1$. This implies that all *negative dual points* (P_{neg}) with $\lambda_p^* \leq 0$ are located inside the target shape.

That negative dual points are sparse and located close to the target shape boundary is illustrated in [5, Fig. 9] for a particular pricing iteration of a target shape. With this insight, we propose two pricing strategies to effectively find ICSs.

1) Fast Pricer: The basic idea of the fast pricer is to look for ICSs in the vicinity of $p(x, y) \in P_{\text{neg}}$, since any candidate shot s with negative reduced cost must be located such that it covers or is close to at least one negative dual point (see Algorithm 2). The intuition behind constraining $x_{\text{bl}}(s)$, $y_{\text{bl}}(s)$, $x_{\text{tr}}(s)$ and $y_{\text{tr}}(s)$ as shown in (5) is that such candidate shots will have nonzero intensity at the negative dual point under consideration and are likely to obey the first pruning rule (not exposing any pixel in P_1)

$$
\begin{aligned}
&x_{\text{low}} - \alpha \leq x_{\text{bl}}(s) \leq x + \beta,\ \ x - \beta \leq x_{\text{tr}}(s) \leq x_{\text{high}} + \alpha \\
&y_{\text{low}} - \alpha \leq y_{\text{bl}}(s) \leq y + \beta,\ \ y - \beta \leq y_{\text{tr}}(s) \leq y_{\text{high}} + \alpha.
\end{aligned} \tag{5}
$$

2) Optimal Pricer: Although our pricing heuristic above effectively identifies most ICSs which can be inserted into the RMP, it does not guarantee that if no ICS is found, then there does not exist any ICS. Hence, if the heuristic fails to find any ICS, we call the optimal pricer that is guaranteed to find a candidate shot with negative reduced cost, if it exists. The optimal pricer iterates over all candidate shots in the vicinity of the negative dual points. The method is described in Algorithm 3. The optimal pricer first constructs square boxes of size $2 \times \beta$ centered at each negative dual point. Any candidate shot which could have negative reduced cost must overlap with at least one of these boxes. Hence, we could iterate over all such candidate shots to find ICSs. But several dual points may lie close to each other which may cause candidate shots to be generated twice. To avoid this, we first merge the boxes using polygon Boolean OR operation. We then find the bounding box of each resulting polygon. All candidate shots that overlap with these bounding boxes are then checked for insertion into the RMP.

Algorithm 3 Optimal Pricer

Input: Target shape t, and list of pixels with negative dual values P_{neg}.
Output: Set of candidate shots inserted into the RMP
1: *mergedBoxes* $\leftarrow$ new list of polygons
2: **for all** $p(x, y) \in P_{neg}$ **do**
3: *rect* $\leftarrow$square box of size $2 \times \beta$ with $p(x, y)$ as center
4: *mergedBoxes* $\leftarrow$ *mergedBoxes* $\bigvee$ *rect* (Polygon Boolean OR operation)
5: **end for**
6: **for all** *polygon* $\in$ *mergedBoxes* **do**
7: Find bounding box of *polygon*
8: Find all candidate shots that overlap with the bounding box of *polygon*
9: Insert (up to $\frac{N_C}{|P_{neg}|}$) candidate shots that satisfy reduced cost, pruning, CD control and branching constraints into the RMP
10: **end for**

E. Initialization and Overall Summary

In addition to solving the pricing subproblem efficiently, B&P benefits significantly from a good initial feasible solution. B&P can discover feasible solutions using Farkas pricing [1], but can take many iterations of pricing. In this paper, we use the lowest shot count solution that satisfies all the constraints, over the results from the GSC, MP, and GC heuristics, as the initial solution for B&P. Although B&P resolves the problem of excessive memory usage, it takes a long time to converge to the optimal solution. Since our objective is to evaluate suboptimality, we run B&P with a fixed time limit and report UB and LB on the optimal shot count.

For each split-shape $t_{\text{ori},i}$, we first obtain an initial solution and solve the ILP using B&P. Based on the UB and LB of each smaller ILP, we obtain lower and upper bounds for t_{ori} using Lemmas 1 and 2. To improve these bounds, we combine the solutions from each split-shape by merging shots lying at the split locations. If this does not give a feasible solution for t_{ori}, we apply shot refinement to fix the violations and use this as the initial solution for solving the larger ILP for t_{ori}. To improve the runtime of the pricing method, we parallelize both fast and optimal pricing methods. For fast pricer, line 4 in Algorithm 2 can be parallelized since candidate shot can be checked for insertion to the RMP independently. Similarly for optimal pricer (Algorithm 3), line 9 can be parallelized. Note that the use of fast pricer before optimal pricer is critical for faster convergence.[10]

F. Experimental Results

Our B&P-based suboptimality evaluation method has implemented in C++. We use the OpenAccess API [39] to parse layouts, Boost Polygon Library [35] to perform polygon operations, and Eigen Library [37] to perform matrix operations. To implement B&P, we use the SCIP framework [1], along with *CPLEX v12.5* [36] as the LP solver. We parallelize the pricing methods using OpenMP [40].

We set the resist threshold $\rho = 0.5$ and use a Gaussian e-beam proximity effect model with two values of σ, 6.25 nm and 4 nm.[11] For CD tolerance γ, we consider values of

[10] In our experiments, the wall time of optimal pricer is $10-100\times$ the wall time of fast pricer.

[11] $\sigma = 6.25$ nm is consistent with recent work on mask fracturing [21], [22]. We also show results for $\sigma = 4$ nm to highlight the impact of Gaussian blur on shot count.

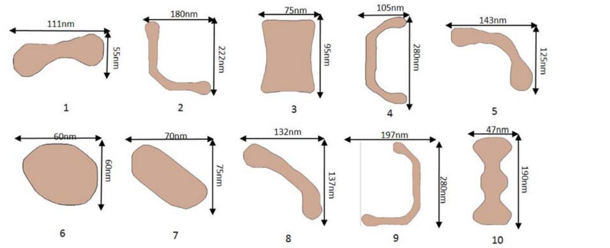


Fig. 4. ILT mask shapes after applying ILT from the ICCAD-2013 contest [38] (wafer scale).

2 and 1 nm. The shot dimension constraints are $W_{\min} = 13$ nm, $W_{\max} = 1000$ nm and $\Delta = 1$ nm.[12] The pixel size is 1 nm.

We apply ILT to benchmark pre-RET layouts from the 2013 ICCAD contest [38], using a 2013 production release of a commercial EDA tool. From the ILT solutions, we select ten representative mask shapes for evaluation (see Fig. 4).

For each of the ten target shapes, we run B&P on an eight-core machine with a time limit of 12 h. Half the time limit is devoted to solving the ILP corresponding to the split-shapes, with the time limit of each split-shape $t_{\mathrm{ori},i}$ proportional to the size of its bounding box. The remaining time limit is spent in solving the larger ILP corresponding to t_{ori}.

In any branch and bound based search method for integer programs, the UB corresponds to the best integral solution that has been discovered so far. The LB corresponds to the LP relaxation at a particular level of the branch and bound tree. We report the UB and LB reached by B&P within the set time limit.

The shot count and runtime of the MP heuristic depend strongly on Δ_{MP} and the value by which each shot size is shifted when searching for new shots. Selecting a large value for these parameters reduces the runtime, but typically increases shot count, and can cause many mask shapes to have CD violations even after shot refinement. For this paper, we set $\Delta_{\mathrm{MP}} = 15$ nm, and shots are shifted by $(\Delta_{\mathrm{MP}}/2)$.

Table II shows the shot count, runtime and memory usage for four different heuristics (GSC, MP, GC, and PROTO-EDA). For ten benchmark shapes and three scenarios, our method reports a LB based on LP relaxation.[13] Although this seems trivial, typical LP methods (simplex and barrier methods) run out of memory while trying to solve the LP relaxation of the ILP in (4) for these benchmark shapes. Hence, our B&P-based method appears to be enabling to the computation of this LB. Moreover, for the case with $\sigma = 6.25$ nm, $\gamma = 2$ nm, our method discovers a fracturing solution (UB) better than any of the heuristics for five of ten shapes. For two of them, optimal solution is found.

Some results in bold indicate that the solution has CD violations; since these shot counts are therefore "optimistic," we do not include these results in our suboptimality analysis.[14] If we assume that the LB reported by the ILP is indeed the optimal shot count, the suboptimality of GSC, MP, GC, and PROTO-EDA heuristics ranges, respectively, from 1.7× to 5.6×, 1.6× to 6.0×, 1.3× to 4.7×, and 2.0× to 3.7×, for $\sigma = 6.25$ nm and $\gamma = 2$ nm.

Since the gap between the optimal solution of an ILP and the LP relaxation can be very large, suboptimality analysis based on the LB may be too pessimistic. If we make the optimistic assumption that the integer solutions obtained by the ILP are in fact optimal, i.e., the UB is equal to the optimal shot count, then the suboptimality of the GSC, MP, GC, and PROTO-EDA heuristics could be as large as 3.9×, 4.5×, 2.3×, and 2.6×, respectively. These results suggest that there is significant room for improving the shot count of current mask fracturing solutions. Regarding the runtime, PROTO-EDA always runs faster than the other heuristics. MP takes the longest time to obtain the solutions among four heuristics, but this heuristic finds the better solutions (i.e., smaller number of shot counts) than GSC and PROTO-EDA. Note that we do not include our B&P-based method in this runtime comparison because B&P-based method is not for mask fabrication, but for benchmarking of any mask fracturing heuristics. We also show the peak memory usage: GSC, MP, and GC have essentially the same peak memory usage which is dominated by the manner in which we implement our fast convolution.

Improvements in mask writing tools are likely to reduce the blurring of shot intensity caused by forward scattering since reducing blurring helps improve the resolution of the tool. Consequently, σ in the Gaussian model will decrease. The shot counts of different heuristics, along with the UB or LB, are also shown in Table II, when σ is reduced to 4 nm. The impact of change in σ on the shot count varies for different shapes. For most shapes, the change in shot count of the GSC heuristic is not very large. However, the MP heuristic is highly sensitive to the value of σ, and the shot count increases with $\sigma = 4$ nm for most mask shapes. This is because smaller σ reduces the covering distance range of a shot, which uses more shots in the first phase of MP. Among the four heuristics, GC shows the best solution (i.e., minimum shot count) within a relatively short time (<10 s) in most cases.

In addition to σ, another important factor that can affect the shot count is CD tolerance γ. Tighter CD tolerance will increase the number of constraints that a fracturing solution must follow, leading to higher shot count. More constraints also slows down B&P and increases the gap between the reported UB and LB. This is illustrated in Table II, if we compare the shot counts of the scenarios with $\sigma = 4$ nm, $\gamma = 2$ nm and with $\sigma = 4$ nm, $\gamma = 1$ nm.

V. Benchmark Generation With Known Optimum

Section IV-F shows that ILP-based benchmarking requires considerable computational resources to find LB and UB on the optimal shot count. In this section, we propose a scalable method to evaluate the suboptimality of mask fracturing heuristics by constructing target shapes for which the minimum shot count is known. This benchmark generation method is based on the key observation that there is a set of boundary segments each of which requires at least two shots in *any* fracturing solution. Here, we use B^n to denote the set of all boundary segments such that each boundary segment $b^n \in B^n$ requires at least n shots. For example, the union of green and red lines in Fig. 7(a) is a boundary segment b^2.

We first use exactly two shots to generate a target shape which contains a boundary segment b^2. By the definition of b^2, we need at least two shots to produce the boundary segment. Since we use exactly two shots to generate the target shape, our solution is optimal and the minimum shot count is two.

[12]The minimum shot size constraint accounts for slivers. Although some prior fracturing work mention that aspect ratio of a shot should be constrained, based on our discussion with an EDA vendor, we believe that setting a minimum shot size is a more appropriate constraint for a e-beam mask write tool.

[13]The fractional LP relaxation value is rounded up to the next integer to obtain the LB.

[14]The percentage of pixels that are failing (i.e., number of failing pixels/total number of pixels × 100), over all results, is at most 0.16%.

TABLE II
COMPARISON OF SHOT COUNT, RUNTIME, AND MEMORY USAGE FOR ILT MASK SHAPES SHOWN IN FIG. 4 FOR FOUR DIFFERENT HEURISTICS (GSC, MP, GC, AND PROTO-EDA) ALONG WITH LB AND UB OBTAINED FROM B&P. SHOT COUNT IS SHOWN FOR THREE SCENARIOS WITH DIFFERENT VALUES OF SIGMA (σ) AND CD TOLERANCE (γ). WE SET 12 h TIME LIMIT FOR B&P. BOLD-FONT NUMBERS INDICATE INFEASIBLE SOLUTIONS (AT LEAST ONE FAILING PIXEL IN THE SOLUTION)

Clip-ID	Shot Count (Runtime (s))															Memory usage (MB)
	$\sigma = 6.25nm$, $\gamma = 2nm$					$\sigma = 4nm$, $\gamma = 2nm$					$\sigma = 4nm$, $\gamma = 1nm$					
	GSC	MP	GC	PROTO-EDA	B&P LB/UB	GSC	MP	GC	PROTO-EDA	B&P LB/UB	GSC	MP	GC	PROTO-EDA	B&P LB/UB	GSC, MP and GC
1	14 (< 1)	18 (5)	6 (1)	7 (< 1)	3/4	14 (< 1)	10 (7)	13 (< 1)	12 (< 1)	4/5	22 (< 1)	13 (13)	9 (2)	12 (< 1)	5/9	239
2	18 (3)	13 (8)	13(2)	21 (< 1)	6/13	23 (1)	23 (18)	28 (1)	31 (< 1)	6/9	27 (2)	46 (42)	21 (2)	32 (< 1)	8/15	241
3	5 (1)	4 (18)	4(1)	7 (< 1)	3/3	3 (< 1)	9 (3)	**3 (13)**	27 (< 1)	3/3	13 (< 1)	75 (28)	**7 (6)**	27 (< 1)	3/9	239
4	31 (2)	12 (27)	20(1)	21 (< 1)	6/13	40 (< 1)	25 (19)	**22 (22)**	36 (< 1)	7/17	60 (4)	51 (40)	**35 (17)**	36 (< 1)	10/36	241
5	23 (< 1)	25 (106)	8 (4)	12 (< 1)	5/8	27 (1)	16 (45)	14 (1)	22 (< 1)	5/13	24 (11)	**19 (83)**	**19 (11)**	22 (< 1)	8/23	239
6	9 (< 1)	5 (1)	5 (< 1)	6 (< 1)	3/3	9 (< 1)	5 (1)	5 (< 1)	11 (< 1)	3/4	17 (< 1)	27 (8)	6 (1)	11 (< 1)	4/6	239
7	10 (< 1)	6 (4)	5 (< 1)	8 (< 1)	3/4	15 (< 1)	5 (3)	5 (< 1)	11 (< 1)	3/4	22 (1)	**34 (27)**	**9 (3)**	14 (< 1)	4/7	239
8	26 (< 1)	10 (89)	14 (< 1)	12 (< 1)	5/9	26 (< 1)	16 (51)	16 (< 1)	21 (< 1)	5/15	44 (< 1)	**25 (91)**	25 (< 1)	23 (< 1)	8/20	240
9	39 (3)	25 (25)	18 (14)	26 (< 1)	7/10	39 (< 1)	22 (32)	**13 (14)**	53 (< 1)	4/13	63 (3)	**55 (67)**	**39 (20)**	54 (< 1)	6/30	242
10	14 (< 1)	7 (13)	14 (< 1)	11 (< 1)	3/6	15 (< 1)	10 (8)	20 (< 1)	19 (< 1)	4/5	24 (1)	12 (15)	**12 (33)**	21 (< 1)	6/12	239

To extend the target shape, we add a new shot adjacent to one of the existing shots. We select the location of new shot such that there is a new b^2 in the extended target shape. Note that we only increase the total shot count by one (and reuse an existing shot) to produce the new b^2 which requires two shots. Because the extended target shape cannot be produced by stretching or shifting the shots in the previous solutions (i.e., at least one more shot is required), the solution corresponding to the extended target shape remains optimal regarding shot count.

A. Boundary Segment Analysis

To determine the set of boundary segments which require at least two shots, we analyze the relationship between straight/concave boundary segments and the image produced by a shot. We do not analyze the case of convex boundary segments because the convexity of shot image boundary (i.e., corner rounding) makes it hard to determine the set of convex boundary segments which require at least two shots.

1) Straight Boundary Segment: Since a mask shot must be isothetic, a single mask shot cannot produce a long straight boundary at an angle (θ) which is not a multiple of 90°. Fig. 5 shows a straight boundary segment b_{seg} (black solid line) at an angle θ. The dashed lines parallel to b_{seg} are the inner and outer boundaries. The inner (resp. outer) boundary is obtained by shrinking (resp. expanding) the target boundary toward the inside (resp. outside) of the target shape by the value of γ. To produce the straight boundary using a single shot, we must place a corner of the shot close to b_{seg}. The longest straight boundary covered by the single shot is the length $[L^{\theta}_{\mathrm{lin}}(W, H)]$ between the crossing points (blue cross marks in Fig. 5) of the inner target boundary and the image boundary.[15] To maximize the coverage of a single shot, we must shift the shot and therefore the image boundary to touch the outer boundary as shown in Fig. 5. The shot must not be shifted beyond the outer boundary because $I(x, y, s)$ must be less than ρ for all pixels in P_0.

2) Concave Boundary Segment: Fig. 6(a) shows a concave boundary b_{seg} and its inner ($b_{\mathrm{seg\_in}}$) and outer ($b_{\mathrm{seg\_out}}$) boundaries. For a concave target boundary, the maximum boundary length covered by a single shot is defined by the straight line between the points of intersection between $b_{\mathrm{seg\_in}}$ and the shot image boundary [i.e., the blue cross marks in Fig. 6(a) and (b)]. From the straight line between the points of intersection, we define a "virtual" straight line (b_{vir}) and its inner ($b_{\mathrm{vir\_in}}$) and outer ($b_{\mathrm{vir\_out}}$) boundaries. Note that because of the concavity of b_{seg}, any point along $b_{\mathrm{vir\_in}}$ is always closer than $b_{\mathrm{seg\_in}}$ to

[15] W and H correspond to the width $W(s)$ and height $H(s)$ of the shot s under consideration.

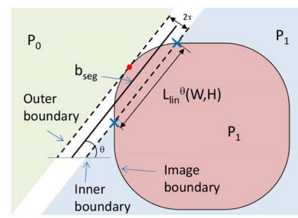
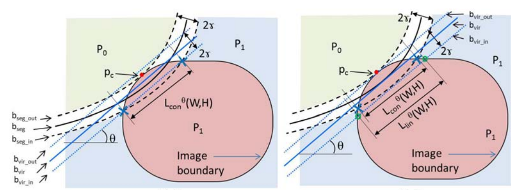


Fig. 5. Definition of the length $L^{\theta}_{\mathrm{lin}}(W, H)$ of a straight-line target boundary covered by a single shot.



(a) (b)

Fig. 6. (a) Definition of the length $L^{\theta}_{\mathrm{con}}(W, H)$ of a concave target boundary covered by a single shot. (b) Comparison of the lengths covered by a single shot for concave versus straight-line target boundaries.

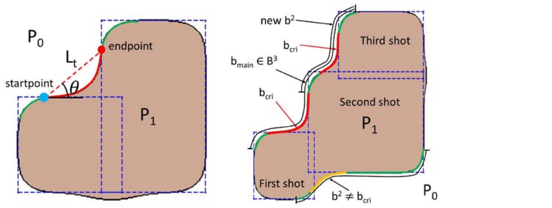
the point that touches the target boundary [i.e., p_c in Fig. 6(a)]. Thus, $b_{\mathrm{vir\_out}}$ is always in P_0, outside the boundary of the shot image. This means that $b_{\mathrm{vir\_out}}$ can be shifted until the $b_{\mathrm{vir\_out}}$ touches the shot image boundary [see Fig. 7(b)] so that we obtain the longest straight boundary $L^{\theta}_{\mathrm{lin}}(W, H)$ covered by the shot image. As a result, the length of the virtual straight line, which is the same as $L^{\theta}_{\mathrm{lin}}(W, H)$ at the same θ, is always larger than the length $L^{\theta}_{\mathrm{con}}(W, H)$ of the concave target boundary.

3) Maximum Length Covered by Shot: As mentioned above, the rounded corner of a single shot image determines the maximum length covered by the shot. As the shot size increases, the corner rounding due to the e-beam proximity effect saturates. As a result, the $L^{\theta}_{\mathrm{lin}}(W, H)$ does not change further with respect to the shot size. Therefore, we can calculate the $L^{\theta}_{\max}$ by increasing W and H iteratively

$$L^{\theta}_{\max} = \max_{s \in S} \left\{ L^{\theta}(W(s), H(s)) \right\}. \tag{6}$$

Since $L^{\theta}_{\mathrm{con}}(W, H) < L^{\theta}_{\mathrm{lin}}(W, H)$ for any shot s, the maximum $L^{\theta}_{\mathrm{lin}}(W, H)$ is an UB on $\max_{s \in S}\{L^{\theta}_{\mathrm{con}}(W(s), H(s))\}$.

Lemma 3: For a mask fracturing problem with finite γ and σ, if a target boundary segment is a straight line or concave shape with length L_t [defined in Fig. 7(a)] larger than



(a) (b)

Fig. 7. (a) L_t is the Euclidean distance between the startpoint and the endpoint on the target boundary, provided that the target boundary from the startpoint to the endpoint is concave or straight line. (b) Example of benchmark generation with three shots. b_{main} is the union of green and red lines and contains two b_{cri}.

$L^{\theta}_{\max}$, more than one mask shot is required to pattern the target boundary segment.[16]

B. Construction of Target Shape

We now describe a systematic method to construct a target shape with known minimum shot count. We first construct a b_{seg} using two shots by placing the second shot to the top right of the first shot as shown in Fig. 7(b). We define the top left boundary [e.g., the union of green and red lines in Fig. 7(b)] as the *main boundary* (b_{main}).[17] By placing the second shot far enough from the first shot, we create a *critical boundary segment* $b_{\text{cri}} \in B^2$, which is part of b_{main}. The b_{cri} is a straight line or a concave segment with length L^{θ} larger than $L^{\theta}_{\max}$ (Lemma 3). Although there can be many boundary segments $\in B^2$, only those overlapping with b_{main} are considered as the critical boundary segments. For example, the yellow boundary segment in Fig. 7(b), while an element of B^2, is not considered to be a b_{cri} because it does not overlap with b_{main}.

Lemma 4: Given a boundary segment b^n of a target with $n-1$ critical boundary segments, and its corresponding shots, we can add a shot to obtain b^{n+1} with an optimal $(n+1)$-shot solution if the addition satisfies the following conditions.

1) Adding a shot does not affect the critical boundary segments of b^n.
2) b_{main} of the new target shape is continuous.
3) There is a b^2 in the b_{main} of the new target shape which cannot be made by extending the shots which produce b^n without altering the critical boundary segments of b^n.

Based on Lemma 4, we add a shot at the top right of the existing target shape. This ensures that we have a continuous b_{main}. Moreover, the top-left coordinate of the newly added shot is selected such that there is a b^2 in the new b_{main}. Since the new b^2 is always at the top of the target shape, it cannot be made by extending previous shots unless the existing critical boundary segments are altered. Also, placing the shot at the top right does not affect the existing critical boundary segments. By adding $n-2$ shots to the target shape generated by two shots, we can obtain a target shape $\in B^n$.

An important property of our method is that the critical boundary segments are defined only by the top-left coordinates of the shots. Therefore, we may freely place the bottom-right coordinates of the shots to create different target shapes as long as they do not affect the critical boundary segments.

[16]Proof of Lemmas 3–6 are given in [5].

[17]b_{main} is at the top left boundary because we place the next shot to the top right of previous shots.

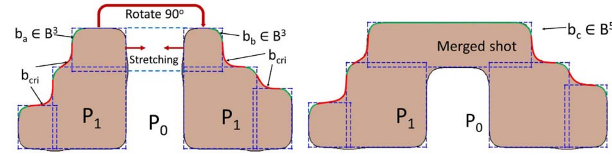


Fig. 8. Example of rotating a target shape for merging.

C. Merging Target Shapes

Lemma 5: Given two target shapes with critical boundary segments $b_a \in B^{n_a}$ and $b_b \in B^{n_b}$, which have, respectively, $n_a - 1$ and $n_b - 1$ critical boundary segments, we can merge b_a and b_b by stretching a shot to create $b_c \in B^{n_a+n_b-1}$ if the following conditions are satisfied.

1) The stretched shot must not alter the critical boundary segments in b_a or b_b.
2) The stretched shot must merge a shot from b_a with a shot from b_b.
3) The nonstretched shots in b_a must be far apart from or misaligned from the nonstretched shots in b_b so that any two nonstretched shots cannot be merged to reduce the number of shots.

The first condition in Lemma 5 imposes a tight constraint on merging the target shapes generated by the method of Section V-B. That is, we can only stretch a shot by moving the lower right corner of the shot in either the rightward and/or downward direction, such that the critical boundary segments are not affected. However, stretching a shot of a target shape to the right and/or down will affect the critical boundary segments on the other target shape. This problem can be solved by rotating the target shapes before merging them.

Lemma 6: A b^n rotated by 90° is still an element of B^n.

Fig. 8 shows an example in which we use Lemmas 5 and 6 to merge a target shape and its rotated copy into a larger and more complex target shape. Using the incremental target boundary extension (Lemma 4) and merging/rotation of optimal target shapes, we can generate a variety of different benchmarks with arbitrary values of optimal shot count.

D. Experimental Results

Using the same experimental setup in Section IV-F, with $\sigma = 6.25$ nm, $\gamma = 2$ nm, we generate two types of target shapes.

1) Arbitrary Generated Benchmarks: We generate five shapes with known optimal shot count using the method of Section V-B. These are shown in [5, Fig. 11(a)].

2) Realistic Generated Benchmarks: Since generated benchmarks can often be unrealistic compared to actual ILT mask shapes, we also generate five mask shapes that look similar to actual ILT shapes with known optimal shot count, again using the method of Section V-B (see [5, Fig. 11(b)]). We manually select shot locations so that the generated benchmarks are similar to actual ILT mask shapes.

We compare the optimal shot count of our generated benchmarks with the shot counts of the comparison heuristics in Table III. For the ten target shapes, the suboptimality ranges from 1.6× to 4.3×, 1.0× to 4.6×, 1.0× to 2.2×, and 1.6× to 2.9× for the GSC, MP, GC, and PROTO-EDA heuristics.[18]

For comparison, we also report the LB and UB obtained from B&P for our generated benchmarks in Table III. The

[18]We have performed the additional experiments for more RGB shapes which are similar to the shapes 6–10 in Fig. 4; these yield qualitatively similar results.

TABLE III
COMPARISON OF SHOT COUNT FOR GENERATED BENCHMARKS WITH KNOWN OPTIMAL SOLUTION. BOLD-FONT NUMBERS INDICATE INFEASIBLE SOLUTIONS THAT HAVE AT LEAST ONE FAILING PIXEL

Clip-ID		Shot count					
		Opt	GSC	MP	GC	PROTO-EDA	Branch and Price LB/UB
AGB	1	3	8	4	5	7	3/3
	2	16	**36**	**16**	**25**	30	11/16
	3	17	**49**	38	**37**	40	5/24
	4	7	26	**9**	7	20	5/7
	5	3	13	4	4	8	3/3
RGB	1	5	12	**7**	6	8	3/5
	2	7	11	15	8	14	5/7
	3	5	12	23	**6**	12	4/5
	4	9	17	13	12	17	6/12
	5	6	21	22	9	14	4/6

results show that for testcases arbitrary generated benchmark (AGB)-{1, 5} and realistic generated benchmark (RGB)-{3, 5}, the B&P method can find the optimal solution, i.e., the UB is equal to the optimal shot count. However, for some shapes, the UB reported by B&P within the set time limit may be very far from the optimal shot count (testcases AGB-2, AGB-4, and RGB-4).

The generated benchmarks are more wavy (i.e., have high-frequency components in the boundary of the target shape) compared to actual ILT shapes. This could make the sub-optimality estimation pessimistic. However, we believe that highlighting scenarios where mask fracturing heuristics perform poorly can contribute to the development of better heuristics.

VI. AUTOMATED BENCHMARK GENERATION

In Section V, we constructed benchmark shapes by placing shots manually. This can be extremely tedious, especially for generating benchmarks similar to real ILT shapes. In this section, we propose an automated benchmark generation (AutoBG) method to generate a benchmark ILT mask shape (t_{gen}) which resembles a given actual shape (t_{ori}), with known optimal shot count. To guarantee that the optimal fracturing solution of t_{gen} is known, AutoBG places shots such that they obey the constraints specified in Section V.

In AutoBG, we first split t_{ori} using the method described in Section IV-C.[19] For each split-shape ($t_{\text{ori},i}$), we generate a separate benchmark shape ($t_{\text{gen},i}$) with known minimum shot count that resembles $t_{\text{ori},i}$. We then apply Lemma 5 to obtain the benchmark shape t_{gen}, which resembles t_{ori}.

To obtain $t_{\text{gen},i}$ from $t_{\text{ori},i}$, we first enumerate several candidate sets of line segments such that each set approximates part of the boundary of $t_{\text{ori},i}$. The candidate set of line segments that is eventually picked becomes the main boundary segment b_{main} of $t_{\text{gen},i}$. Next, we determine the locations of corner points of shots to construct b_{main}. Lastly, for each corner point of a shot used to generate b_{main}, we find the diagonally opposite corner point to minimize the XOR difference between the input and generated shape [$d(t_{\text{ori},i}, t_{\text{gen},i})$]. In the remainder of this section, we describe the details of these steps.

[19]For certain target shapes, we use $L_{\text{th}} < 2\beta$ so as to improve the similarity between the generated benchmark and the target shape. To ensure that the fracturing solutions are still optimal, we check the intensity maps to ensure that the fracturing solutions are still optimal, and we make sure that the boundaries obtained by applying the resist threshold to the intensity of split shapes do not overlap with each other.

Algorithm 4 Cost Function for Selecting a Pair of Vertices as Part of b'_{main}

Procedure: *slopeCDcost(target shape t, two vertices* $v_k(t)$ and $v_l(t)$, θ_{LB}, θ_{UB})
Output: *Cost of the segment formed by vertices* $v_k(t)$ *and* $v_l(t)$

```
θ_(v_k(t),v_l(t)) ← the angle with respect to the x-axis of the segment formed by vertices v_k(t) and v_l(t)
if (θ_LB ≤ θ_(v_k(t),v_l(t)) < θ_UB) then
    cost ← CD violating area of line segment between v_k(t) and v_l(t) (Figure 9)
else
    cost ← +∞
end if
return cost;
```

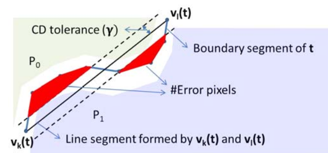


Fig. 9. Number of error pixels along the segment $v_k(t) - v_l(t)$.

A. Finding Candidate Main Boundary Segments

Based on Lemma 4, b_{main} is a continuous boundary segment of the generated mask shape which determines the minimum number of shots required to construct $t_{\text{gen},i}$. Moreover, all the shot corner points used to generate b_{main} must be of the same type (i.e., bottom-left, bottom-right, top-left, or top-right). Given the input split-shape $t_{\text{ori},i}$ (after splitting), to construct the benchmark shape with known optimal shot count $t_{\text{gen},i}$, we need to find a boundary segment that can be used as the main boundary segment to place the optimal shot corners. For that, we enumerate several candidate main boundary segments; each is a set of connected line segments and approximates part of the boundary of $t_{\text{ori},i}$. We then construct the optimal benchmark shape for each candidate main boundary segment, and finally pick the generated shape that is the most similar to the input shape $t_{\text{ori},i}$ as $t_{\text{gen},i}$. Note that the similarity between any two shapes is measured by the area of the region obtained after polygon Boolean XOR operation between the two given shapes.

To find a candidate main boundary segment, we select an ordered sublist of the vertices of $t_{\text{ori},i}$ (V_{main}) such that the line segments obtained after connecting the vertices approximates some boundary segment of $t_{\text{ori},i}$ within the CD tolerance γ. We first define a cost function *slopeCDcost* for selecting two vertices $v_k(t)$ and $v_l(t)$ to be part of V_{main} in Algorithm 4. The cost is equal to the number of the error pixels which are outside the CD tolerance region defined by the line segment connecting $v_k(t)$ and $v_l(t)$, as shown in Fig. 9. To ensure that the set of line segments obtained from V_{main} can be constructed using only one type of shot corner points, all the chosen line segments must have angle with x-axis in the same quadrant. We add an additional constraint to the cost function that sets the cost to infinity if the angle of the line segment with the x-axis is not between the specified LB and UB (θ_{LB} and θ_{UB}). The values of $\theta_{\text{LB}}/\theta_{\text{UB}}$ could be $0°/90°$, $90°/180°$, $180°/270°$, or $270°/360°$ corresponding to upper-left, upper-right, lower-right, and lower-left shot corners.

Based on Sato's [31] dynamic programming method to approximate any given curve by a set of line segments, we propose a similar technique to find a candidate main boundary segment as shown in Algorithm 5. For each vertex of the input shape, we iterate over all the previous vertices in the list to find the vertex with minimum cost (*slopeCDcost* returned

Algorithm 5 Dynamic Programming Algorithm to Obtain a Candidate Main Boundary Segment for Target Shape t

Input : *Vertices* $V(t)$, *target shape* t, CD tolerance γ, θ_{LB}, θ_{UB})
Output: *Ordered sublist of* $V(t)$, V_{main}

1: $k = 1$; $last\_index = 1$; $min\_cost(k) = 0$;
2: **for** $k = 1$ to $|V(t)|$ **do**
3: **for** $l = 1$ to k **do**
4: $cost(l) = slopeCDcost(v_l(t), v_k(t), t, \theta_{LB}, \theta_{UB})$;
5: **end for**
6: $sol\_index(k)$ = the index l which has the minimum cost
7: $min\_cost(k) = cost(sol\_index(k))$
8: **if** ($min\_cost(k) == 0$) **then**
9: $last\_index = k$
10: **end if**
11: **end for**
12: $j = last\_index$; Insert $v_j(t)$ into V_{main}
13: **while** $j \neq 1$ **do**
14: Insert $v_{sol(j)}(t)$ into V_{main}; $j = sol(j)$
15: **end while**
16: return V_{main}

by Algorithm 4) (lines 5–8). This is based on the assumption that $d(t_{\mathrm{ori},i}, t_{\mathrm{gen},i})$ is likely to be smaller when we use a candidate main boundary segment which approximates the boundary of the input shape $t_{\mathrm{ori},i}$ as much as possible with zero cost. In lines 10–12, we store the last vertex with zero cost, and then backtrack from this last vertex to the first vertex to obtain the candidate main boundary segment with zero cost that approximates part of the input shape $t_{\mathrm{ori},i}$ (lines 14–20).

The list of vertices of any polygon, $V(t)$, is cyclical, i.e., any vertex can be used as the starting point. Moreover, the vertices can be ordered in clockwise or anti-clockwise direction. The choice of both the starting vertex and the direction affect the candidate boundary segment that we obtain from Algorithm 5. A poor choice of the starting vertex or direction in determining V_{main} could result in a $t_{\mathrm{gen},i}$ which is not similar to $t_{\mathrm{ori},i}$. To avoid this, we pick different starting vertices and for each consider both the clockwise and anti-clockwise directions to obtain different candidate main boundary segments. We obtain a benchmark shape for each candidate segment, then select the one which is most similar to $t_{\mathrm{ori},i}$. We select the starting vertices for generating candidate V_{main} as follows.

1) If the target shape t_{ori} has no split location, we select the four vertices that are the top, bottom, left and right vertices of t_{ori} as the starting vertices since this will maximize the approximation of the candidate main boundary segment for one quadrant of the boundary of t_{ori}. In other words, starting from these vertices will find the longest candidate main boundary segments.
2) Based on Lemma 5, at least one of the shots of a split-shape will be merged with a shot of another split-shape. We call a shot which will be merged as a "merged-shot." Note that the edges of the merged-shots also define the main boundary. Thus, it is reasonable to select starting vertices around the edges of merged-shots. More specifically, for each splitting edge, we create a virtual merged-shot and stretch the shot along the direction perpendicular to the splitting edge (so that it covers more area and potentially reduces the total number of shots). The starting vertices are defined at the locations where edges of merged-shots intersect with the boundary of $t_{\mathrm{ori},i}$.

B. Determine Corner Points

For each candidate main boundary segment from Algorithm 5, we place shot corner points to construct the boundary segment. The set of shot corner points C_{main}

Algorithm 6 Obtain Shot Corner Points (C_{main}) to Construct a Candidate Main Boundary Segment

Input: *Ordered list of points* V_{main}, *CD tolerance* γ and *the distance between corner point of shot and its image* $\zeta(\theta)$
Output: *Set of shot corner points* C_{main}

1: If $v_{main,1}$ has largest x-coordinate in V_{main} reverse order of V_{main}
2: $V^{sft}_{main} \leftarrow$ Shift every point of V_{main} such that every line segment between consecutive points is shifted by $\zeta(\theta) + \gamma$
3: $C^{l}_{main} \leftarrow$ getCnrPtsFrmSrt(V^{sft}_{main}); $V^{sft,rvr}_{main} \leftarrow$ Reverse order of V^{sft}_{main}
4: $C^{r}_{main} \leftarrow$ getCnrPtsFrmSrt($V^{sft,rvr}_{main}$)
5: Reverse order of C^{r}_{main}
6: **if** ($L(c^{l}_{main,1}, c^{r}_{main,1}) \leq \gamma$) **then**
7: $C_{main} = C^{l}_{main}$
8: **else**
9: Insert $c^{l}_{main,1}$ into C_{main}
10: **for** $i = 2$ to $|C^{l}_{main}| - 1$ **do**
11: $c_{main,i} \leftarrow (c^{l}_{main,i} + c^{r}_{main,i})/2$; Insert $c_{main,i}$ into C_{main}
12: **end for**
13: Insert $c^{r}_{main,|C^{r}_{main}|}$ into C_{main}
14: **end if**
15: return C_{main}

Procedure: *getCnrPtsFrmSrt*(ordered list of points V^{sft}_{main})
Output: *Set of shot corner points* C

1: $V_{samp} \leftarrow$ Sample points on all the line segments obtained from V^{sft}_{main}
2: Insert $v_{samp,1}$ into C; $c_{prev} \leftarrow v_{samp,1}$
3: **for** $i = 2$ to $|V_{samp}|$ **do**
4: **if** ($L(v_{samp,i}, c_{prev}) \geq L^{\theta}_{th}$) **then**
5: Insert $v_{samp,i}$ into C; $c_{prev} \leftarrow v_{samp,i}$
6: **end if**
7: **end for**
8: return C

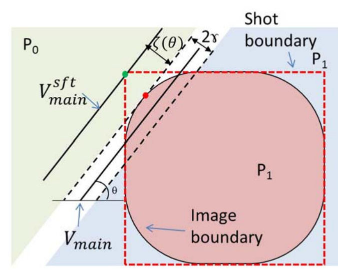


Fig. 10. Gap [$\zeta(\theta)$] between a shot corner point and the line segment with slope θ that is part of boundary segment.

must be placed such that Lemmas 3 and 4 are obeyed, i.e., so that the fracturing solution of the generated benchmark split-shape $t_{\mathrm{gen},i}$ is optimal. Algorithm 6 outlines the steps to find C_{main}.

We first order the points in V_{main} such that they are sorted by x-coordinate (line 1). We then shift the line segments of V_{main} to obtain $V^{\mathrm{sft}}_{\mathrm{main}}$, such that the shot corner points must lie on the line segments obtained by connecting consecutive points of $V^{\mathrm{sft}}_{\mathrm{main}}$ (line 2). This shift compensates for the difference between the rectangular mask shot and its rounded corner due to the e-beam proximity effect, as illustrated in Fig. 10.

Once we obtain the shifted set of points $V^{\mathrm{sft}}_{\mathrm{main}}$ such that all shot corner points lie on the line segments connecting consecutive points from $V^{\mathrm{sft}}_{\mathrm{main}}$, we can find a set of shot corner points using the function *getCnrPtsFrmSrt*(). We first sample the line segments connecting consecutive points from $V^{\mathrm{sft}}_{\mathrm{main}}$ and get all points with integral coordinates that lie on these line segments, V_{samp} [line 1 of *getCnrPtsFrmSrt*()]. We then include the first point of V_{samp} as a shot corner point (line 2).

Next, we iterate over the set of sampled points; if the distance between the previously added shot corner point c_{prev} and the sampled point $v_{\mathrm{samp},i}$ is greater than L^{θ}_{th}, we add the sampled point to the set of shot corner points (lines 4–9). Note that θ is the angle between the x-axis and the line

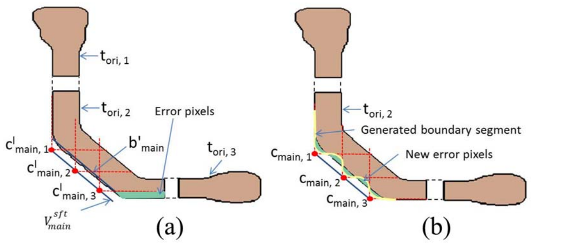


Fig. 11. Example of (a) C^l_{main} and (b) C_{main}.

segment connecting c_{prev} and $v_{\text{samp},i}$. This distance condition ensures the optimality of the fracturing solution of $t_{\text{gen},i}$ by satisfying the conditions of Lemmas 3 and 4.

Obtaining shot corner points from the shifted set of points $V^{\text{sft}}_{\text{main}}$ using the method described in *getCnrPtsFrmSrt*() of Algorithm 6 could lead to large error between the input mask shape and generated shape near the location of the last point of $V^{\text{sft}}_{\text{main}}$. This is illustrated in Fig. 11(a), which shows $V^{\text{sft}}_{\text{main}}$, V_{main}, and $t_{\text{ori},i}$. If the leftmost point of $V^{\text{sft}}_{\text{main}}$ is the first point, then the shot corner points we obtain from *getCnrPtsFrmSrt*() are $c^l_{\text{main},1}$, $c^l_{\text{main},2}$, and $c^l_{\text{main},3}$. Due to the minimum distance constraint between shot corner points imposed by Lemmas 3 and 4, no additional shot corner points can be chosen after $c^l_{\text{main},3}$. As a result, there is significant pixel error after the last shot corner point $c^l_{\text{main},3}$. Moreover, using this method, $t_{\text{gen},2}$ is not mergeable with $t_{\text{gen},3}$ because the shot with $c^l_{\text{main},3}$ violates the second condition of Lemma 5.

To reduce pixel error in generating a mergeable shape, we first obtain two sets of potential shot corner points: 1) C^l_{main} with ordered list of points $V^{\text{sft}}_{\text{main}}$ and 2) C^r_{main} with reverse-ordered list of the same points $V^{\text{sft,rvr}}_{\text{main}}$; these are given as input to *getCnrPtsFrmSrt*() (lines 2–5). If the first corner points of C^l_{main} and the last corner point of C^r_{main} are close to each other ($\leq\gamma$), all points of C^l_{main} and C^r_{main} will be close to each other, and we can use C^l_{main} as the set of shot corner points C_{main} that can construct the line segments formed by V_{main} (lines 6 and 7). However, if the potential shot corner points of C^l_{main} and C^r_{main} are not close to each other, we take the average of the x- and y-coordinates of the corresponding points in C^l_{main} and C^r_{main}. We also include the points with lowest and highest x-coordinates, i.e., the first point of C^l_{main} and that of C^r_{main}. Fig. 11(b) shows the result with this choice of corner points. Although this creates error pixels all along the main boundary, it does not cause any large pixel error after the last corner point and guarantees that $t_{\text{gen},2}$ is mergeable with $t_{\text{gen},3}$.

C. Determine Opposite Corner Points

We now describe the method to determine the locations of diagonally opposite corner points ($C^{\text{opp}}_{\text{main}}$) of C_{main}. Since a shot is determined by $C^{\text{opp}}_{\text{main}}$ and C_{main}, $C^{\text{opp}}_{\text{main}}$ must be placed such that the shot size constraints are obeyed, and the generated shape $t_{\text{gen},i}$ is similar to $t_{\text{ori},i}$.

Algorithm 7 summarizes our method for finding the opposite shot corner points. Given a fixed corner point $c_{\text{main}} \in C_{\text{main}}$, we first enumerate all points which could become the opposite corner point $c^{\text{opp}}_{\text{main}} \in C^{\text{opp}}_{\text{main}}$ in Line 3. If c_{main} is a top-left shot corner, we can find candidate opposite points by considering all the points within distance γ of the boundary of $t_{\text{ori},i}$ that also satisfy the following two conditions.

1) The points lie below and to the right of c_{main}.
2) The distance from c_{main} is such that the corresponding shot will satisfy shot size constraints.

Candidate opposite points for c_{main} when it is a bottom-left, top-left or top-right shot corner can be obtained similarly.

Algorithm 7 Determine Opposite Corner Points for Given Set of Shot Corners

Input : *Shot corner points* C_{main}, *input shape* $t_{ori,i}$
Output: *A set of opposite corner points* C^{opp}_{main}
1: **for all** $c_{main} \in C_{main}$ **do**
2: $maxCover \leftarrow 0$; $C^{can\_opp}_{main} \leftarrow$ Candidate opposite shot points for c_{main}
3: **for all** $c \in C^{can\_opp}_{main}$ **do**
4: $s \leftarrow$ Shot with opposite corners c_{main} and c
5: $cover \leftarrow area(XOR(s, t_{ori,i}))$
6: **if** $cover > maxCover$ **then**
7: $c^{opp}_{main} \leftarrow c$, $maxCover \leftarrow cover$
8: **end if**
9: **end for**
10: Add c^{opp}_{main} to C^{opp}_{main}
11: **end for**

After finding the candidate set of opposite corner points in Algorithm 7, we iterate over this set and find the opposite point for which the corresponding shot best covers the input shape $t_{\text{ori},i}$ (lines 4–10). This opposite point is then inserted to the list of opposite shot corner points (line 11). Once the shot corner points (C_{main}) are known along with the corresponding opposite shot corner points ($C^{\text{opp}}_{\text{main}}$), we can obtain the shape $t_{\text{gen},i}$ by adding the intensity of all the corresponding shots and applying the resist threshold.

TABLE IV
VALIDATION OF AUTOBG METHOD

Clip-ID		Manual	AutoBG		
		Shot count	Shot count	Similarity (%)	Runtime (s)
AGB	1	3	3	87	<1
	2	16	10	87	4
	3	17	16	85	8
	4	7	8	70	2
	5	3	3	87	2
RGB	1	5	3	82	2
	2	7	7	85	4
	3	5	3	90	4
	4	9	8	79	4
	5	6	6	85	4

D. Experimental Results

We first generate shapes by using AGB and RGB shapes as input to our AutoBG in C++. Table IV shows the shot count, runtime and the similarity (area of XOR of input shape and generated shape divided by the area of the input shape). The shapes generated by AutoBG are also shown in Fig. 12.

From Table IV, AutoBG can generate shapes that are $\geq$80% similar to input mask shapes for most cases. The similarity is somewhat less for a few complex shapes such as AGB-4. In addition to similarity, the optimal shot count of the input shapes and the optimal shot count of the AutoBG generated shapes are fairly close,[20] and identical for four cases. This suggests that the optimal shot count of the AutoBG generated shapes for real ILT mask shapes will be close to the unknown optimal shot count of the ILT shapes. The runtime to generate each benchmark shape is less than 8 s.

We also generate benchmark shapes using the 10 mask shapes in Fig. 4 as inputs with different σ and γ. Table V summarizes the suboptimality of the heuristics on these

[20]The shot count of the AutoBG generated shapes is still optimal since the generation process obeys the constraints of Section V.

TABLE V

COMPARISON OF OPTIMAL SHOT COUNT FOR AUTOBG-GENERATED BENCHMARK SHAPES TO THE SHOT COUNTS OBTAINED BY FOUR FRACTURING HEURISTICS. BENCHMARK SHAPES ARE GENERATED FOR THREE SCENARIOS WITH DIFFERENT VALUES OF SIGMA AND CD TOLERANCE. BOLD-FONT NUMBERS INDICATE INFEASIBLE SOLUTIONS (AT LEAST ONE FAILING PIXEL IN THE SOLUTION)

Clip-ID	Shot Count																	
	$\sigma = 6.25nm$, $\gamma = 2nm$						$\sigma = 4nm$, $\gamma = 2nm$						$\sigma = 4nm$, $\gamma = 1nm$					
	Optimal	GSC	MP	GC	PROTO-EDA	B&P LB/UB	Optimal	GSC	MP	GC	PROTO-EDA	B&P LB/UB	Optimal	GSC	MP	GC	PROTO-EDA	B&P LB/UB
1	3	9	5	8	7	3/5	4	7	8	7	13	4/5	6	7	18	7	15	5/7
2	7	21	26	9	16	5/7	7	10	14	11	35	5/8	10	32	35	**11**	28	7/10
3	1	1	9	2	6	1/1	1	1	1	1	8	1/1	3	**3**	**3**	4	13	2/3
4	9	26	17	13	15	6/13	11	21	14	11	27	6/11	13	31	42	**21**	22	9/14
5	6	18	15	**6**	11	4/14	7	16	11	14	19	5/7	10	32	**17**	**18**	23	7/22
6	3	4	10	**4**	6	3/3	3	4	4	5	13	3/3	3	16	15	3	12	3/3
7	4	11	4	4	8	3/4	4	7	6	8	13	3/4	6	15	14	**8**	13	4/6
8	7	24	9	9	9	5/9	9	19	15	**17**	19	5/9	11	33	**17**	**16**	19	8/23
9	9	29	18	11	21	4/11	10	20	39	**12**	29	6/12	14	41	**29**	**26**	34	5/16
10	4	7	5	7	9	4/4	4	9	8	**3**	17	2/5	7	18	21	**8**	19	3/7

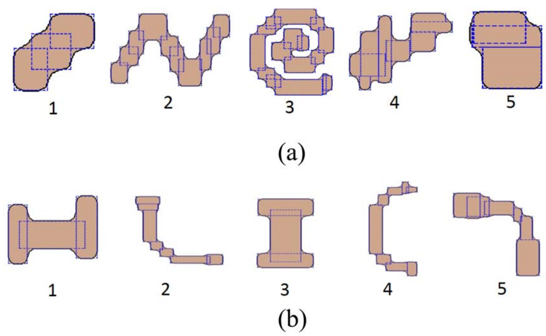


Fig. 12. Illustration of AutoBG generated benchmark shapes. (a) AutoBG generated shapes using AGB shapes of [5] as input. (b) AutoBG generated shapes using RGB shapes of [5] as input.

benchmark shapes. The suboptimality of GSC, MP, GC, and PROTO-EDA across the 10 shapes for the baseline case ($\sigma = 6.25$ nm, $\gamma = 2$ nm) ranges from 1.0× to 3.4×, 1.0× to 9.0×, 1.0× to 2.7×, and 1.3× to 6×. We also report the UB and LB obtained from B&P, which find the optimal solution (i.e., the UB is equal to the optimal shot count) for 16 out of 30 cases.

VII. CONCLUSION

The use of aggressive RET techniques such as ILT, the need for e-beam proximity effect correction, and the use of overlapping shots have transformed mask fracturing into a very challenging computational problem. Although several heuristics have been proposed in the last few years, there has been no systematic study to analyze the quality of solutions. In this paper, we propose two methods to evaluate the suboptimality of mask fracturing heuristics. First, we formulate the mask fracturing problem as an integer linear problem and develop a practical B&P method to generate tight UB and LB on the optimal shot count. Second, we introduce a systematic method to generate a set of benchmarks with known, provably optimal solutions. Further, we describe an automated benchmark generation method to construct shapes which look similar to real ILT shapes.

Furthermore, we evaluate the suboptimality of four mask fracturing heuristics: greedy set cover, MP, graph coloring and a state-of-the-art PROTO-EDA. Our experimental results show that PROTO-EDA has up to 6.0× more shots compared to the optimal solution for generated benchmarks, and has up to 2.6× more shots for ILT mask shapes with unknown optimal solution. These results suggest that there remains considerable opportunity to improve mask fracturing heuristics.

Our source code and benchmark suite are available publicly (http://impact.ee.ucla.edu/maskFracturingBenchmarks). We hope that this will stimulate further research toward development of improved mask fracturing heuristics.

ACKNOWLEDGMENT

The authors would like to thank E. Sahouria of Mentor Graphics for his participation and guidance of this project, and for his co-authorship of [5]. They would also thank P. Ghosh and B. Durvasula of Mentor Graphics for valuable discussions and help in obtaining fracturing results, respectively.

REFERENCES

[1] T. Achterberg, "SCIP: Solving constraint integer programs," *Math. Program. Comput.*, vol. 1, no. 1, pp. 1–41, 2009.

[2] S. Arora, "Approximation schemes for NP-hard geometric optimization problems: A survey," *Math. Program.*, vol. 97, nos. 1–2, pp. 43–69, 2003.

[3] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Oper. Res.*, vol. 46, no. 3, pp. 316–329, 1998.

[4] R. N. Bracewell, "Rectangle function of unit height and base, PI(x)," in *The Fourier Transform and Its Applications*. New York, NY, USA: McGraw-Hill, 1965.

[5] T. B. Chan *et al.*, "Benchmarking of mask fracturing heuristics," in *Proc. ICCAD*, San Jose, CA, USA, 2014, pp. 246–253.

[6] J. Choi, J. S. Pack, I. K. Shin, and C.-U. Jeon, "Inverse e-beam lithography on photomask for computational lithography," *J. Micro Nanolith. MEMS MOEMS*, vol. 13, no. 1, pp. 011003-1–011003-9, 2013.

[7] G. S. Chua *et al.*, "Optimization of mask shot count using MB-MDP and lithography simulation," in *Proc. SPIE PT*, vol. 8166. Monterey, CA, USA, 2011, pp. 816632-1–816632-11.

[8] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, pp. 233–235, 1979.

[9] R. Cinque *et al.*, "Shot count reduction for non-manhattan geometries: Concurrent optimization of data fracture and mask writer design," in *Proc. SPIE PT*, vol. 8880. Monterey, CA, USA, 2013, pp. 88801F-1–88801F-8.

[10] J. C. Culberson and R. A. Reckhow, "Covering polygons is hard," *J. Algorithms*, vol. 17, no. 1, pp. 2–44, 1994.

[11] A. Elayat, T. Lin, E. Sahouria, and S. F. Schulze, "Assessment and comparison of different approaches for mask write time reduction," in *Proc. SPIE PT*, vol. 8166. Monterey, CA, USA, 2011, pp. 816634-1–816634-13.

[12] D. S. Franzblau and D. J. Kleitman, "An algorithm for constructing regions with rectangles: Independence and minimum generating sets for collections of intervals," in *Proc. ACM Theory Comput.*, Washington, DC, USA, 1984, pp. 167–174.

[13] D. S. Franzblau, "Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles," *SIAM J. Discr. Math.*, vol. 2, no. 3, pp. 307–321, 1989.

[14] A. Fujimura, T. Kamikubo, and I. Bork, "Model-based mask data preparation (MB-MDP) and its impact on resist heating," in *Proc. SPIE ALT III*, vol. 7970. San Jose, CA, USA, 2011, pp. 797012-1–797012-10.

[15] A. Fujimura *et al.*, “Best depth of focus on 22-nm logic wafers with less shot count,” in *Proc. SPIE PNGLMT XVII*, vol. 7748. Yokohama, Japan, 2010, Art. no. 77480V.

[16] R. Galler *et al.*, “Modified dose correction strategy for better pattern contrast,” in *Proc. SPIE EML*, vol. 7545. Grenoble, France, 2010, pp. 75450F-1–75450F-12.

[17] P. Gupta, A. B. Kahng, A. Kasibhatla, and P. Sharma, “Eyecharts: Constructive benchmarking of gate sizing heuristics,” in *Proc. DAC*, Anaheim, CA, USA, 2010, pp. 597–602.

[18] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966.

[19] L. Heinrich-Litan and M. E. Lübbecke, “Rectangle covers revisited computationally,” *J. Exp. Algorithmics*, vol. 11, nos. 2–6, pp. 1–21, 2006.

[20] H. Imai and T. Asano, “Efficient algorithms for geometric graph search problems,” *SIAM J. Comput.*, vol. 15, no. 2, pp. 478–494, 1986.

[21] S. Jiang and A. Zakhor, “Application of signal reconstruction techniques to shot count reduction in simulation driven fracturing,” in *Proc. SPIE PT*, vol. 8166. Monterey, CA, USA, 2011, pp. 81660U-1–81660U-14.

[22] S. Jiang and A. Zakhor, “Shot overlap model-based fracturing for edge-based OPC layouts,” in *Proc. SPIE OM XXVII*, vol. 9052. San Jose, CA, USA, 2014, pp. 90520L-1–90520L-19.

[23] A. A. Kagalwalla and P. Gupta, “Effective model-based mask fracturing for mask cost reduction,” in *Proc. DAC*, San Francisco, CA, USA, 2015, pp. 1–6.

[24] A. B. Kahng, X. Xu, and A. Zelikovsky, “Yield- and cost-driven fracturing for variable shaped-beam mask writing,” in *Proc. SPIE PT*, Monterey, CA, USA, 2004, pp. 360–371.

[25] A. B. Kahng, X. Xu, and A. Zelikovsky, “Fast yield driven fracture for variable shaped beam mask writing,” in *Proc. SPIE PNGLMT XIII*, vol. 6283. Yokohama, Japan, 2006, pp. 62832R-1–62832R-10.

[26] T. Lin, E. Sahouria, N. Akkiraju, and S. Schulze, “Reducing shot count through optimization-based fracture,” in *Proc. SPIE PT*, Monterey, CA, USA, 2011, pp. 81660T-1–81660T-13.

[27] X. Ma, S. Jiang, and A. Zakhor, “A cost-driven fracture heuristics to minimize external sliver length,” in *Proc. SPIE OM XXIV*, San Jose, CA, USA, 2011, pp. 79732O-1–79732O-11.

[28] S. G. Mallat and Z. Zhang, “Matching pursuits with time-frequency dictionaries,” *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[29] D. W. Matula, G. Marble, and J. D. Isaacson, “Graph coloring algorithms,” in *Graph Theory and Computing*. New York, NY, USA: Academic Press, 1972.

[30] J. M. Pavkovich, “Proximity effect correction calculations by the integral equation approximate solution method,” *J. Vac. Sci. Technol. B*, vol. 4, no. 1, pp. 159–163, 1986.

[31] Y. Sato, “Piecewise linear approximation of plane curves by perimeter optimization,” *Pattern Recognit.*, vol. 25, no. 12, pp. 1535–1543, 1992.

[32] S. F. Schulze, E. Y. Sahouria, and E. A. Miloslavsky, “High-performance fracturing for variable shaped beam mask writing machines,” in *Proc. SPIE PNGLMT X*, vol. 5130. Yokohama, Japan, 2003, pp. 648–659.

[33] B. Yu, J.-R. Gao, and D. Z. Pan, “L-shape based layout fracturing for e-beam lithography,” in *Proc. ASP-DAC*, Yokohama, Japan, 2013, pp. 249–254.

[34] H. R. Zable, A. Fujimura, T. Komagata, Y. Nakagawa, and J. S. Petersen, “Writing wavy metal 1 shapes on 22-nm logic wafers with less shot count,” in *Proc. SPIE PNGLMT XVII*, vol. 7748. Yokohama, Japan, 2010, pp. 77480X-1–77480X-10.

[35] J. Siek, L. Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley, 2002.

[36] IBM ILOG CPLEX, *V12. 1: User’s Manual for CPLEX*, Int. Bus. Mach. Corporat., Armonk, NY, USA, 2009.

[37] G. Guennebaud and B. Jacob. (2010). *Eigen V3*. [Online]. Available: http://eigen.tuxfamily.org

[38] S. Banerjee, Z. Li, and S. R. Nassif, “ICCAD-2013 CAD contest in mask optimization and benchmark suite,” in *Proc. ICCAD*, San Jose, CA, USA, 2013, pp. 271–274

[39] *OpenAccess API*. Accessed on Jan. 2014. [Online]. Available: http://www.si2.org

[40] L. Dagum and M. Ramesh, “OpenMP: An industry-standard API for shared-memory programming,” *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

Tuck-Boon Chan (S’09) received the M.S. degree from National Taiwan University, Taipei, Taiwan, in 2007, and the Ph.D. degree from the University of California at San Diego, San Diego, CA, USA, in 2014.

He is currently with the Qualcomm Technologies, Inc. San Diego, CA, USA. His current research interests include optimize very large scale integration (VLSI) circuits through design/manufacturing co-optimization, mitigating VLSI circuit variability, and improving manufacturing yield.

Puneet Gupta (SM’16) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi, New Delhi, India, in 2000, and the Ph.D. degree from the University of California at San Diego, San Diego, CA, USA, in 2007.

He is currently a Faculty Member with the Electrical Engineering Department, University of California at Los Angeles, Los Angeles, CA, USA. He Co-Founded Blaze DFM Inc., Sunnyvale, CA, USA, (acquired by Tela Inc.) in 2004 and served as its Product Architect untill 2007. He has authored over 130 papers, 16 U.S. patents, a book and a book chapter. His current research interests include building high-value bridges across application-architecture-implementation-fabrication interfaces for lowered cost and power, increased yield, and improved predictability of integrated circuits and systems.

Dr. Gupta was a recipient of the NSF CAREER Award, the ACM/SIGDA Outstanding New Faculty Award, and SRC Inventor Recognition Award, and the IBM Faculty Award. He currently leads the IMPACT+ Center which focuses on future semi-conductor technologies.

Kwangsoo Han (S’11) received the B.S. and M.S. degrees in electrical engineering from Hanyang University, Seoul, South Korea. He is currently pursuing the Ph.D. degree with the VLSI CAD Laboratory, University of California at San Diego, San Diego, CA, USA.

His current research interests include design for manufacturability and very large scale integration physical design optimization.

Abde Ali Kagalwalla (S’10) received the B.Tech. degree from the Indian Institute of Technology Bombay, Mumbai, India, in 2009, and the Ph.D. degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2014, both in electrical engineering.

He is currently a Software Engineer with the Computational Lithography Group, Intel, Hillsboro, OR, USA. His current research interests include computeraided design of very large-scale integrated circuits, semiconductor design-manufacturing interface, lithography, and algorithms.

Andrew B. Kahng (M’03–SM’07–F’10) received the Ph.D. degree in computer science from the University of California at San Diego (UCSD), San Diego, CA, USA, in 1989.

He was with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA, USA, from 1989 to 2000. Since 2001, he has been with the Jacobs School of Engineering, UCSD. His current research interests include IC physical design, the design-manufacturing interface, combinatorial optimization, and technology roadmapping.