

Effective Model-Based Mask Fracturing for Mask Cost Reduction

Abde Ali Kagalwalla
EE Department
University of California, Los Angeles
abdeali@ucla.edu

Puneet Gupta
EE Department
University of California, Los Angeles
puneet@ee.ucla.edu

ABSTRACT

The use of aggressive resolution enhancement techniques like multiple patterning and inverse lithography (ILT) has led to expensive photomasks. Growing mask write time has been a key reason for the cost increase. Moreover, due to scaling, e-beam proximity effects can no longer be ignored. Model-based mask fracturing has emerged as a useful technique to address these critical challenges by allowing overlapping shots and compensating for proximity effects during fracturing itself. However, it has been shown recently that heuristics for model-based mask fracturing can be suboptimal by more than $1.6\times$ on average for ten real ILT shapes, highlighting the need for better heuristics. In this work, we propose a new model-based mask fracturing method that significantly outperforms all the previously reported heuristics. The number of e-beam shots of our method is 23% less than a state-of-the-art prototype version of capability within a commercial EDA tool for e-beam mask shot decomposition (PROTO-EDA) for ten ILT mask shapes. Moreover, our method has an average runtime of less than 1.4s per shape.

Keywords

Mask Fracturing, Model-based, E-beam, Photomasks, Semiconductor Manufacturing, E-beam Proximity Effect, Graph Coloring, Lithography, Rectilinear Covering, Mask Data Prep, VLSI, DFM, CAD, EDA, MDP, MB-MDP.

1. INTRODUCTION

The persistent delay in extreme ultraviolet lithography (EUVL) has forced the semiconductor industry to continue to use 193nm lithography even at 14nm technology node. Such sub-wavelength scaling has been made possible by the use of several aggressive resolution enhancement techniques (RETs) such as multiple patterning and mask optimization, which includes techniques like optical proximity correction (OPC), inverse lithography (ILT) and sub-resolution assist features (SRAF) to compensate for diffraction.

Although the use of RETs has been a key enabler for semiconductor scaling, it has increased manufacturing cost significantly. In particular, the cost of mask fabrication has increased tremendously. The use of aggressive mask optimization techniques that lead to complex, curvilinear mask shapes is a major cause. In addition, the use of multiple patterning means that more masks are required

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-3520-1/15/06\$15.00.

<http://dx.doi.org/10.1145/2744769.2744828>.

to pattern critical layers. According to ITRS, more than 70% of the cost of ownership of wafers is due to masks [1]. As a result, reducing mask manufacturing cost is extremely important to keep scaling economically sustainable.

Masks are fabricated using a variable shaped electron beam tool. Instead of exposing single pixels, axis-parallel rectangles, commonly referred to as *shots*, are directly exposed. Despite significant improvements in e-beam tool throughput, mask write times are increasing rapidly with each technology node due to the use of aggressive RETs for critical layers, and can take more than two days for critical masks [2]. Mask fracturing is the computational step that is used to get e-beam shots from the mask pattern. Since the number of shots is proportional to mask write time [3, 4], reducing shot count is a key metric for mask fracturing tools. Since mask write is typically around 20% of the mask manufacturing cost [4], a reduction of even 10% in shot count would roughly translate to 2% improvement in mask cost¹. Since the mask set for a single modern design typically costs more than a million dollars, this is a significant cost reduction for semiconductor manufacturing.

The conventional approach to mask fracturing treats it as a geometric partitioning problem for rectilinear polygons. Known to be polynomial time, this problem has been studied extensively. For example, an $O(n^{1.5} \log n)$ time algorithm has been proposed by Imai and Asano to solve this partitioning problem using the minimum number of rectangles [5]. To handle additional mask manufacturing constraints such as reducing slivers (narrow shots), Kahng et al. propose an ILP formulation [6], and a faster heuristic that selects rays from concave corners of mask shapes [7].

However, the growing complexity of mask shapes, coupled with e-beam proximity effect has led mask makers to adopt model-based mask fracturing, which is characterized by two main features:

- *Overlapping shots* are allowed, which can help reduce shot count since there is greater flexibility in placing shots. Consequently, mask fracturing must now be treated as a geometric covering problem, instead of partitioning. Rectilinear covering is known to be NP-hard [8]. Several heuristics with different bounds and constraints on the rectilinear polygon (such as hole-free, vertically or horizontally convex) have been proposed to solve the rectilinear covering problem [9, 10, 11].
- *E-beam proximity effect* must be accounted for during mask fracturing. Caused by forward scattering of electrons, proximity effect has now become comparable to mask feature sizes [12] and must be considered during fracturing.

Several heuristics have been proposed recently to solve the model-based mask fracturing problem. Jiang and Zakhor propose matching pursuit for complex SRAF shapes [13] and a greedy approximation covering method for simpler OPC shapes [14]. Lin et

¹This assumes that mask write time is proportional to cost of mask write, which is reasonable since mask write cost is dominated by e-beam tool depreciation.

Table 1: Glossary of Terminology.

Term	Meaning
S	Set of shots under consideration
s	Rectangular shot under consideration
(x, y)	Coordinate of point under consideration
$R_s(x, y)$	2D Rectangular function of shot s
$G(x, y)$	2D Gaussian kernel function
$I_s(x, y)$	Intensity of shot s at point (x, y)
$I_{tot}(x, y)$	Sum of intensity of all shots in S at (x, y)
Δ_p	Pixel size
$p(x, y)$	Pixel at location (x, y)
P_{on}	Set of pixels inside target mask shape
P_{off}	Set of pixels outside target mask shape
P_{fail}	Set of pixels which violate the constraint specified in Equation 4
γ	CD tolerance for fracturing
L_{min}	Minimum allowed shot size
$(x_{bl}(s), y_{bl}(s))$	Bottom-left coordinate of shot s
$(x_{tr}(s), y_{tr}(s))$	Top-right coordinate of shot s
V_M	Vertices of target mask shape to be fractured
V_M^s	Subset of V_M that approximates boundary of target polygon
v_k	Vertex of mask shape under consideration
C	Set of shot corner points
c_i	Shot corner point under consideration
L_{th}	Longest 45° line segment that can be made using a single shot
$G(V, E)$	Undirected graph with set of vertices V corresponding to shot corner points C , and set of edges E
$G^{inv}(V, E^{inv})$	Inverse graph of $G(V, E)$ with edge between any non-neighboring pair of vertices in $G(V, E)$
$cost_{ref}$	Cost function for iterative shot refinement
N_{max}	Maximum number of shot refinement iterations
N_H	Number of non-improving iterations after which shot addition/removal is done

al. compare several different model-based fracturing methods [15]. Recently, Chan et al. propose a new methodology to benchmark mask fracturing heuristics [16]. The authors have shown that even a state-of-the-art prototype [version of] capability within a commercial EDA tool for e-beam mask shot decomposition (PROTO-EDA) can be suboptimal by up to $3.6\times$. On their website, the authors of the benchmarking work have compared the suboptimality of two heuristics (Greedy set cover (GSC) and matching pursuit (MP)), along with a newer version of PROTO-EDA [17]. In this work, we propose a new mask fracturing method that can perform significantly better than prior art, including GSC, MP and PROTO-EDA. Our method first performs approximate fracturing using a graph coloring based method and then uses a shot refinement step to fix all the CD violations.

The rest of the paper is organized as follows. Section 2 defines the model-based mask fracturing problem. Section 3 describes our graph coloring based approximate fracturing method. Our iterative shot refinement method to fix all the CD violations is described in Section 4. We show the results of our method in Section 5. Finally, we conclude the paper in Section 6. The notation we use in this paper is summarized in Table 1.

2. PROBLEM DESCRIPTION

The goal of model-based mask fracturing is to cover a given mask shape using as few shots as possible while accounting for e-beam proximity effect. For a full-field mask, each shape can be

fractured independently. However, since a mask contains billions of polygons, a practical fracturing method must be extremely fast for any given polygon. In this work, we focus on solving the fixed dose model-based fracturing problem with rectangular shots, as done in the benchmarking work [16]. Although variable-dose shots [18] and non-rectangular shots [19, 20] have been proposed, Elayat et al. have concluded that fixed-dose fracturing with rectangular shots is the most viable approach to reduce shot count without significant changes to the mask writing tools [21].

To model the proximity effect, we first define a simple rectangular function for a particular shot s in Equation 1. Then, we convolve this function with the proximity kernel function (Equation 2) to obtain the intensity of shot s , $I_s(x, y)$ in Equation 3.

$$R_s(x, y) = \begin{cases} 1 & \text{if } x_{bl}(s) \leq x \leq x_{tr}(s) \& y_{bl}(s) \leq y \leq y_{tr}(s) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$G(x, y) = \begin{cases} \frac{1}{\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{\sigma^2}\right) & \text{if } \sqrt{x^2+y^2} \leq 3\sigma \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$I_s(x, y) = G(x, y) \star R_s(x, y) \quad (3)$$

For any given target shape that needs to be fractured, we first sample the shape to get pixels. The set of pixels $p(x, y)$ are partitioned into three sets depending on the shape and the required CD tolerance γ : P_x is the set of pixels which lie within γ of the target shape boundary, P_{on} are the pixels inside the target shape and P_{off} are outside the target shape.

Given a sampled mask shape and e-beam proximity model, the goal of model-based mask fracturing is to find a minimal set of shots S that satisfy the following conditions:

1. The total intensity of all the shots at any pixel $p(x, y)$ must satisfy the constraint defined in Equation 4. If any pixel $p(x, y) \in P_{on} \cup P_{off}$ violates this constraint, we refer to it as a *failing pixel*.

$$\sum_{s \in S} I_s(x, y) = I_{tot}(x, y) = \begin{cases} \geq \rho & \text{if } p(x, y) \in P_{on} \\ < \rho & \text{if } p(x, y) \in P_{off} \end{cases} \quad (4)$$

2. Each shot $s \in S$ must satisfy the minimum shot size constraint, i.e. $x_{tr}(s) - x_{bl}(s) \geq L_{min}$ and $y_{tr}(s) - y_{bl}(s) \geq L_{min}$.

3. GRAPH COLORING BASED APPROXIMATE FRACTURING

The goal of this first step of our approach is to quickly obtain an initial fracturing solution with as few shots as possible. The generated fracturing solution is imprecise, i.e. it may contain CD violations, which are later fixed by shot refinement. We first approximate the boundary of the input mask shape to reduce the number of vertices. Then we determine a set of shot corner points required to cover this approximate boundary of the shape. Using these shot corner points as vertices of a graph, we model the fracturing problem as a graph clique covering problem and then solve this graph problem using a simple sequential coloring heuristic. These steps are described in more detail in this section, and summarized in Figure 1.

First we approximate the boundary of the mask shape using Ramer-Douglas-Peucker algorithm [22]. This method reads in the list of vertices of the mask target shape, V_M , and iteratively determines a subset of the vertices $V_M^s \subset V_M$ such that the perpendicular distance between any vertex $v_k \notin V_M^s$ and the closest line segment connecting two consecutive points in V_M^s is less than an input tolerance, which we set to γ . This approximation, illustrated in Figure 1, simplifies the shape boundary and helps in reducing the shot count at the end of this initial step.

After approximating the shape boundary, we find a set of shot corner points, C , required to construct this approximate boundary.

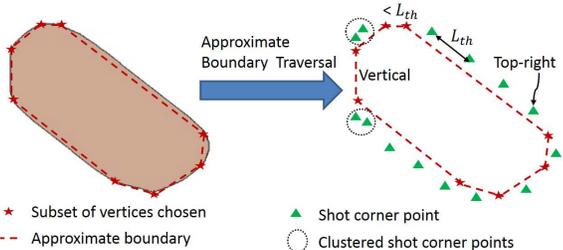


Figure 1: Approximation of the boundary of a mask target shape using Ramer-Douglas-Peucker algorithm (left) followed by shot corner point extraction by traversal of the approximate boundary.

Apart from its location coordinate, each shot corner point is characterized by its type, i.e. whether it corresponds to the bottom-left, bottom-right, top-left or top-right corner of a shot. To find shot corner points, we exploit the fact that non-rectilinear boundary segments can be created by using the corner rounding due to e-beam proximity effect. Based on the proximity model, we define a threshold distance L_{th} that corresponds to the longest 45° line that we can create at the shot corner in a manner similar to [16], as shown in Figure 2. A horizontal or vertical boundary segment, on the other hand, can be written easily by placing the horizontal or vertical edge of a rectangular shot at the segment location.

To obtain the shot corner points, we traverse the list of line segments connecting every pair of consecutive vertices in V_M^S , v_k and v_{k+1} and then add shot corner points using the following criteria:

- If the line segment connecting v_k and v_{k+1} is vertical or horizontal, then this segment should be constructed by a single shot edge. So we place two shot corner points at v_k and v_{k+1} . We then shift these two shot corner points by $L_{th}/\sqrt{2}$ away from the line segment in the same direction (vertical or horizontal) as the line segment connecting v_k and v_{k+1} to account for corner rounding. The corner type for the two shot corner points can be easily determined based on the direction of the boundary segment and its location relative to the target shape. For example, if the segment v_k and v_{k+1} is vertical and it lies to the left of the target shape, then v_k and v_{k+1} must be the bottom-left and top-left shot corners.
- If the line segment is not vertical or horizontal, then it must be constructed by using corner rounding. Hence we place several shot corner points on the line segment, such that the distance between any two shot corner points is equal to L_{th} . We then shift the shot corner points such that they are outside the target shape and the perpendicular distance between the line segment and the points is $L_{th}/\sqrt{2}$. The corner type can be easily determined based on the slope of the segment connecting v_k and v_{k+1} .
- If the distance between v_k and v_{k+1} is less than L_{th} , we ignore this line segment since it can get approximately covered by the shot corner points of the neighboring line segments.

After obtaining the shot corner points by traversing the approximate boundary, as illustrated in Figure 1, we cluster any two shot corner points that have the same type and the distance between the points is less than L_{th} . The set of shot corner points obtained after clustering is the set of shot points C . Next, we need to find the smallest number of shots such that all the shot corner points in C are used at least once. Moreover, this set of shots must construct the target shape as closely as possible to reduce the burden on the subsequent refinement step.

To solve the problem of finding an appropriate set of shots from shot corner points C , we construct a graph $G(V, E)$ with each shot

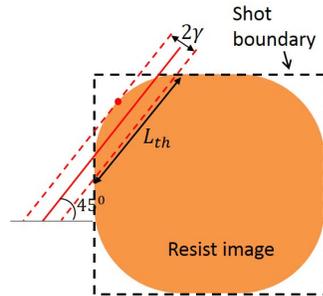


Figure 2: Illustration of corner rounding of a shot due to e-beam proximity effect and definition of L_{th} .

corner point in C as a vertex. There exists an edge between two shot corner points $c_i \in C$ and $c_j \in C$ if and only if the following conditions are met:

- c_i and c_j must be different corner types.
- The test shot formed by using c_i and c_j as corners must satisfy the minimum size criteria and must be such that most of the shot overlaps with the target shape². If c_i and c_j are diagonally opposite corner points, then the test shot is unique. However, if c_i and c_j are non-diagonal, we construct the test shot by extending the shot to the minimum size. For example, if c_i is a bottom-left corner and c_j is top-left, we construct a shot with width equal to the minimum allowed size, and c_i and c_j as the corners of the left edge of the shot.

In the constructed graph $G(V, E)$, every clique corresponds to a shot that could become a part of the fracturing solution. Since the goal of this initial fracturing step is to find the smallest number of shots such that all the shot corner points in C are used, we need to solve the minimum clique partition problem for $G(V, E)$. However, clique partition is a well-known NP-complete problem [23]. Moreover, it can be easily transformed to a graph coloring problem for an inverse graph $G^{inv}(V, E^{inv})$, where an edge exists between two vertices $v_{G,k} \in V$ and $v_{G,l} \in V$ if there is no edge connecting them in $G(V, E)$ [24]. In this work, we color $G^{inv}(V, E^{inv})$ using a simple sequential greedy coloring heuristic [25]. Although better heuristics exist for solving both the clique partition problem directly and graph coloring, we found this fast and simple method to be sufficient to achieve good fracturing solutions. This graph coloring based approximate fracturing is illustrated in Figure 3.

After solving the graph coloring problem of the inverse graph, each color corresponds to one e-beam shot. If the set of shot corner points that have the same color is such that at least a pair of diagonal shot corner points are a part of the set, then the shot is unique and can be easily placed. However, if a particular color is used by just one shot corner point, or if a particular color is used by only two non-diagonal shot corner points, we first place a rectangular shot of minimum allowed width/height using the shot corner points of the color under consideration. We then increase the shot size by shifting the shot edges that do not use the colored shot corner points such that the shot edges touch the opposite boundary of the target shape. This operation is illustrated in Figure 4.

4. ITERATIVE SHOT REFINEMENT

In this step, we take the fracturing solution from graph coloring based approximate fracturing and iteratively modify the fracturing solution to fix all CD violations. However, instead of using the

²More precisely, we used the criteria that more than 80% of the test shot area must overlap with target shape since it gave the best fracturing results.

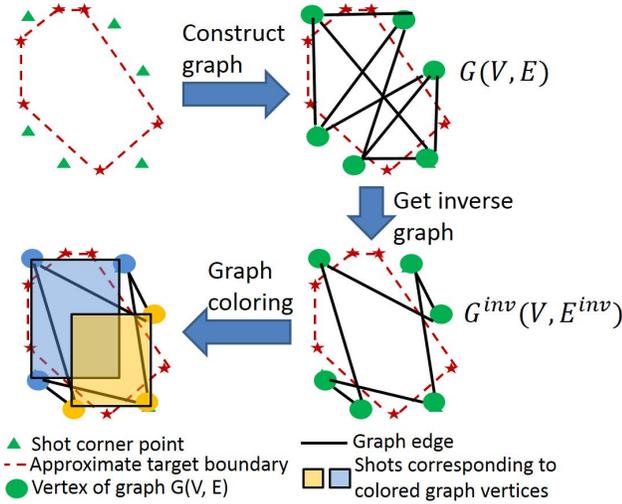


Figure 3: Illustration of steps involved in graph coloring based approximate fracturing.

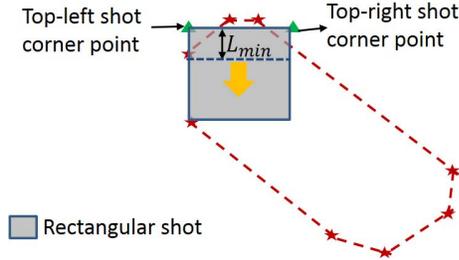


Figure 4: Illustration of a shot formed by same colored top-left and top-right shot corner points. The bottom edge of the minimum height shot (dotted) is extended to touch the lower boundary of target shape.

number of failing pixels as a cost function, we use the sum of intensity gap at failing pixels as the main cost metric that we reduce. This metric, defined in Equation 5, is continuous and is a more sensitive indicator of whether a shot refinement step is useful.

$$cost_{ref} = \sum_{p(x,y) \in P_{fail}} |I_{tot}(x,y) - \rho| \quad (5)$$

We summarize the steps of shot refinement to reduce the cost function of Equation 5 in Algorithm 1. The method runs for up to N_{max} iterations and stops early only if the number of failing pixels reaches zero. In each iteration, we first compute the cost function and find the set of failing pixels in Line 3. We keep track of the fracturing solution with the smallest number of failing pixels so far, and the cost value for the past N_H iterations. If the cost does not improve, we either add or remove shots (Line 5). We add a shot when the number of failing pixels in P_{on} is more than that in P_{off} in Line 7, since adding a shot is likely to resolve violations in pixels inside the target shape. If that is not the case, we remove a shot instead to fix violations in pixels of P_{off} in Line 9. After adding or removing a shot, we use a simple method to merge any two aligned shots. The details of the method used to add a shot, remove a shot or merge shots in covered in Sections 4.3, 4.4 and 4.5 respectively.

Note that the primary method we use to reduce the cost function is shot edge movement (GreedyShotEdgeAdjustment function in Line 13), which we describe further in Section 4.1. If this method is unable to find any shot edge that would reduce the cost function,

we bias all the shot edges (BiasAllShots function in Line 15), which we describe in Section 4.2. Addition, removal or merging of shots is done only when both these simple methods, which do not change the shot count, fail to improve the cost function.

Algorithm 1 Iterative shot refinement algorithm

Input: Mask target shape, Set of shots S that correspond to approximate fracturing solution
Output: Modified set of shots S with fewer CD violations

```

1:  $iter \leftarrow 0$ 
2: while  $iter < N_{max}$  &&  $|P_{fail}| > 0$  do
3:    $(C, P_{fail}) \leftarrow \text{GetFailingPixelsAndCost}(S)$ 
4:   Store fracturing solution with lowest value of  $|P_{fail}|$ 
5:   if  $\Delta C < 10^{-6}$  for previous  $N_H$  iterations then
6:     if  $|P_{fail} \cap P_{on}| > |P_{fail} \cap P_{off}|$  then
7:       AddShot( $S$ )
8:     else
9:       RemoveShot( $S$ )
10:    end if
11:    MergeShots( $S$ )
12:  else
13:    GreedyShotEdgeAdjustment( $S$ )
14:    if no shot edge moved then
15:      BiasAllShots( $S$ )
16:    end if
17:  end if
18:   $iter \leftarrow iter + 1$ 
19: end while

```

4.1 Greedy Shot Edge Adjustment

This is the main method that we use to reduce the cost function and help fix CD violations. For all the four edges of every shot in the current solution set S , we only consider two possible moves. We either move the edge by $+\Delta_p$ or $-\Delta_p$. However, if a particular move makes the shot size less than L_{min} , it is considered invalid and ignored. We then compute the change in cost for the two edge moves and then pick the move with the most reduction in cost. This is done for all the shot edges and we sort all the shot edges such that the edge with the most cost reduction is at the beginning of the list.

We then iterate over the sorted list of shot edges and start accepting the moves which reduce cost. After accepting a particular shot edge move, we do not allow any other edge which is within distance 2σ of the current shot edge to be moved in the current iteration. This blocking step is necessary to avoid cycling, i.e. a set of shot edge moves which cancel out the benefit of each other in subsequent iterations. We use 2σ as the distance for blocking because the intensity of any shot is almost zero ($< 10^{-6}$) at that distance outside a shot.

This shot edge adjustment step is typically the slowest and most time consuming step of our method. In particular, the computation of change in cost for each shot edge move is expensive. To reduce the time taken by this step, we compute the cost incrementally, and only recompute the intensity of the shot corresponding to the shot edge, instead of all the shots in S . Even then, three convolutions are required to compute the cost of the two potential moves for each edge. To speed up the convolution step itself, we use a lookup table based method.

4.2 Bias All Shot Edges

This is a simple operation that can often help the fracturing solution escape from a local minima without changing the number of shots. First we locate all the failing pixels P_{fail} . If the number of failing pixels that belong to P_{on} is more than the number of failing pixels that belong to P_{off} , then we shrink all shot edges, i.e. for every shot $s \in S$ we increment $x_{bl}(s)$ and $y_{bl}(s)$ and decrement $x_{tr}(s)$ and $y_{tr}(s)$ ³. Similarly, if the number of failing pixels that belong to P_{off} is more than P_{on} , we expand all shot edges.

³If the width or height of a particular shot will be less than L_{min} , then the corresponding shot edge is not shrunk.

4.3 Add Shot

Shot addition is done when the number of failing pixels in P_{on} is more than the number in P_{off} , and shot edge adjustment and bisecting fail to resolve all CD violations. To add a new shot, we first merge all the failing pixels in P_{on} using Boolean OR operation to construct polygons. This procedure merges all neighboring failing pixels into a single polygon. Then, we find the bounding box of each such polygon, increase the width or height of the polygon if it is less than L_{min} , and then pick the bounding box that covers the most number of failing pixels in P_{on} . The rectangular shot corresponding to this bounding box is added to the current fracturing solution. Note that only one new shot is added in a particular iteration of shot refinement.

4.4 Remove Shot

To remove a shot $s \in S$, we find the number of failing pixels in $p(x, y) \in P_{off}$ such that the smallest distance between $p(x, y)$ and shot s is less than σ . The shot for which this number is the largest is picked as the one to be removed. The intensity of any shot s , $I_s(x, y)$ is less than 0.5 at a distance of σ . By finding the number of failing pixels which are less than this distance from a shot, we know that removing the shot is likely to resolve the CD violation for those pixels. Note that removing a shot typically leads to many CD violations for pixels in P_{on} which are resolved in later iterations of shot refinement.

4.5 Merge Shots

This step helps to keep the shot count low by finding any two shots that can be merged or combined to reduce the shot count. This routine is necessary to control the shot count during the iterative shot refinement step which is primarily focused on resolving CD violations. For every pair of shots, $s_i \in S$ and $s_j \in S$, we have three criteria for merging shots:

1. If $|x_{bl}(s_i) - x_{bl}(s_j)| \leq \gamma$ and $|x_{tr}(s_i) - x_{tr}(s_j)| \leq \gamma$, or if $|y_{bl}(s_i) - y_{bl}(s_j)| \leq \gamma$ and $|y_{tr}(s_i) - y_{tr}(s_j)| \leq \gamma$, then we can potentially merge the two shots by vertical or horizontal extension, respectively, as illustrated in Figure 5. Note that s_i and s_j are merged only if more than 90% of the shot area lies inside the target mask shape.
2. If the two shots s_i and s_j are such that one of the shots is completely covered by the second shot, then the smaller shot is redundant and can be safely removed.

5. EXPERIMENTAL RESULTS

We have implemented our method using C++ with OpenAccess API for reading/writing mask shapes [26], Boost Polygon Library for polygon Boolean operations [27], Boost Graph Library for graph coloring [28] and Eigen for matrix operations [29]. We compare our method to three different heuristics which have been reported at the benchmarking website [17]; greedy set cover (GSC), matching pursuit (MP) and a commercial EDA tool for e-beam mask shot decomposition (PROTO-EDA). We compiled the implementation of our method and the source code of GSC and MP heuristics (obtained from the authors of [17]) using gcc version 4.6.3 and ran them on a 64-bit Intel i7-3820 CPU. All heuristics are run on a single thread and we report the wall time for all the methods. We demonstrate results for CD tolerance $\gamma = 2nm$, $\sigma = 6.25nm$ and $\Delta_p = 1nm$ as done in the previous work on benchmarking [16]. We shall report results for ten real ILT mask shapes and ten generated benchmark shapes with known optimal solution, all of which are available at the benchmarking website [17]⁴.

The shot count and runtime for the ten real ILT mask shapes provided at [17] are summarized in Table 2. We also report a sum

⁴Shapes are not shown here for the sake of brevity

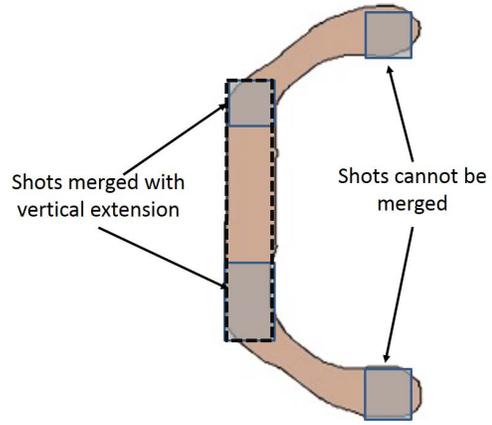


Figure 5: Illustration of two pairs of shots with aligned x-coordinates. In the first case (left), shots can be merged by vertical extension since most of the extended shot lies inside the target shape. The shots cannot be merged in the second case since that would expose too many P_{off} pixels.

of normalized shot count of the ten shapes, where the shot count is normalized with respect to the reported upper bound. With the pessimistic assumption that the upper bound of the benchmarking method is the optimal, the normalized shot count is also the suboptimality ratio. For the normalized shot count, our method is 23% better than PROTO-EDA with comparable average runtime. Compared to MP, our method has 47% lower normalized shot count and 33× faster runtime. Interestingly our method is able to find a solution better than the ILP-based benchmarking method, which ran for 12h on an eight-core machine, for three shapes (Clip-ID 5, 8 and 9). If we just add the number of shots of all the shapes, our method is about 21% better than PROTO-EDA.

In addition to real ILT shapes, we compare the shot count and runtime of our method to prior work for benchmark shapes with known optimal shot count in Table 3. Even for these shapes our method is significantly better in terms of shot count. However, the runtime is significantly higher than GSC and PROTO-EDA methods. This increase in runtime is due to a larger number of iterations during the shot refinement step to fix all CD violations. Moreover, our method fails to find a feasible solution for three shapes, AGB-2, AGB-3 and RGB-3, which have 15, 56 and 8 failing pixels respectively (less than 0.05% of the pixels inside the bounding box of the respective shape). PROTO-EDA is also unable to find feasible solutions for most benchmark shapes and the number of failing pixels is significantly higher (1% – 2% of pixels inside the bounding box of shape) due to different termination criteria. Hence, comparing the shot count for these cases may not be fair. This suggests that the wavy boundary of the complex benchmark shapes, a characteristic not seen in real ILT shapes, makes it challenging for both our method and PROTO-EDA to find feasible solutions in a reasonable time.

6. CONCLUSION

Increasing mask write time has become a key contributor to growing semiconductor manufacturing cost with technology scaling. In this work, we proposed a new model-based mask fracturing method that helps reduce mask write time by reducing the shot count. We first obtained an approximate fracturing solution using a graph coloring based method. Then, we used an iterative shot refinement method that uses a combination of greedy shot edge adjustment, shot addition/removal and shot merging operations to fix CD violations while keeping the shot count low. Using our proposed method, the average suboptimality of our method for ten real mask

Table 2: Comparison of shot count and runtime (in seconds) for real ILT mask shapes for three different heuristics (GSC, MP and PROTO-EDA) with our proposed method. Lower and upper bounds on optimal shot count provided by [17] are also shown here. We do not report the precise runtime for PROTO-EDA since we did not have access to the executable but it is around one second or less on a different machine.

Clip-ID	Shot Count LB/UB	GSC		MP		PROTO-EDA	Our method	
		Shot Count	Runtime	Shot Count	Runtime	Shot Count	Shot Count	Runtime
1	3/4	14	0.5	14	8.0	7	6	1
2	5/9	18	3.0	13	16.0	21	13	1.5
3	3/3	5	1.5	4	4.4	7	4	1.0
4	6/17	31	3.0	14	50.4	21	20	0.5
5	5/13	23	0.5	25	123.2	12	8	3.5
6	3/3	9	0.0	5	2.2	6	5	0
7	3/4	10	0.0	7	4.8	8	5	0
8	5/17	26	0.0	9	157.4	12	14	0.5
9	7/20	39	4.0	14	37.8	26	14	4.5
10	4/8	14	0.5	7	23.2	11	14	0.5
Sum of Normalized Shot Count wrt Upper Bound		21.49		14.54		15.96	12.26	

Table 3: Comparison of shot count and runtime (in seconds) of benchmark mask shapes with known optimal shot count provided in [17] for three previously proposed heuristics (GSC, MP and PROTO-EDA) with our proposed method. Note that PROTO-EDA and our method do have some failing pixels for these shapes.

Clip-ID	Optimal Shot Count	GSC		MP		PROTO-EDA	Our method	
		Shot Count	Runtime	Shot Count	Runtime	Shot Count	Shot Count	Runtime
AGB-1	3	8	0.0	4	1.3	7	5	0.0
AGB-2	16	64	10.0	26	822.4	30	25	32.0
AGB-3	17	52	17.8	35	56.0	40	37	88.5
AGB-4	7	26	2.1	9	80.8	20	7	2.5
AGB-5	3	13	2.1	6	9.6	7	4	0.0
RGB-1	5	12	0.0	19	13.8	8	6	0.0
RGB-2	7	15	3.4	31	29.4	14	8	8.5
RGB-3	5	13	3.0	28	13.3	12	6	19.0
RGB-4	9	45	23.5	19	56.5	17	12	9.5
RGB-5	6	21	4.0	16	8.4	14	9	1.0
Sum of Normalized Shot Count wrt Optimal		10	33.42	26.91		22.31	14.12	

shapes was less than $1.3\times$. Compared to a state-of-the-art prototype version of capability within a commercial EDA tool for e-beam mask shot decomposition, our method has more than 20% lower shot count with similar runtime. The latest version of our source code is available publicly (<http://nanocad.ee.ucla.edu/Main/DownloadForm>).

Acknowledgments

We would like to acknowledge the support of the IMPACT+ program (<http://impact.ee.ucla.edu/>). We thank Emile Sahouria for preliminary discussions of this idea. We deeply thank Prof. Andrew Kahng, Tuck Boon-Chan and Kwangsoo Han from UCSD for the collaborative benchmarking work [16] that helped us tremendously in developing this method.

7. REFERENCES

- [1] "International Technology Roadmap for Semiconductors(ITRS)," <http://www.itrs.net/>, 2009.
- [2] M. Malloy, "2013 mask industry survey," http://public.semtech.org/SEMATECH%20Publications/PDAR242_bpm.pdf, 2013.
- [3] Y. Zhang, R. Gray, S. Chou, B. Rockwell, G. Xiao, H. Kamberian, R. Cottle, A. Wollben, and C. Proglar, "Mask cost analysis via write time estimation," in *Proc. SPIE*, 2005.
- [4] K. Hosono and K. Kato, "PMJ panel discussion overview on mask complexities, cost, and cycle time in 32-nm system LSI generation: conflict or concurrent?" vol. 7122. *SPIE*, 2008.
- [5] H. Imai and T. Asano, "Efficient algorithms for geometric graph search problems," *SIAM J. Comput.*, 1986.
- [6] A. B. Kahng, X. Xu, and A. Zelikovsky, "Yield- and cost-driven fracturing for variable shaped-beam mask writing," in *Proc. SPIE Photomask*, 2004.
- [7] —, "Fast yield driven fracture for variable shaped beam mask writing," 2006.
- [8] J. Culberson and R. Reckhow, "Covering polygons is hard," in *29th Annual Symp. on Foundations of Computer Science*, 1988.
- [9] S. Arora, "Approximation schemes for NP-hard geometric optimization problems: a survey," *Mathematical Programming*, 2003.
- [10] D. S. Franzblau, "Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles," *SIAM J. on Discrete Mathematics*, 1989.
- [11] A. K. Ramesh and H. Ramesh, "Covering rectilinear polygons with axis-parallel rectangles," in *Proc. ACM Symp. Theory of Computing*, 1999.
- [12] D. Tsunoda, M. Shoji, and H. Tsunoe, "Proximity effect correction concerning forward scattering," in *Proc. SPIE Photomask*, 2010.
- [13] S. Jiang and A. Zakhor, "Application of signal reconstruction techniques to shot count reduction in simulation driven fracturing," in *Proc. SPIE Photomask*, 2011.
- [14] —, "Shot overlap model-based fracturing for edge-based OPC layouts," in *Proc. SPIE*, 2014.
- [15] T. Lin, E. Sahouria, N. Akkiraju, and S. Schulze, "Reducing shot count through optimization-based fracture," in *Proc. SPIE*, vol. 8166, 2011.
- [16] T.-B. Chan, P. Gupta, K. Han, A. A. Kagalwalla, A. Kahng, and E. Sahouria, "Benchmarking of Mask Fracturing Heuristics," in *Proc. ICCAD*, 2014.
- [17] "UC Benchmark Suite for Mask Fracturing," <http://impact.ee.ucla.edu/maskFracturingBenchmarks>.
- [18] R. Galler, D. Melzer, M. Boettcher, M. Krueger, M. Suelzle, and C. Wagner, "Modified dose correction strategy for better pattern contrast," in *Proc. SPIE*, vol. 7545, 2010.
- [19] "Writing wavy metal 1 shapes on 22-nm logic wafers with less shot count," in *Proc. SPIE Photomask*.
- [20] B. Yu, J.-R. Gao, and D. Pan, "L-shape based layout fracturing for e-beam lithography," in *Proc. ASP-DAC*, 2013.
- [21] A. Elayat, T. Lin, E. Sahouria, and S. F. Schulze, "Assessment and comparison of different approaches for mask write time reduction," in *Proc. SPIE Photomask*, vol. 8166, 2011.
- [22] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, 1972.
- [23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Springer US, 1972, pp. 85–103.
- [24] J. Bhasker and T. Samad, "The clique-partitioning problem," *Computers & Mathematics with Applications*, vol. 22, 1991.
- [25] D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," *Graph theory and computing*, 1972.
- [26] "Openaccess API," <http://www.si2.org/>.
- [27] "Boost Polygon Library," http://www.boost.org/doc/libs/1_55_0/libs/polygon/doc/index.htm.
- [28] "Boost Graph Library," http://www.boost.org/doc/libs/1_55_0/libs/graph/doc/.
- [29] G. Guennebaud et al., "Eigen v3," <http://www.eigen.tuxfamily.org>, 2010.