Optimization for Multiple Patterning Lithography with Cutting Process and Beyond

Jian Kuang and Evangeline F.Y. Young
Department of Computer Science and Engineering, The Chinese University of Hong Kong
{jkuang, fyyoung}@cse.cuhk.edu.hk

Abstract—**Multiple Patterning Lithography (MPL) is indispensable for producing sub-22nm devices. Recently, multiple patterning with cutting (MPC) was proposed. For example, in triple patterning with cutting (LELECUT), the first two masks are used to do double patterning, whereas the third mask is used to cut off the unwanted parts. In this paper, we will systematically study the problem of cut candidate generation, and propose a flow to optimally minimize the manufacturing cost for standard cell based design with MPC. We will further extend the optimization flow to handle multiple patterning with e-beam cuts. Experiments demonstrate the effectiveness of the proposed algorithms.**

I. Introduction

Multiple Patterning Lithography (MPL) [1]–[5] is indispensable for producing sub-22nm devices. Recently, Multiple Patterning with Cutting (MPC) was proposed [6]–[9]. Triple patterning with cutting process (LELECUT), namely Litho-Etch-Litho-Etch-Cut, was firstly proposed in [6] . Its three masks work as follows: the first two masks are used to do double patterning and the third cut mask is used to cut off the unwanted parts. An example is shown in Fig. 1.

The advantages of MPC over conventional MPL are mainly two folds. **(i)** MPC is more flexible for layout decomposition, especially when there is native conflict for conventional MPL. For example, in triple patterning lithography, as pointed out in [10], a K4 composed of four points from different features causes a native conflict, so the features in Fig. 1(a) are not decomposable even with stitches because the four red points form a K4. However these features can be easily decomposed with MPC. **(ii)** MPC needs fewer stitches than conventional MPL. As overlay errors with stitches will cause extremely high manufacturing cost and yield lost in advanced technologies, some leading foundries even prohibit stitching. Thus, MPC has the potential to reduce the manufacturing cost compared with conventional MPL. Note that cutting process was also proposed for Self-Aligned Double Patterning (SADP) [11], [12]. However, MPC is fundamentally different from SADP.

Layout decomposition is the most critical problem for any type of MPL. For layout decomposition for MPC, the authors of [7], [8] proposed an ILP formulation to minimize the stitch number and conflict number between the first two masks in LELECUT, but ILP has poor scalability. Thus, positive semidefinite relaxation and random rounding were proposed in [9], but the relaxation will lose optimality and even with relaxation, the solving process is still not much faster than the original ILP. Besides, during mask assignment, the cut candidates that may be used to resolve conflicts should be generated and selected properly. The existing cut generation method is inefficient and ignores many practical issues as discussed later.

In this paper, we will first systematically study the problem of cut candidate generation, and then propose a comprehensive flow to optimally minimize the manufacturing cost for MPC

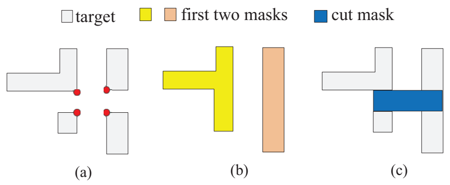


Fig. 1. Illustration for LELECUT process. (a) The target layout. (b) Double patterning with the first two masks. (c) Unwanted part is removed by cut masks.

layout decomposition of standard cell based design in polynomial time with the generated cut candidates. As standard cell based design is one of the most popular design styles, the proposed flow will be useful in practice.

Furthermore, we will extend the flow to optimize layout decomposition for Multiple Patterning with E-beam Cuts (MPEC). It is widely believed that hybrid lithography is among the most promising solutions to handle IC manufacturability with extreme scaling [12], [13]. Optical lithography has high speed but low accuracy. E-beam lithography has high resolution but low throughput. By combining the two, more powerful manufacturing capability can be achieved.

The major contributions of this paper are as follows. **(1)** We present a systematic approach for cut candidates generation that takes practical issues into account. **(2)** We propose a method to optimally minimize the manufacturing cost for standard cell based design with MPC. Its effectiveness and efficiency are verified by experiments. **(3)** We extend the method to MPEC to minimize the writing time of the e-beam system.

The rest of the paper is organized as follows. Section II introduces our optimization methods for MPC. Section III discusses the optimization for MPEC. Section IV reports experimental results and Section V concludes this paper.

II. Optimization for MPC

A. Preliminaries

In the problem of layout decomposition for MPC with K masks, we are given a 2D layout of features in the shape of rectilinear polygons, and K masks, namely $K-1$ "positive" masks and one cut mask ("negative" mask). We first construct a conflict graph (Fig. 2(a)) in which a node represents a polygon and an edge exists between two nodes if the distance between the two corresponding polygons is smaller than the minimum colorable space cs_{min}, i.e., there is a conflict between them.

For each pair of conflicting polygons, it is possible to merge them first and then remove the unwanted part by a cut. Cut candidates need to be generated before using them to resolve conflicts. For example, in Fig. 2(b), three cut candidates are generated, namely c_1, c_2 and c_3. In this paper, using cut c_1 implies that features a and b are merged as specified by c_1. It can be seen that cut masks must cover all the unwanted patterns to create the target layout correctly. It is also assumed

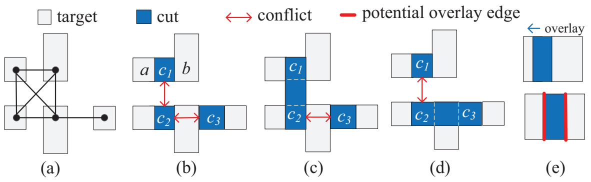


Fig. 2. (a) Target features and the conflict graph. (b) Cut candidates. (c) c_1 and c_2 are mergeable. (d) c_2 and c_3 are not mergeable. (e) Potential overlay edges of a cut.

that cut masks can cover area without any pattern (Fig. 1(c)), but cut masks cannot cover area with target patterns. As all the cuts are assumed to be manufactured by a single exposure-etch process, there will also be a conflict between two cuts if the distance in between is smaller than a threshold c_{min}. Following previous work [8], it is assumed that $c_{min} = cs_{min}$. For example, c_2 conflicts with c_1 and c_3 as shown in Fig. 2(b). Two conflicting cuts are $mergeable$ if they can be merged together without violating any design rules, e.g., c_1 and c_2 are mergeable (Fig. 2(c)) while c_2 and c_3 cannot be merged because otherwise the feature between them will be affected (Fig. 2(d)). Similarly, a conflict graph for the cut candidates can be built. Cut generation will be elaborated in Section II-B.

An obvious objective of layout decomposition for MPC is to minimize the number of *unresolved conflicts* (conflicts that still exist in a decomposition solution). Besides, stitches may be inserted into a feature to slice it into two or more pieces, so that they can be manufactured by different masks [1]. The number of stitches should also be minimized. A shortcoming of the formulations in [8], [9] is that they can only find decomposition solutions with unresolved conflicts between positive masks, which limits the solution space. In our formulation, we further allow unresolved conflicts between cuts in a solution to explore larger solution space.

The work in [8] has no consideration about the cost of cuts and the work in [9] only tries to minimize the number of cuts. To formulate the manufacturing cost of cuts more accurately, we consider the possible overlay errors induced by cuts. Notice that the overlay errors of cuts can be very severe because the features produced may be seriously distorted. Here we formulate the cost of a cut as the total length of the potential overlay edges induced by the cut, i.e., overlay length (Fig. 2(e)).

The problem of layout decomposition for MPC can be formulated as follows:

Problem 1. Given a layout, a minimum colorable spacing cs_{min}, $K-1$ positive masks and one cut mask, assign the features to one of the $K-1$ masks, generate and select cuts, such that the weighted sum of the cost of the cuts and the number of unsolved conflicts are minimized, i.e.,

$$min.\ \sum(\alpha \cdot cost_{cut} + \beta \cdot cost_{conflict})$$

More generally, when stitch is allowed,

$$min.\ \sum(\alpha \cdot cost_{cut} + \beta \cdot cost_{conflict} + \gamma \cdot cost_{stitch})$$

In the formulae, α, β and γ are all user defined parameters.

B. Cut Generation

First, some terminologies are introduced here. A *cut rectangle* is defined as a rectangle in the cut mask. A *cut* is defined as the union of a set of rectangles that enable the merging of two conflicting features and remove the unwanted part. Cut generation is to decide the layout of the cuts for every pair of conflicting features if possible.

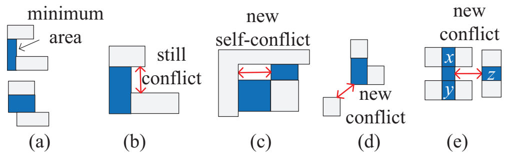


Fig. 3. Issues of cut generation. (a) The method in [8] always selects the cut with the minimum area. (b) An example from [8] shows that the conflict still exists even with the cut (Issue 1). (c) New self-conflict (Issue 2). (d) Merging two features introduces a new conflict (Issue 3). (e) Merging two cuts causes a new conflict (Issue 4).

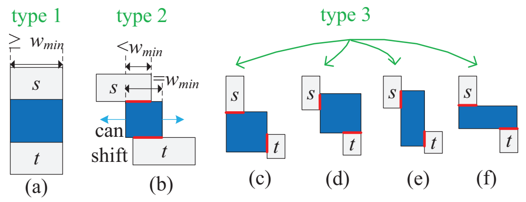


Fig. 4. Different types of conflicts between two rectangles.

To generate cuts between two polygons, previous work [8] exhausts cuts between every pair of edges of the polygons. If there is no overlapping between the cuts, all the cuts will be kept. Otherwise the cut with the minimum area will be taken (Fig. 3(a)). Besides the low efficiency, this method ignores many critical issues in practice as follows:

• **Issue 1.** It selects a cut out of some overlapped cuts. It is possible that the selected cut cannot resolve the conflict. An example taken from [8] directly is shown in Fig. 3(b).

• **Issue 2.** A cut may introduce new self-conflict (a conflict between different parts of the same feature), as shown in Fig. 3(c), which is a self-conflict inside the merged feature.

• **Issue 3.** Merging is not free. Using a cut and merging two features may cause new conflict between the merged feature and some other features, as shown in Fig. 3(d).

• **Issue 4.** Two conflicting cuts can be merged (Fig. 2). However, similar to Issue 3, merging two cuts may introduce new conflict between the merged cut and some other cuts (Fig. 3(e)).

• **Issue 5.** A pair of conflicting features may have multiple choices of cuts. These cuts may have different cost and different conflicting relationships with other cuts. It is non-trivial to select properly (e.g., multiple choices exist for a cut in Fig. 4(c)-(f)).

To take these issues into consideration, Algorithm 1 is proposed. It is a rectangle-based approach. For a pair of conflicting rectangles s and t, there are three types of conflicts, and different cut rectangles will be generated for each type.

Type 1 is that s overlaps with t in one direction and the overlapping length is not smaller than the minimum width w_{min} of the cut mask (Figure 4(a)). For this type of conflict, obviously the optimal cut rectangle is as shown in Figure 4(a).

Type 2 is that s overlaps with t in one direction but the overlapping length is smaller than w_{min} as shown in Figure 4(b). In this case, we will choose a cut rectangle with width being w_{min}, as larger width will potentially cause longer overlay length. However, different cut rectangles with the same width and cost can be obtained by shifting left or right as shown in Figure 4(b).

Type 3 is that s does not overlap with t in any direction, which is the most complicated case. Four different cut rectangles as in Fig. 4(c)-(f) will be considered. To control overlay error, the shapes of the cut rectangles are set in such a way

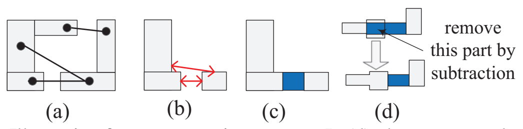


Fig. 5. Illustration for cut generation process. In (d), the cut rectangle overlaps with one of the target features and the overlapped part will be removed.

Algorithm 1 Cut Generation

```
1: Input: two polygons of features $a$ and $b$
2: Output: cuts between $a$ and $b$
3: $ra$=decomposeRectangle($a$), $rb$=decomposeRectangle($b$)
4: Construct a conflict graph $G$ between rectangles in $ra$ and $rb$
5: Find independent components of $G$ by depth-first search
6: for each independent component $C$ of $G$ do
7:     while there is Type 1 or Type 2 conflict in $C$ do
8:         Add a cut rectangle $cr$ between each pair of rectangles with Type 1
           or Type 2 conflict
9:         Check whether $cr$ conflicts with some other rectangles in $C$ (new
           conflicts as in Issue 2)
10:        if new conflicts are found then
11:            Add the new conflicts to $C$
12:        end if
13:    end while
14:    if every pair of conflicting rectangles in $C$ form a polygon that is
       both X-monotone and Y-monotone then
15:        Continue to process the next component
16:    else
17:        Process remaining conflicts similar to lines 7-15, but this time,
           process Type 3 conflict instead of Type 1 or 2 conflict
18:    end if
19: end for
20: Take the union of all cut rectangles
21: Subtract the target layout of $a$ and $b$
22: Return the resulted cuts
```

that the overlay lengths of s and t (i.e., the lengths of the read lines in the figures) are both w_{min}.

It is obvious that all conflicts can be divided into Type 1-3. For both Type 2 and Type 3 conflict, there may be multiple available cuts, as described in Issue 5.

As shown in Algorithm 1, a conflict graph between rectangles are first constructed and divided into independent components, e.g., there are two components in Fig. 5(a). Each component can then be processed separately. Type 1 and Type 2 conflicts are processed first (line 7) because solving a conflict of Type 1 or Type 2 is likely to solve multiple conflicts between rectangles at the same time. For instance, in Fig. 5 (b) there are two conflicts, one is of Type 1 and the other is of Type 3. After solving the Type 1 conflict, the Type 3 conflict is also solved (Fig. 5 (c)). To test whether all conflicts have been solved, we can check whether every pair of conflicting rectangles form a polygon that is both X-monotone and Y-monotone after adding cuts (line 14). A polygon is X-monotone (Y-monotone) if any vertical (horizontal) line intersects with its edges at most twice except at the endpoints [5]. If a cut rectangle overlaps with the two input target features, we will do a subtraction (line 21) as shown in Fig. 5(d) to remove the overlapped part. In order to be litho-friendly, we will verify that every cut rectangle returned by Algorithm 1 does not violate the rule of minimum feature size.

We then introduce how we can tackle the practical issues 1-5 by our algorithm.

• **Solution to Issue 1**: The checking in line 14 makes sure that all original conflicts have been solved.

• **Solution to Issue 2**: The checking in line 9 makes sure that new conflicts are not missed.

• **Solution to Issue 3**: Merging two features may cause new conflicts with other features. Thus, if the distance between a cut and some other feature is smaller than cs_{min}, we will record it as a conflict. This additional conflict information will be kept when generating cuts and be used later in Section II-C2(5).

• **Solution to Issue 4**: Merging two cuts may cause new conflicts, e.g., in Fig. 3(e), there will be a conflict only when the three cuts x, y and z are used at the same time. In this case, the additional conflict information will be kept and used in Section II-C2(5) so as to avoid x, y and z being used simultaneously.

• **Solution to Issue 5**: There may be multiple choices for a cut rectangle. **Type 1** conflict has only one optimal solution. For **Type 2** conflict, by default, we simply choose the one that induces the same overlay length to s and t as shown in Fig. 4(b) (i.e., the two red lines have the same length). However, if such a cut rectangle will cause new conflict, e.g., the cut rectangle conflicts with a feature other than s and t, denoted as r, we will shift the cut rectangle away from r and try to make it not conflict with r (subject to design rules). If the shifting cannot eliminate the conflict, the default cut rectangle will be used. For **Type 3** conflict, one cut rectangle r_1 is called inferior to another rectangle r_2 if they conflict with the same features and the cost of the overlay of r_1 is larger than that of r_2. All cut rectangles that are not inferior to any other cut rectangle will be recorded and used later.

C. Optimal Solution for Standard Cell Based Design

In this section, a method that can optimally minimize the manufacturing cost for standard cell based design with MPC will be presented. We first assume $K = 3$ to illustrate the idea.

1) Solution Graph:

In standard cell based design, cells are placed in rows, and adjacent rows are separated by power/ground rails. So, each row can be decomposed independently [3], [10]. Ref. [3] provided a method to decompose a row for TPL in polynomial time, in which a row is divided into sets according to the slicing lines at the left boundaries of the polygons, e.g., in Fig. 6(a), there are two sets. Note that c appears in both sets. All decomposition solutions for each set are then enumerated. A solution graph is built in which each node is a decomposition solution for a set, and two nodes are connected if they are compatible (see Fig. 6(b), where 1, 2 and 3 denote different masks). Any path from the virtual source s to the virtual sink t represents a decomposition solution of the row.

2) Extensions on Solution Graph:

However, the method described above cannot be applied to MPC decomposition directly. The major reason is that there are cuts in MPC and conflicting cuts may appear in non-adjacent sets. A simple example in Fig. 6(c) shows that cut c_1 conflicts with c_2, thus a path in the solution graph may be an illegal decomposition because the graph can only ensure the compatibility between adjacent nodes. To handle these issues, the following extensions are proposed to construct a correct solution graph for MPC.

(1) Dividing the Features into Sets: Instead of using only one slicing line to define a set as in Fig. 6(a), we use two slicing lines to define a set such that the features lying between two neighboring lines and the features being cut by either line

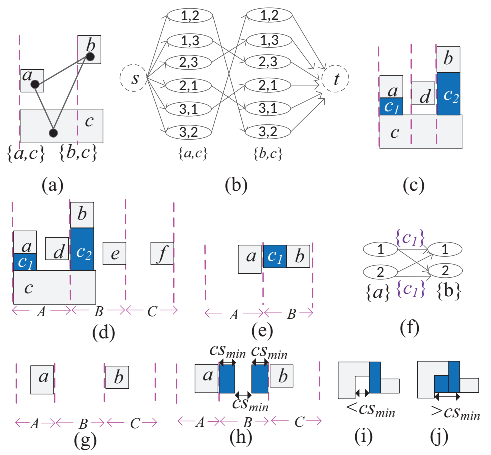
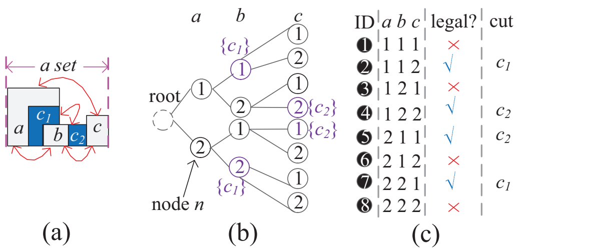


Fig. 6. Illustration for solution graph construction.



ID	a b c	legal?	cut
❶	1 1 1	×	
❷	1 1 2	√	c_1
❸	1 2 1	×	
❹	1 2 2	√	c_2
❺	2 1 1	√	c_2
❻	2 1 2	×	
❼	2 2 1	√	c_1
❽	2 2 2	×	

Fig. 7. Illustration for solution enumeration. (a) A set of features. (b) The tree for solution enumeration. (c) All solutions for the tree in (b), disallowing unresolved conflicts.

will be in the same set. The lines will be placed in such a way that they are always coincident with the left or right boundary of a polygon, because otherwise more features will appear in multiple sets and the solution enumeration of the sets will be slower. The slicing lines are spaced such that conflicting cuts will only appear in adjacent sets, e.g., unlike the situation in Fig. 6(c), c_1 and c_2 appear in adjacent sets in Fig. 6(d). More details on the separation of slicing lines will be discussed later.

(2) Enumeration of Solutions for a Set: We then enumerate the MPC decomposition solutions for each set. In TPL decomposition, we only enumerate the assignment of masks to the features in a set such that there is no conflict within a set. However, in MPC decomposition, solution enumeration should consider using cuts to resolve conflicts and avoiding conflicts between cuts. A tree structure is used to help the enumeration. For instance, to enumerate the solutions for a set with 3 features as in Fig. 7(a), a tree with a virtual root is built level by level as in Fig. 7(b). When $K = 3$, each node has K-1=2 descendants in normal cases (exceptional cases will be discussed later). A node in the tree represents a mask assignment for a feature and is associated with a set of cuts. The mask for a is either 1 or 2. Consider the mask for b, if its mask is the same as the mask of a, cut c_1 should be included. Consider the mask for c, we need to check all its ancestors. For example, solution 1 is illegal, because of the following. (i) There will be a conflict between a and c and there is no available cut to resolve this conflict. (ii) The conflict between b and c needs c_2, but c_2 conflicts with c_1 that is stored in its parent. At each level, if a solution is found illegal, it will not have descendants anymore. A path from a leaf node to the root may represent one legal solution for a set. In this example, 4 solutions are found, namely solution 2, 4, 5 and 7 (Fig. 7(c)) and each solution is associated with one cut.

(3) Edges in the Solution Graph: In a solution graph for TPL decomposition, two solution nodes are compatible (and thus are connected by an edge) when the following requirements are satisfied. **(a)** If two sets contain the same feature, the mask for this feature in these two nodes is the same. **(b)** If two sets contain some conflicting features, the masks for these features in these two nodes are different. In MPC decomposition, requirement (a) should be kept. While for requirement (b), as cuts might be available, the situation is different. For instance, when we consider connecting a node of set A and a node of set B as in Fig. 6(e), if the mask of a is the same as that of b, we may use the cut c_1. So, part of the solution graph is as shown in Fig. 6(f). It can be seen that each edge in the solution graph may be associated with a set of cuts. A node n_1 of set A and a node n_2 of set B can be connected by an edge e only when the above requirement (a) and the following requirements are all satisfied: **(i)** There is no conflict between the cuts in the edge connecting n_1 and n_2. **(ii)** There is no conflict between the cuts in e and the cuts in n_1 and n_2. **(iii)** There is no conflict between the cuts in n_1 and the cuts in n_2. **(iv)** If n_1 and n_2 contain two conflicting features a and b, either the masks for a and b are different or a cut between a and b exists in the edge e.

(4) Separation of Slicing Lines: After constructing a solution graph as described above, it is desirable to have each path in the solution graph representing a legal solution. To achieve this, we need to make sure that the distance between any two slicing lines is large enough such that two non-adjacent sets are independent. Here "independent" means that a feature a completely in set A should not affect the mask assignment of another feature b completely in another non-adjacent set C in any way as shown in Fig. 6(g). To find this distance, we assume the extreme case that both features a and b abut with the boundaries of set B and we try to compute the required width of set B such that feature a and feature b will not affect each other. A natural thought is as follows. First, a safe distance between two cuts is cs_{min}. The width of a cut between two features should not exceed cs_{min} as otherwise the two features will be far away enough and the cut is not needed. Thus the total distance between two slicing lines should be at least $3 \cdot cs_{min}$. An example is shown in Fig. 6(h). However, this ignores Issue 2 as mentioned above. For instance, as in Fig. 6(i), after adding a cut rectangle, there is a new self-conflict. One more cut rectangle is needed, and thus the width of the cut $> cs_{min}$ as shown in Fig. 6(j). To solve this problem, after generating all the cuts, we record the width of the widest cut wc_{max}, and require the minimum distance between two slicing lines to be $2 \cdot wc_{max} + cs_{min}$.

(5) Considering Issues on Cut Generation: Now we discuss how Issues 1-5 mentioned in Section II-B will be avoided in our solution graph construction. Issues 1-2 have been taken care of when generating cuts. When enumerating solutions in a set and connecting two nodes of neighboring sets, we will avoid two types of additional conflicts, i.e., conflicts between cuts and features as in Issue 3, and conflicting relationships between three or more cuts as in Issue 4 (there will be a conflict only if all of them appear at the same time). For Issue 5, we have recorded multiple solutions for a cut. So when enumerating solutions, a node in a tree as in Fig. 7(b) may have more than K-1 descendants, e.g, if there are two choices for c_1, node n will have 3 descendants (one of them assigns b to mask 1 and the other two assign b to mask 2). Besides, multiple choices are available for connecting two nodes in the solution graph. By now, all of Issues 1-5 have been resolved by our cut generation algorithm and new solution graph.

(6) Calculating Overlay Length: To get the minimum total overlay length, we set the cost of a node in the solution graph as the total length of the overlay caused by the cuts in that node, and set the cost of an edge according to the cuts used in the edge connecting two nodes. As a feature may appear in multiple sets, the cost of a cut may be counted multiple times in a path of the solution graph. To avoid this, we assign a *primary set* to each feature. A set A is the primary set of a feature a iff $LeftBound(A) \leq LeftBound(a) < RightBound(A)$. In this case, a is called one of the *primary features* of A. Each feature has one and only one primary set. When setting the cost of a node for a set A, we only count the overlay length induced by the cuts between the primary features of A. When setting the cost of an edge connecting from set B to its adjacent set A, we count the overlay length induced by a cut between feature b in B and feature a in A only when A is the primary set for a and not the primary set for b. With this setting, each cut will be counted exactly once.

(7) Reporting Unresolved Conflicts: By now, a solution with the minimum total cost of overlay errors and without any unresolved conflict is guaranteed to be found by searching a shortest path if there is one. If stitch is allowed, the cost of stitches can also be easily incorporated into the solution graph [3]. However, if there is no unresolved-conflict-free solution for a row, the solution graph cannot report the locations of the unresolved conflicts. Reporting details of the unresolved conflicts to designers can help to modify and further optimize the layout, and is thus critical. To handle this, the solution graph is extended as follows to report unresolved conflicts for MPC decomposition. First, unresolved conflicts are allowed to appear in the nodes. By modifying the tree in Fig. 7(b) slightly, all the solutions with or without unsolved conflicts for the layout in Fig. 7(a) can be enumerated. Second, unresolved conflicts are also allowed when connecting two nodes, and the cost of the edge is a weighted sum of the number of unresolved conflicts, the cost of overlay errors and possibly the number of stitches. To avoid multiple counting of conflicts, we apply a similar technique as in the counting of cuts. With this extension, a shortest path in the solution graph can give the minimum number of unresolved conflicts and the locations of these conflicts.

A full solution graph allowing unresolved conflicts may have a large number of nodes and edges. To speed up, before constructing a full solution graph, we will first build a solution graph without considering unresolved conflicts, and try to find a legal solution first. If this fails, we will build a full solution graph with unresolved conflicts. We will maintain an upper bound on the number of conflicts in each node and edge in a full solution graph to reduce the complexity of the graph.

3) Summary:

Layout decomposition for MPC can be solved optimally in polynomial time for a row-structure standard cell based design with our solution graph. This is because there are a fixed number of tracks in a row, and the distance between two adjacent slicing lines is normally $2 \cdot wc_{max} + cs_{min}$, and the cuts and features in non-adjacent sets are not related. Sometimes the distance between two adjacent slicing lines may be larger than $2 \cdot wc_{max} + cs_{min}$ as a line may be moved to make it coincide with boundaries of features, but this move will only increase the number of features in a set by a constant. Therefore there are upper bounds on the number of features and number of cuts in a set, and thus there are also an upper bound on the size of the trees used to enumerate the solutions of each set. Thus the enumeration and the shortest path searching can be done in polynomial time [3], [11]. Notice that our solution graph approach is also applicable when $K = 4$. The major difference is that each non-leaf node will have at least $K-1 = 3$ instead of 2 descendants in the tree of Fig. 7(b) when enumerating solutions for a set.

III. Optimization for MPEC

Instead of using optical lithography to do cutting in MPC, MPEC uses e-beam to do cutting. By using the maskless lithography with very high resolution, the cutting process is much more accurate. We assume VSB that creates one rectangle with one shot is used and there is no conflict between e-beam cuts. The objective in MPEC decomposition is to minimize the number of e-beam shots and the number of conflicts between positive masks.

Similar to MPC decomposition, we first find out cut candidates. To avoid sliver (a rectangle with width smaller than a given threshold value ε), when generating cut rectangles for the 3 types of conflicts (Section II-B), we will not generate rectangles with width $< \varepsilon$. As our cut generation is a rectangle based approach, we can directly use these rectangles (before we take union of these rectangles) in VSB shots.

We use a procedure called *rectangle merging* to optimize the number of e-beam shots. It seems that merging is not needed in MPEC because cuts will not conflict with each other. However, rectangle merging can reduce the shot count effectively. If a rectangle a is used to cover two rectangles b and c without violating design rules, b and c are called merged into a and this merging can reduce one shot. We require the two rectangles to be within cs_{min} to avoid producing an oversize rectangle, which is not preferred in e-beam lithography.

There are two situations where the rectangle merging procedure will be applied when constructing the solution graph. The first one is when we enumerate solutions for a set, we try to minimize the number of shots in a solution. The second one is when we connect two nodes in the solution graph, we try to merge rectangles to reduce the shot count in the combined solution of the two sets. To minimize the number of shots and the number of conflicts between positive masks for a standard cell row, we set the node cost and edge cost of a solution graph as a weighted sum of the numbers of shots and conflicts (and also the number of stitches if stitch is allowed). A shortest path can be found similarly. Our method can also be easily extended to handle multiple patterning lithography with hybrid optical and e-beam cuts, by setting the cost in the solution graph accordingly.

IV. Experimental Results

The proposed algorithms are implemented in C++, on a 3.0 GHz Linux machine with 7 GB memory.

Result for MPC:

We first compare with the previous work [8] for MPC

TABLE I. RESULT FOR MPC

Benchmark		[8]		Ours			
Data	P#	C#	Time	C#	Time	Speedup	OVRed(%)
C1	1109	0	9.91	0	0.06	165.2	42.1
C2	2216	0	10.76	0	0.17	63.3	29.8
C3	2411	0	15.70	0	0.11	142.7	59.2
C4	3262	1	27.52	1	0.13	211.7	53.4
C5	5125	3	31.42	3	0.23	136.6	49.0
C6	7933	0	37.47	0	0.40	93.7	54.3
C7	10189	1	56.98	1	0.49	116.3	58.8
C8	14603	1	71.78	1	0.67	107.1	59.5
C9	14575	39	147.34	39	0.60	245.6	43.6
C10	21253	1	109.80	1	1.04	105.6	56.0
S1	4611	0	16.82	0	0.21	80.1	65.8
S2	67696	10	374.73	10	2.86	131.0	52.2
S3	157455	32	916.66	32	7.24	126.6	47.8
S4	168319	34	925.04	34	7.23	127.9	48.6
S5	159952	29	854.27	29	7.48	114.2	51.8
Avg.	42714	10	-	10	-	131.2	51.5

TABLE II. RESULT FOR MPEC

	Modified ILP		Ours			
Data	C#	Time	C#	Time	Speedup	ShotRed(%)
C1	0	8.80	0	0.26	33.8	65.9
C2	0	10.11	0	0.87	11.6	53.5
C3	0	15.21	0	0.22	69.1	70.5
C4	0	26.65	0	0.23	115.9	72.7
C5	0	29.21	0	0.61	47.9	67.8
C6	0	35.66	0	1.46	24.4	67.7
C7	0	48.47	0	1.23	39.4	72.0
C8	0	67.82	0	1.57	43.2	73.3
C9	0	134.51	0	1.22	110.3	72.1
C10	0	103.79	0	2.88	36.0	71.9
S1	0	15.87	0	0.31	51.2	73.7
S2	0	320.28	0	8.94	35.8	69.6
S3	0	842.37	0	44.28	19.0	68.0
S4	0	890.24	0	25.31	35.2	67.7
S5	0	740.34	0	24.50	30.2	67.6
Avg.	0	-	0	-	46.9	68.9

decomposition when $K = 3$ on the benchmarks provided by the authors of [8] using the same setting of cs_{min} (the benchmarks used in [9] are not available to us).

The cut generation method proposed in this work is different from the method used in [8], and thus we cannot compare with the decomposition results reported in [8] directly. Therefore, we implemented the method in [8] (an ILP approach) by using the same set of cuts of ours (i.e., the inputs to the ILP approach and our decomposition approach are exactly the same), and use CPLEX [14] as the ILP solver. To compare with [8] fairly, in our method, we turn off the option that unresolved conflicts between cutting masks can exist in a solution (i.e., unresolved conflicts can only exist between positive masks) and make the cost of conflicts dominate the cost of overlay.

The comparisons are shown in Table I, where P$\#$ is the number of polygons, C$\#$ denotes the number of unresolved conflicts, Time is wall-clock time in seconds, and OVRed is the reduction of total overlay length when we consider overlay while setting the cost in the solution graph, compared with the case that we do not consider this (i.e., only minimizing conflict numbers). Comparing with the method in [8], our method can always get the optimal C$\#$, which verifies our effectiveness. Our method also runs orders of magnitude faster with speedup up to 245.6$\times$. For overlay cost, it can be seen that with our overlay length minimization, the total overlay length is reduced significantly (the conflict numbers remain the same and runtime is almost the same). Hence, if overlay is not considered as in [8], much larger overlay length will be resulted. We can conclude that our approach can reduce manufacturing cost remarkably in much shorter time comparing with [8].

We then compare with [9]. The average speedup of [9] over the method in [8] is only 1.59$\times$, as reported in [9], while our method is on average 131.2$\times$ faster than the method in [8]. Thus, our method is much faster than that in [9]. Besides, our method can always get the optimal solution while [9] may lose optimality due to the relaxation.

Similar to [9], in the experiments we focus on the situation where stitch is not used. However, stitch can be easily incorporated into our method as described in Section II.

Result for MPEC:

For the purpose of comparison on MPEC decomposition, we modify the ILP formulation in [8] to perform MPEC decomposition and use its result as a baseline. The result for MPEC is shown in Table II. It can be seen that our method can achieve 46.9$\times$ speedup on average over the ILP method. As there is no conflict between e-beam cuts, zero conflict can be achieved for all the datasets. ShotRed is the reduction on the number of e-beam shots of our method, compared with the case that we do not consider the minimization of shot number. The average reduction is 68.9%, which verifies the effectiveness of our method on the optimization of e-beam shots.

V. CONCLUSION

In this paper, we have proposed layout decomposition methods for multiple patterning lithography with cutting process, including MPC and MPEC. Experimental results demonstrated our effectiveness and efficiency.

REFERENCES

[1] A. B. Kahng, C. H. Park, X. Xu and H. Yao, “Layout Decomposition for Double Patterning Lithography,” in *Proc. ICCAD*, 2008.

[2] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, “Layout Decomposition for Triple Patterning Lithography,” in *Proc. ICCAD*, 2011.

[3] H. Tian, H. Zhang, Q. Ma, Z. Xiao and M. D. F. Wong, “A Polynomial Time Triple Patterning Algorithm for Cell Based Row-structure Layout,” in *Proc. ICCAD*, 2012.

[4] J. Kuang and E. F. Y. Young, “An Efficient Layout Decomposition Approach for Triple Patterning Lithography,” in *Proc. DAC*, 2013.

[5] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan and X. Zeng, “Layout Decomposition with Pairwise Coloring for Multiple Patterning Lithography,” in *Proc. ICCAD*, 2013.

[6] B. J. Lin, “Lithography Till Ehe End of Moore’s Law,” in *Proc. ISPD*, 2012.

[7] B. Yu, J.-R. Gao and D. Z. Pan, “Triple Patterning Lithography (TPL) Layout Decomposition Using End-Cutting,” in *Proc. SPIE*, 2013.

[8] B. Yu, S. Roy, J.-R. Gao and D. Z. Pan, “Triple Patterning Lithography Layout Decomposition Using End-Cutting,” in *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 14, no. 1, 2014.

[9] Y. Kohira, et al., “Fast Mask Assignment Using Positive Semidefinite Relaxation in LELECUT Triple Patterning Lithography,” in *Proc. ASPDAC*, 2015.

[10] J. Kuang, W.-K. Chow and E. F. Y. Young, “Triple Patterning Lithography Aware Optimization for Standard Cell Based Design,” in *Proc. ICCAD*, 2014.

[11] Z. Xiao and Y. Du and H. Tian and M. D. F. Wong, “Optimally Minimizing Overlay Violation in Self-aligned Double Patterning Decomposition for Row-based Standard Cell Layout in Polynomial Time,” in *Proc. ICCAD*, 2013.

[12] J.-R. Gao, B. Yu and D. Z. Pan, “Self-aligned Double Patterning Layout Decomposition with Complementary E-Beam Lithography,” in *Proc. ASPDAC*, 2014.

[13] Y. Ding, C. Chu and W.-K. Mak, “Throughput Optimization for SADP and E-beam Based Manufacturing of 1D Layout,” in *Proc. DAC*, 2014.

[14] CPLEX Optimizer: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/