A Novel Layout Decomposition Algorithm for Triple Patterning Lithography*

Shao-Yun Fang[1], Yao-Wen Chang[1,2], and Wei-Yu Chen[1]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
yuko703@eda.ee.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw; cweiyu@eda.ee.ntu.edu.tw

ABSTRACT

While double patterning lithography (DPL) has been widely recognized as one of the most promising solutions for the sub-22nm technology node to enhance pattern printability, triple patterning lithography (TPL) will be required for gate, contact, and metal-1 layers which are too complex and dense to be split into only two masks, for the 15nm technology node and beyond. Nevertheless, there is very little research focusing on the layout decomposition for TPL. The recent work [16] proposed the first systematic study on the layout decomposition for TPL. However, the proposed algorithm extending a stitch-finding method used in DPL may miss legal stitch locations and generate conflicts that can be resolved by inserting stitches for TPL. In this paper, we point out two main differences between DPL and TPL layout decompositions. Based on the two differences, we propose a novel TPL layout decomposition algorithm. We first present two new graph reduction techniques to reduce the problem size without degrading overall solution quality. We then propose a stitch-aware mask assignment algorithm, based on a heuristic that finds a mask assignment such that the conflicts among the features in the same mask are more likely to be resolved by inserting stitches. Finally, stitches are inserted to resolve as many conflicts as possible. Experimental results show that the proposed layout decomposition algorithm can achieve around 56% reduction of conflicts and more than 40X speed-up compared to the previous work.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design, Performance

Keywords

Triple Patterning Lithography, Layout Decomposition, Manufacturability

*This work was partially supported by IBM, SpringSoft, TSMC, and NSC of Taiwan under Grant No's. NSC 100-2221-E-002-088-MY3, NSC 99-2221-E-002-207-MY3, NSC 99-2221-E-002-210-MY3, and NSC 98-2221-E-002-119-MY3.

DAC'12, June 03 - 07, 2012, San Francisco, California, USA.

1. INTRODUCTION

In order to overcome the resolution limit of conventional optical lithography, various next-generation lithography systems have been proposed, such as extreme ultraviolet (EUV) lithography and electron beam (e-beam) lithography. However, these next-generation lithography systems are still not available for mass production due to several barriers. Consequently, multiple patterning lithography, which extends current 193nm immersion lithography to patterning the sub-32nm half pitch node, corresponding to the sub-22nm technology node, has been regarded as one of most promising solutions to overcome the resolution limit of conventional optical lithography [1, 11].

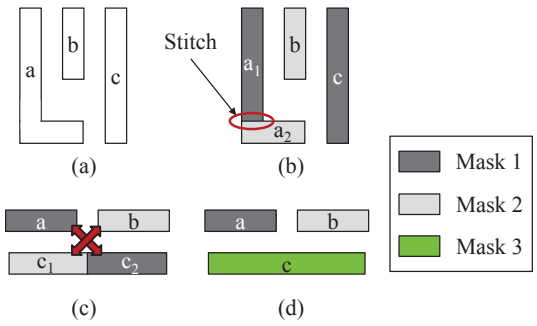

Figure 1: Multiple patterning lithography. (a) An example layout. (b) Layout decomposition for DPL with an inserted stitch. (c) An example layout which cannot be decomposed into two masks with stitch insertion. (d) The layout can be decomposed into three masks with TPL.

The simplest case of multiple patterning lithography is double patterning lithography (DPL). During the layout decomposition process in DPL, if the distance between two features is less than the minimum coloring spacing min_{cs}, they should be assigned to different masks. Otherwise, there will be a conflict between these two features. In some cases, to resolve a conflict, a feature may be split into two touching parts and assigned to different masks. As shown in Figures 1(a) and (b), the layout can be decomposed into two masks by inserting a *stitch*. However, even with stitch insertion, some conflicts still cannot be resolved in DPL. Figure 1(c) shows an example of an irresolvable conflict in DPL. Since there exists a region in feature c which is within min_{cs} from features a and b, even though we split feature c into two parts, each of the two parts still forms a conflict with a and b. As a result, it is impossible to produce a conflict-free solution for this layout with DPL. However, this problem can be easily resolved if three masks are available; that is, the layout can be decomposed without conflict with triple patterning lithography (TPL), as shown in Figure 1(d).

For the 15nm technology node and beyond, TPL will be required for gate, contact, and metal-1 layers, which are too complex and dense to be split into two masks [2, 3, 9, 11, 12, 16]. With the same principle of DPL, the original layout is decomposed into three masks and manufactured through three exposure/etching steps in TPL. Although TPL can resolve some conflicts that cannot be resolved in DPL, the

TPL layout decomposition problem is not easier than the DPL layout decomposition problem. Actually, since the minimum coloring spacing min_{cs} set in TPL is larger than that in DPL, more pairs of features are within min_{cs} of each other; that is, more pairs of features need to be assigned to different masks in TPL. Thus, the problem turns out to be more complicated and difficult.

There is very little research focusing on the layout decomposition for TPL. As the DPL layout decomposition problem is generally regarded as a two-coloring problem, the TPL layout decomposition problem can be modeled as a three-coloring problem. Cork et al. [3] investigated the challenges involved in the TPL layout decomposition problem for the contact layer. They proposed a SAT-based three-coloring algorithm and then compared the scalability of different coloring algorithms using a variety of contact patterns based on Penrose tiles, which have been proven to be three-colorable. However, this work only deals with contact arrays, not general layouts, and thus the stitch issue is not considered. Yu et al. [16] proposed the first systematic study on the general layout decomposition for TPL. The algorithm first generates a set of stitch candidates by using the projection method proposed by Kahng et al. [7] and widely used in DPL. Then, the TPL layout decomposition is formulated as an integer linear programming (ILP) to simultaneously minimize the numbers of conflicts and stitches. Since solving ILP is time-consuming, they also proposed three acceleration techniques without loss of solution quality. To improve the scalability, they further proposed semidefinite programming (SDP) based approximation algorithm. However, the projection method they used to find a set of stitch candidates may miss legal stitches, which we will prove in Section 2.2.2. Thus, the algorithm may generate some conflicts that can be resolved by inserting stitches for TPL.

In this paper, we point out two main differences between DPL and TPL layout decompositions. Based on the two differences, we propose a novel TPL layout decomposition algorithm. Experimental results show that our layout decomposition algorithm can achieve around 56% reduction of conflicts and more than 40X speed-up, compared to the previous work proposed by Yu et al. [16]. In addition, we compute the cost of our layout decomposition results by using the objective function proposed in [16], and the resulting cost is also reduced by more than 34% compared to their algorithm. This phenomenon reveals that their method may fail to resolve conflicts which can be resolved by inserting stitches.

The rest of this paper is organized as follows: Section 2 gives the problem formulation of this paper and points out the two differences between DPL and TPL. In Section 3, our layout decomposition approach for TPL is presented. Experimental results are reported in Section 4. Finally, we conclude our work in Section 5.

2. PRELIMINARIES

In this section, we first formally define the layout decomposition problem for TPL in Section 2.1. Then, we discuss two main differences between DPL and TPL layout decompositions in Section 2.2, which show that the TPL layout decomposition problem is more difficult than the DPL one.

2.1 Problem Formulation

In this work, we solve the layout decomposition problem for TPL. There are two critical issues in the layout decomposition problem: conflicts and stitches. If the distance between two features in the layout is less than the minimum coloring spacing min_{cs}, they should be assigned to different masks. Otherwise, a conflict exists between these two features. To resolve a conflict, a feature may be split into two parts and assigned to different masks. However, this introduces stitches, which may cause yield loss due to overlay error and increase manufacturing cost. Therefore, conflict and stitch minimizations are the main challenges in the layout decomposition for TPL. Thus, the layout decomposition problem for TPL can be formulated as follows:

PROBLEM 1. *Given a layout represented by a set of polygonal features, the minimum wire width, the minimum spacing, the overlay margin, and the minimum coloring spacing for TPL, assign all features in the layout to three masks while the numbers of conflicts and stitches are minimized.*

2.2 Differences between DPL and TPL

In this section, we will discuss two main differences between DPL and TPL layout decompositions, which show that the TPL layout decomposition problem is more complicated and more difficult than the DPL layout decomposition problem.

2.2.1 Problem Complexity

The DPL layout decomposition problem is generally modeled as a two-coloring problem on a conflict graph. Then, determining whether a graph is two-colorable can be done in linear time by checking if there exists any odd cycle. Moreover, two-coloring a two-colorable graph can also be done in linear time with a breadth-first search (BFS) algorithm.

However, for the TPL layout decomposition, the problem becomes much more complicated. As the DPL layout decomposition problem is modeled as a two-coloring problem, the TPL layout decomposition problem can be modeled as a three-coloring problem. It can be shown that a three-coloring problem is NP-complete. First, determining whether a general graph is three-colorable is NP-complete, even for a planar graph [5]. Furthermore, even though the graph is three-colorable, coloring a three-colorable graph with four colors is NP-complete [8], and thus three-coloring a three-colorable graph is also NP-complete. Therefore, the TPL layout decomposition problem is NP-complete even if the conflict graph used to model TPL conflicts in a layout is three-colorable and planar [16].

2.2.2 Stitch Finding

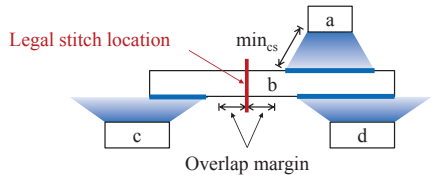

Figure 2: The projection method for stitch location finding.

Most previous works for DPL and TPL [16] utilize a projection method proposed by Kahng et al. [7] to find legal stitch locations before mask assignment. The reason is that this projection method has been proven to be able to find all possible stitch locations in DPL [7]. As illustrated in Figure 2, the method computes the projections on each feature from its neighboring features within the minimum coloring spacing min_{cs}. Then, a stitch can be inserted at a location where no projection covers and no overlap margin violation occurs.

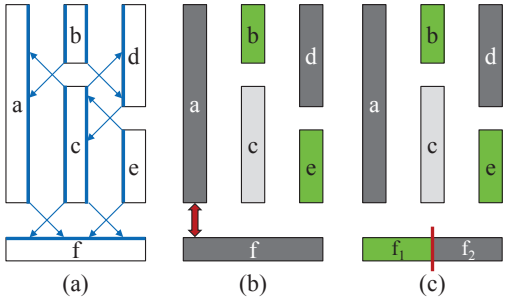

Figure 3: An example showing that not all legal stitch locations can be found by the projection method in TPL. method.

However, the property does not hold in TPL. That is, we cannot find all legal stitch locations by using the projection method in TPL. Figure 3 shows an example. In the layout depicted in Figure 3(a), no legal stitch position can be found by using the projection method since all features are covered by projections. Since the layout is not three-colorable, at least one conflict exists after a mask assignment process, as shown in Figure 3(b). In fact, as illustrated in Figure 3(c), we can find a legal stitch location and resolve the conflict by inserting a stitch. With the above example, we can conclude that not all legal stitch locations can be found by the projection method in TPL. Therefore, if we only insert stitches at the locations found by the projection method, we may miss legal stitches and generate conflicts that can be resolved with stitch insertion.

3. LAYOUT DECOMPOSITION FOR TPL

In this section, we first present the overall flow of our layout decomposition approach for TPL in Section 3.1. Then, we detail each step in the flow in Sections 3.2–3.5.

3.1 Algorithm Flow

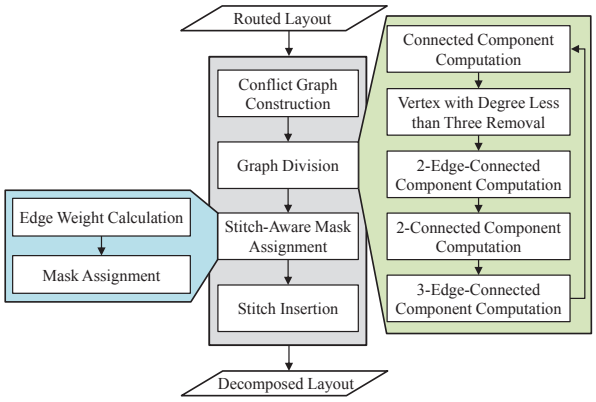

Figure 4: The flow of our layout decomposition for TPL.

The overall flow of our layout decomposition for TPL is shown in Figure 4. First, we construct a conflict graph to transform the original geometric problem into a graph problem, and thus the TPL layout decomposition problem can be modeled as a three-coloring problem. Since the three-coloring problem is NP-complete, the time required to exactly solve it increases dramatically with the numbers of vertices and edges in the graph. Therefore, to reduce the problem size, we shall divide the conflict graph into a set of coloring-independent groups, and then the color assignment problem can be solved independently for each group without affecting overall solution quality. The graph division method incorporates three graph simplification methods proposed by Yu et al. [16]: *connected component computation*, *vertex with degree less than three removal*, and *2-edge-connected component computation*. In addition, we propose two new graph reduction techniques, *2-connected component computation* and *3-edge-connected component computation*, to further enhance the program efficiency. Next, to solve the color (mask) assignment problem, we propose a stitch-aware mask assignment algorithm, which is based on a heuristic that finds a mask assignment such that the conflicts among the features in the same mask are more likely to be resolved by inserting stitches. Finally, we resolve as many conflicts as possible with stitch insertion for each mask. The four steps in the flow are detailed in the following sections.

3.2 Conflict Graph Construction

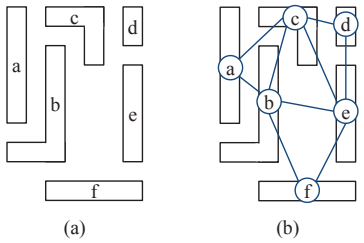

Figure 5: Conflict graph construction. (a) An input layout represented by a set of polygonal features. (b) The corresponding conflict graph.

First, for the purpose of simplification, we construct a conflict graph [7] to transform the original geometric problem into a graph problem. As shown in Figure 5, given a layout composed of a set of polygonal features, the corresponding conflict graph $G = (V, E)$ is constructed in which each vertex $v_i \in V$ represents a feature i and an edge $e_{i,j} \in E$ exists between two vertices if the distance between the two corresponding features i and j is less than the minimum coloring spacing min_{cs}. An edge in the conflict graph is a conflict candidate and is defined as a *conflict edge* (*CE*). Note that in most previous works for the layout decomposition in DPL and TPL, stitch edges (SEs) indicating legal stitch locations are generated with the projection method and inserted in a conflict graph as well. However, in our work, we do not insert SEs in a conflict graph because we have proved that not all legal stitch locations can be found by the projection method in TPL. After the construction of a conflict graph, the layout decomposition problem can be modeled as a three-coloring problem, and a conflict exists if two vertices connected by a CE are assigned the same color.

3.3 Graph Division

Since the three-coloring problem is NP-complete, the time required to exactly solve it increases dramatically with the numbers of vertices and edges in a conflict graph. Therefore, to reduce the problem size, we first incorporate three graph simplification methods proposed by Yu et al. [16]: *connected component computation*, *vertex with degree less than three removal*, and *2-edge-connected component computation*. In addition, we propose two new graph reduction techniques, *2-connected component computation* and *3-edge-connected component computation*, to further reduce the problem size and enhance the efficiency. Note that all of the five techniques can be performed in linear time $O(|V| + |E|)$. We detail the graph reduction techniques in the following subsections.

3.3.1 Connected Component Computation

A connected component of a graph is a subgraph in which any two vertices are connected to each other with paths. Since no edge exists between any two components, the color assignment problem can be solved independently for each component, and the final solution can be obtained by taking the union of sub-solutions.

3.3.2 Vertex with Degree Less than Three Removal

A component in a conflict graph can be simplified by temporarily removing all vertices with degree less than three. As illustrated in Figure 6(a), vertex v_a is a vertex with degree less than three, and thus we temporarily remove it from the component. After solving the three-coloring problem on the remaining graph, as shown in Figure 6(b), the removed vertex v_a is then added and colored according to the colors of its adjacent vertices. The final coloring solution is shown in Figure 6(c).

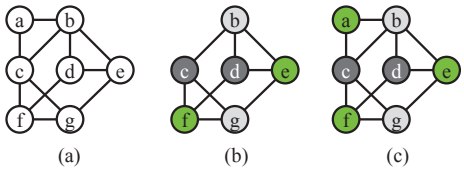

Figure 6: An illustration of the removal of vertices with degree less than three.

3.3.3 2-Edge-Connected Component Computation

Two-edge-connected-component computation is also defined as bridges computation [16]. A bridge of a graph is an edge such that the graph will be decomposed into two components if the edge is removed. If a bridge exists in a conflict graph, the three-coloring problem can be solved independently on the two components connected by the bridge, and then the two sub-solutions can be merged with a simple color remapping process. An example is shown in Figure 7. The conflict graph in Figure 7(a) can be decomposed into two components by removing the edge e_{cd}. After solving the three coloring problem on each component as shown in Figures 7(b) and (c), the two components can be merged and recolored such that no conflict occurs between the vertices v_c and v_d.

3.3.4 2-Connected Component Computation

In addition to the above three methods proposed by Yu et al. [16], we propose two new graph reduction techniques to further reduce the problem size and enhance program efficiency. The first technique is 2-connected component computation. A 2-connected component of a

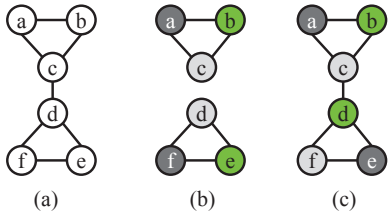

Figure 7: An illustration of 2-edge-connected component computation.

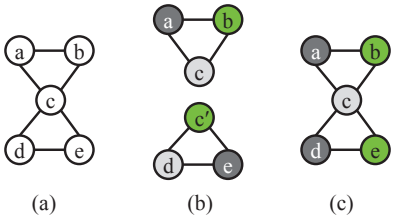

Figure 8: An illustration of 2-connected component computation.

connected graph is a maximal connected subgraph that remains connected while removing any single vertex. In addition, a vertex of a connected graph is called a *cut vertex* if its removal decomposes the graph into two or more connected components. Thus, we can identify 2-connected components of a connected graph by finding cut vertices.

Two 2-connected components derived by duplicating a cut vertex can be colored independently and merged with a similar color remapping method used in 2-edge-connected-component computation. As shown in Figure 8(a), vertex v_c is a cut vertex. By creating a pseudo-vertex c', the conflict graph can be divided into two 2-connected components, as illustrated in Figure 8(b). After solving the three-coloring problem on the two components independently, the two sub-solutions can be merged into one three-coloring solution with a simple recoloring process on one component, as shown in Figure 8(c).

For the 2-connected component computation, we have the following theorem:

THEOREM 1. *Solving the three-coloring problem on each 2-connected subgraph and then merging the sub-solutions at the cut vertex into one three-coloring solution with color remapping does not degrade the overall solution quality; that is, the number of conflicts does not increase with this graph division method.*

We apply a depth-first search (DFS) algorithm presented by Tarjan [13] to find all the 2-connected components of a conflict graph. The algorithm computes all the 2-connected components and identifies all the cut vertices simultaneously during only one DFS in linear time $O(|V|+|E|)$, where $|V|$ and $|E|$ are respectively the number of vertices and the number of edges in a conflict graph $G = (V, E)$.

3.3.5 3-Edge-Connected Component Computation

The second graph reduction technique we propose is 3-edge-connected component computation. A 3-edge-connected component of a graph G is a maximal connected subgraph of G such that for every two distinct vertices in the subgraph, there are at least three edge-disjoint paths in G connecting them. Three-edge-connected components of a conflict graph can be identified by finding 2-cuts in the graph, where a 2-cut is a pair of edges whose removal would disconnect the graph [15].

Similar to 2-edge-connected component computation, each 3-edge-connected component can be colored independently, and then components can be merged with a color remapping process. As shown in Figure 9(a), the pair of edges e_{df} and e_{cg} is a 2-cut, and the conflict graph can be decomposed into two connected components by removing the 2-cut, as illustrated in Figure 9(b). After solving the three-coloring problem on the two components independently, the two sub-solutions can be merged into one three-coloring solution with a simple recoloring process on one component, as shown in Figure 9(c).

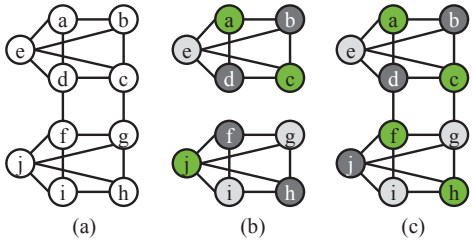

Figure 9: An illustration of 3-edge-connected-component computation.

For 3-edge-connected component computation, we have the following theorem:

THEOREM 2. *Solving the three-coloring problem on each 3-edge-connected subgraph and then merging the sub-solutions at the 2-cut into one three-coloring solution with color remapping does not degrade the overall solution quality; that is, the number of conflicts does not increase with this graph division method.*

We adopt an algorithm presented by Tsin [14] to find all the three-edge-connected components of a conflict graph in linear time $O(|V| + |E|)$, where $|V|$ and $|E|$ are respectively the number of vertices and the number of edges in a conflict graph $G = (V, E)$. The algorithm computes all the three-edge-connected components and identifies all the 2-cuts by performing only one depth-first search (DFS) over the given graph.

3.4 Stitch-Aware Mask Assignment

After the graph division operations, we solve the three-coloring problem on each sub-conflict graph. We propose a stitch-aware mask (color) assignment algorithm, which is based on a heuristic that finds a mask (color) assignment such that the conflicts among the features in the same mask are more likely to be resolved by inserting stitches.

3.4.1 Edge Weight Calculation

We observe that some conflicts are more likely to be resolved by inserting stitches, whereas some conflicts are difficult to be resolved. Therefore, while performing mask assignment, we intend to find a mask assignment such that the conflicts among the features in the same mask are more likely to be resolved by inserting stitches. To achieve this goal, we first assign a weight to each conflict edge to reflect how hard the corresponding conflict can be resolved by inserting stitches. For an edge of a vertex, we calculate the weight w as follows: an edge between the target vertex and an adjacent vertex indicates that the two corresponding features are within min_{cs}. Thus, the adjacent feature creates a projection p on the target feature. If there are more other projections created by other adjacent features overlapped with p, the target conflict is more difficult to be resolved by inserting stitch. Thus, we set the weight of the target edge to be the maximum density (number of projections overlapped with each other) of other projections which are overlapped with p.

See Figure 10 for an example. In the layout depicted in Figure 10(a), feature b is the target feature. For edge e_{ab}, the projection from feature a to b is p_a, and projections of edges e_{cb} and e_{db} are overlapped with p_a. Since the projections of e_{cb} and e_{db} are not overlapped with each other, the maximum density of projections overlapped with p_a is one. Thus, $w_{ab} = 1$ (similarly, $w_{cb} = w_{db} = 1$). During the mask assignment procedure, if the three adjacent features a, c, and d are assigned different colors, a conflict must exist at e_{ab}, e_{cb}, or e_{db}. In this case, as shown in Figure 10(b), we can insert a stitch at feature b to resolve the conflict. In contrast, in the layout depicted in Figure 10(c), the maximum density of projections overlapped with p_a is two, and thus $w_{ab} = 2$ (similarly, $w_{cb} = w_{db} = 2$). In this case, if the three adjacent features a, c, and d are assigned different colors, the conflict at e_{ab}, e_{cb}, or e_{db} cannot be resolved by inserting a stitch at b, since there exists a section b_2 that cannot be assigned any color without conflict, as shown Figure 10(d).

3.4.2 Mask Assignment

After calculating the weight for each edge, we present a stitch-aware mask assignment algorithm to find a mask assignment such that the

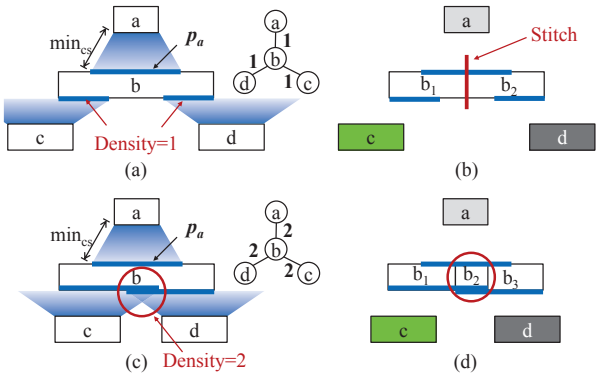

Figure 10: Edge weight calculation. (a) The projections of e_{cb} and e_{db} are overlapped with the projection of e_{ab}, but are not overlapped with each other. Thus $w_{ab} = 1$. (b) A stitch can be inserted to resolve the conflict. (c) The projections of e_{cb} and e_{db} are overlapped with the projection of e_{ab} and are overlapped with each other. Thus $w_{ab} = 2$. (d) In this case, stich cannot be inserted to resolve the conflict.

Algorithm: Stitch-Aware Mask Assignment Algorithm
Input: $G(V, E)$, /* a conflict graph */
$CNUM$, /* # candidate independent sets */
$TNUM$, /* a threshold number of vertices */
$CSIZE$, /* size of a candidate vertex list */
Output: C_1, C_2, and C_3 /* three color classes */

```
1   i ← 0
2   V' ← V
3   while V' ≠ ∅ and i < 3
4       i ← i + 1
5       mincost ← ∞
6       for j = 1 to CNUM
7           U ← V', C ← ∅, X ← ∅
8           while |U| > TNUM
9               CL ← {v_a | Σ_{e_ax∈E, v_x∈X} w_ax is max}, where CL
                     is a candidate list of size CSIZE
10              Select v_a ∈ CL at random
11              N(v_a) ← {v_b | e_ab ∈ E}
12              U ← U − {v_a} − N(v_a)
13              C ← C ∪ {v_a}
14              X ← X ∪ N(v_a)
15          end while
16          C' ← Use exhaustive search to find an independent set
                 that maximizes Σ_{e_ab∈F} w_ab, where
                 F = {e_ab | v_a ∈ C', v_b ∈ V' − C ∪ C'}
17          C ← C ∪ C'
18          cost ← Σ_{e_ab∈E, v_a,v_b∈V'−C} w_ab
19          if cost < mincost
20              C_i ← C
21              mincost ← cost
22          end if
23      end for
24      V' ← V' − C_i
25  end while
26  if V' ≠ ∅
27      Assign each remaining vertex v_a ∈ V' to a color class C_i
        such that the conflict occurs at an edge with the smallest
        weight
28  end if
```

Figure 11: Stitch-aware mask assignment algorihtm

conflicts among the features in the same mask are more likely to be resolved by inserting stitches. The proposed mask assignment algorithm is based on a modified recursive largest first (RLF) algorithm. The RLF algorithm was proposed by Leighton [10] and improved as the XRLF algorithm by Johnson et al. [6], which is a very successful heuristic for solving the graph coloring problem. Therefore, we modify the XRLF algorithm to solve the mask (color) assignment problem on an edge-weighted conflict graph. With an input conflict graph $G(V, E)$, the algorithm generates three output subgraphs $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ and $G_3(V_3, E_3)$, where $V_1 \cup V_2 \cup V_3 = V$, $V_i \cap V_j = \emptyset$ for $i \neq j$, and $E_i = \{e_{ab} | v_a \in V_i \text{ and } v_b \in V_i\}$. Each subgraph represents a color class, and thus vertices in a subgraph will be colored with the same color. Since the weight of an edge indicates how hard the corresponding conflict can be resolved with stitch insertion, an edge with a larger weight is expected not to appear in any of the three subgraphs.

The stitch-aware mask assignment algorithm is summarized in Figure 11. When constructing a color class, the objective is to maximize the size of the color class and minimize the total edge weight of the residual graph. Our algorithm constructs the three color classes by first constructing three independent sets sequentially. For constructing each independent set, a number of candidate independent sets are first generated, and the one minimizing the total edge weight of the residual graph is chosen. After three independent sets are generated, each remaining vertex not included in the independent sets is assigned to one of the three independent sets such that the corresponding conflict occurs at an edge with the smallest weight. Finally, the three color classes are generated and no vertex is left.

An example of a mask assignment process is shown in Figure 12. For an edge-weighted conflict graph shown in Figure 12(a), vertices v_c, v_e are chosen in the first independent set, as shown in Figure 12(b). After the second and the third independent sets are constructed as illustrated in Figures 12(c) and (d), v_g is left as a remaining vertex. Since the weight of the edge e_{fg} is smaller than those of the other adjacent edges of v_g, v_g is assigned to the third color class such that the corresponding conflict is more likely to be solved, as shown in Figure 12(e).

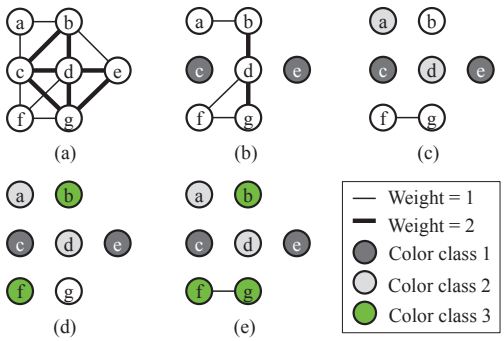

Figure 12: A stitch-aware mask assignment process. (a) An edge-weighted conflict graph. (b)(c)(d) The constructions of three independent sets. (e) The remaining vertex v_g is assigned to C_3 since the edge weight of e_{fg} is smaller than those of the other adjacent edges of v_g.

3.5 Stitch Insertion

After mask assignment, we resolve the conflicts between features by inserting stitches. We first compute the projections from neighboring features on the conflicting feature. See Figure 13(a) for an example. Then, the conflicting feature can be partitioned into several segments according to the ends of these projections, and these segments have different available colors (masks), as illustrated in Figure 13(b). Thus, if we can find at least one available color for each segment, then we can solve the conflict by inserting stitches. Otherwise, the conflict cannot be resolved with stitch insertion. Since there may be several stitch insertion combinations to resolve a conflict, we apply a plane sweep method to solve a conflict with a smaller number of stitches. As shown in Figure 13(c), the plane sweep method scans the conflicting feature from the left to the right (or from the right to the left) to find the longest continuous segments. After that, as illustrated in Figure 13(d), we insert a stitch at a suitable location on the feature and assign each segment an appropriate color.

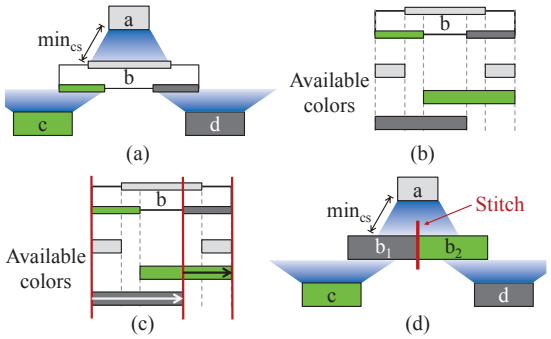

Figure 13: An example of stitch insertion. (a) Projection computation. (b) Feature partition with the ends of the projections (c) Feature scanning with the plane sweep method. (d) Conflict removal with stitch insertion.

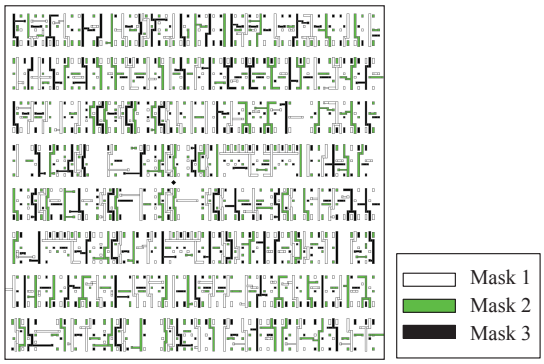

Figure 14: Layout decomposition result of C432.

Table 1: Comparison of the decomposition results.

	SDP Based [16]				Ours			
Design	#C	#S	Cost	cpu (s)	#C	#S	Cost	cpu (s)
C432	3	1	31	0.14	0	6	6	0.01
C499	0	0	0	0.19	0	0	0	0.01
C880	1	6	16	0.27	1	15	25	0.01
C1355	1	6	16	0.21	1	7	17	0.02
C1908	0	1	1	0.29	1	0	10	0.04
C2670	2	4	24	0.53	2	14	34	0.06
C3540	5	6	56	0.72	2	15	35	0.08
C5315	7	7	77	1.01	3	11	41	0.11
C6288	82	131	951	4.49	19	341	531	0.13
C7552	12	15	135	1.72	3	46	76	0.17
S1488	1	1	11	0.33	0	4	4	0.03
S38417	44	55	495	21.67	20	122	322	0.62
S35932	93	18	948	96.45	46	103	563	2.13
S38584	63	122	752	99.80	36	280	640	2.26
S15850	73	91	821	87.22	36	201	561	2.14
Avg.	2.28	0.40	1.51	40.29	1.00	1.00	1.00	1.00

4. EXPERIMENTAL RESULTS

The proposed layout decomposition algorithm for TPL was implemented in the C++ programming language on a 2.93 GHz Linux machine with 48 GB memory. We used the ISCAS-85 & 89 benchmarks provided by the authors of [16] to evaluate our algorithm. In addition, the metal-1 layer was used because it is one of the most complex layers. The minimum coloring spacing min_{cs} was set as 120nm for the first ten cases and as 100nm for the last five cases, which are the same to the updated setting provided by the authors of [16].

We compare the decomposition results between our algorithm and the SDP-based algorithm proposed by Yu et al. [16]. In Table 1, '#C" denotes the number of conflicts, "#S" the number of stitches, and "cpu" the computation time for the decomposition process. In addition, "Cost" is set as #C+0.1 × #S, which is the same as the objective function in [16] (the cost for a conflict is typically much larger than that for a stitch since the conflict incurs much higher manufacturing cost). The updated experimental results of the SDP-based algorithm are also provided by the authors of [16]. As shown in Table 1, compared to the SDP-based algorithm, our layout decomposition approach can averagely reduce the number of conflicts by 56% and achieve 40X speedup. In addition, although more stitches are used to resolve conflicts, the cost is reduced by more than 34%. The significant improvement in cost reduction reveals the fact that their proposed algorithm only inserts stitches at the locations found by the projection method, and thus some legal stitches are missed and some conflicts are generated, which can actually be resolved by inserting stitches in TPL. Figure 14 shows the layout decomposition result of the circuit C432.

5. CONCLUSIONS

In this paper, we have presented a novel layout decomposition approach for TPL. We have proven that the widely used projection method cannot find all legal stitch candidates for TPL. As a result, we have proposed a stitch-aware mask assignment algorithm which is based on a heuristic to find a mask assignment such that the conflicts among the features in the same mask are more likely to be resolved by inserting stitches. To further reduce the problem size and enhance program efficiency, we have also proposed two new conflict graph reduction techniques. Experimental results have shown that our layout decomposition algorithm can efficiently and effectively generate a good layout decomposition result with fewer conflicts and smaller total costs, compared to the previous work.

6. ACKNOWLEDGMENT

We would like to thank Mr. Bei Yu and the authors of [16] for providing benchmarks and updated experimental results.

7. REFERENCES

[1] G. E. Bailey, A. Tritchkov, J.-W. Park, L. Hong, V. Wiaux, E. Hendrickx, S. Verhaegen, P. Xie, and J. Versluijs, "Double pattern EDA solutions for 32nm HP and beyond," *Proc. SPIE*, vol. 6521, pp. 65211K, 2007.

[2] Y. Borodovsky, "Lithography 2009 overview of opportunities," in Semicon West, 2009.

[3] C. Cork, J.-C. Madre, and L. Barnes, "Comparison of triple-patterning decom- position algorithms using aperiodic tiling patterns," *Proc. SPIE*, vol. 7028, pp. 702839, 2008.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein "Introduction to Algorithms." The MIT Press, 2009.

[5] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplifed NP-complete problems," *Proc. STOC*, pp. 47–63, 1974.

[6] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning," Operations Research, 39:378–406, 1991.

[7] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE TCAD*, vol. 29, no. 6, pp. 939–952, 2010.

[8] S. Khanna, N. Linial, and S. Safra, "On the hardness of approximating the chromatic number," *Proc. ISTCS*, pp. 250–260, 1993.

[9] M. LaPedus, "SPIE: Intel to extend immersion to 11-nm," EE Times, Feb. 22, 2010.

[10] F. T. Leighton, "A graph coloring algorithm for large scheduling problems," Journal of Research of the National Bureau of Standards, 84(6):489–506, 1979.

[11] L. Liebmann and A. Torres, "A designer's guide to sub-resolution lithography: enabling the impossible to get to the 15nm node," *Proc. DAC*, 2011.

[12] R. Merritt, "Otellini: Intel to ship more SoCs than PC CPUs–someday," EE Times, Sep. 22, 2009.

[13] R. Tarjan, "Depth-first search and linear graph algorithms," *Proc. SWAT*, pp. 114–121, 1971.

[14] Y. H. Tsin, "A simple 3-edge-connected component algorithm," Theory of Computing Systems, 40:125–142, 2007.

[15] D. B. West, "Introduction to Graph Theory," Prentice Hall, 2001.

[16] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *Proc. ICCAD*, 2011.