An Efficient Layout Decomposition Approach for Triple Patterning Lithography

Jian Kuang
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
jkuang@cse.cuhk.edu.hk

Evangeline F. Y. Young
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT

Triple Patterning Lithography (TPL) is widely recognized as a promising solution for 14/10nm technology node. In this paper, we propose an efficient layout decomposition approach for TPL, with the objective to minimize the number of conflicts and stitches. Based on our analysis of actual benchmarks, we found that the whole layout can be reduced into several types of small feature clusters, by some simplification methods, and the small clusters can be solved very efficiently. We also present a new stitch finding algorithm to find all possible legal stitch positions in TPL. Experimental results show that the proposed approach is very effective in practice, which can achieve significant reduction of manufacturing cost, compared to the previous work.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design

Keywords

Triple Patterning Lithography, Layout Decomposition, Manufacturability

1. INTRODUCTION

The next generation and ideal lithography methods, such as Extreme Ultra-Violet (EUV), still face some critical technological challenges, especially on the manufacturing equipment side, and their availabilities are further delayed. As a consequence, multiple patterning lithography, which decompose one single layer into multiple masks and require the decomposition before manufacturing, is regarded as a good alternative.

In multiple patterning lithography, a conflict occurs when the distance between two features is less than a threshold cs_{min}, namely the minimum coloring spacing, and two conflicting features should be assigned to different masks by layout decomposition. Nonetheless we sometimes cannot do this for all features with a limited number of masks. For example, in Figure 1(a), since the three features are conflicting with each other, we cannot assign them to only two masks and one conflict still exits after decomposition. An obvious objective of decomposition is to minimize the number of remaining conflicts. For this purpose, stitch may be inserted to slice a feature into two pieces, so that they can be assigned to different masks. As shown in Figure 1(b), the conflict-free decomposition can be achieved with one stitch. However, stitch is manufacturing costly, so its number is preferred to be minimized.

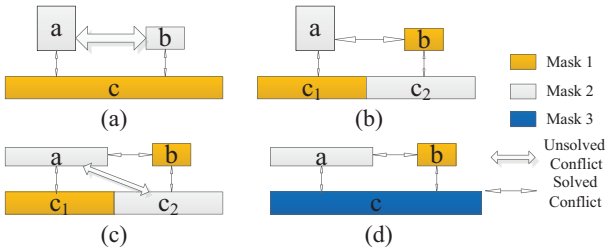

Figure 1: Multiple patterning lithograph

Double Patterning Lithography (DPL), the simplest form of multiple patterning lithography, decomposes a layout into two masks and repeats the exposure/etching process twice. DPL with conflict and stitch minimization has attracted years of extensive research. The most up-to-date result was contributed by Tang *et al.* [8], who presented an optimal method in polynomial time. However, because native conflicts may exist in a layout, DPL may result in a significant number of unsolvable conflicts. As Figure 1(c) shows, the conflict cannot be resolved even with stitch. Although DPL has made the sub-22nm production a reality, with further decrease of the minimum feature size, it is thought to get to its limit, especially for 14nm and beyond technology nodes. TPL that decomposes a layout into three instead of two masks and thus can deal with very dense and complex layouts to attain fewer stitches and conflicts, is a considerable substitute and a natural extension of DPL. The layout in Figure 1(c) can be easily decomposed to three masks without any stitches or conflicts, as shown in Figure 1(d).

There are not many researches on the layout decomposition for TPL. The problem was formulated as an ILP by Yu *et al.* in [14], which is the first systematic study on this topic. Because of the low scalability of ILP, they introduced a semidefinite programming approximation. However, Fang *et al.* [3] pointed out that since this work started

DAC'13, May 29 - June 07 2013, Austin, TX, USA.

with all the candidate stitches obtained by the projection method that is only appropriate for DPL, there is a high chance for them to miss legal TPL stitches. Thus a heuristic was proposed in [3], based on the assumption that the conflict related to a feature with higher maximum overlap density of projections is harder to be resolved by inserting stitch. However, this assumption may not be true in some cases and we will give a counter-example in Section 4.3 to illustrate this. Besides, in terms of stitch finding, the color scanning method used in [3] failed to reflect the requirement of overlap margin, which is important as stitch is very sensitive to overlay error [4, 6]. Most recently, Tian *et al.* [11] proposed a triple patterning algorithm based on standard cell libraries and row structure layout, which may not be applicable to a general layout. Since it adopted the same method in [14] to identify stitch positions, it has the same problem of missing stitches.

In this paper, we study the decomposition problem for TPL, our main contributions can be summarized as follows:

- We present a new stitch position identification method, which can find all legal stitch positions in TPL.
- We use graph simplification techniques to reduce the problem to some simple subproblems. We find that most of the subproblems fall into only a few geometric structures that can be solved, thus we can match them with those structures and solve them very efficiently.
- We propose an effective heuristic to solve those unmatched subproblems.
- We present a simple approach to identify native conflicts in a layout.
- Our decomposition approach performs very well for practical benchmarks, with remarkable reduction in conflict numbers and stitch numbers comparing with the most up-to-date result on this TPL decomposition problem.

The rest of the paper is organized as follows. Section 2 introduces some preliminaries for the TPL problem. Section 3 states the new stitch finding method. Section 4 describes the whole decomposition approach. Section 5 reports experimental results and Section 6 concludes this paper.

2. PRELIMINARIES

Preliminaries of layout decomposition for TPL are introduced in this section, including formal problem formulation and the differences between decomposition for DPL and decomposition for TPL.

2.1 Problem Formulation

As DPL decomposition is formulated as a two-coloring problem, we formulate the decomposition for TPL as a three-coloring problem. We use three colors, namely blue, gray and orange, to represent three different masks. Every feature in the layout is of the shape of a rectilinear polygon. Given a layout specified by features, a conflict graph can be constructed. Conflict graph is an undirected graph with nodes representing features in the layout, and an edge between two nodes means that the two corresponding features are within the minimum coloring spacing cs_{min} from each other. As Figure 2(b) shows, a conflict graph with four nodes can be constructed according to the layout in Figure 2(a). A feature can be split by inserting stitch. However, stitch insertion has

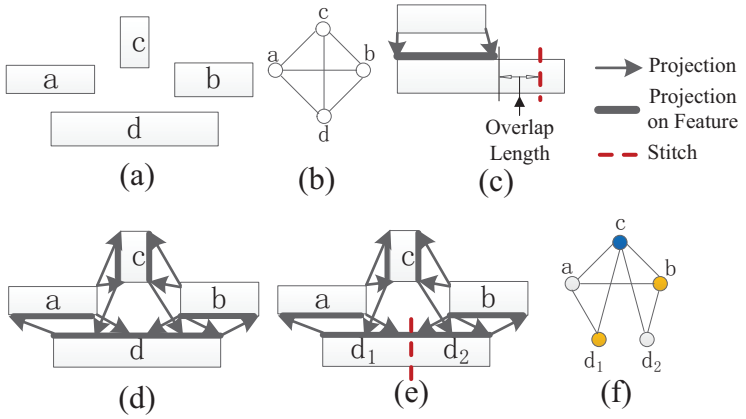

Figure 2: Conflict graph construction and stitch finding

two constraints: *overlap length* and *minimum feature size*. Overlap length (Figure 2(c)) is the length that a stitch position can move horizontally or vertically without causing any new conflicts between newly generated features and the others. The requirement is that the overlap length is not less than a threshold, called *overlap margin*. For the minimum feature size constraint, the size of the generated feature cannot be less than the minimum feature size. Now we can give the problem formulation of layout decomposition for TPL.

Problem 1. Given a layout, a minimum coloring spacing cs_{min}, an overlap margin m_o and a minimum feature size fs_{min}, our objective is to assign one mask out of three for each feature, while the numbers of conflicts and stitches are minimized and all the constraints are satisfied.

2.2 Comparisons Between DPL and TPL

Although with one extra mask, decomposition for TPL is much more complicated than that for DPL, mainly for the following reasons: 1. The minimum coloring spacing for TPL is typically larger than that for DPL, which leads to a conflict graph with more edges. 2. A graph without odd cycle is 2-colorable. However, to decide 3-colorability of a graph is NP-complete. 3. To 2-color a 2-colorable graph can be done efficiently, e.g., depth-first search, but coloring a 3-colorable graph with even four colors is NP-complete [5]. 4. All candidate stitches for DPL can be easily identified by projection, whereas the same method is not applicable for TPL, which has been proved in [3]. There is no general method for stitch finding for TPL in the literature to the best of our knowledge. 5. Native conflict, conflict that cannot be resolved even with stitch, can be identified in DPL as odd cycle in conflict graph with all candidate stitches, while this is not the case for TPL. It is still an open problem of how to identify native conflict in TPL.

3. STITCH FINDING

We present an example for which the DPL projection method fails to find legal TPL stitch. This kind of layout, forming a K_4 conflict graph, is among the most popular ones according to our observation. As Figure 2(d) shows, projections cover all features, thus no stitch is allowed if stitch is forbidden to overlap with any projection (as for DPL stitch). However, a stitch overlapping with the projections of c on

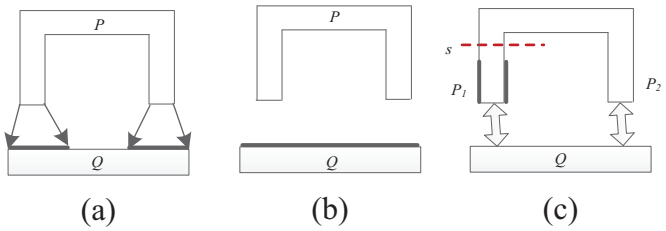

Figure 3: Projection merging and failed stitch position

d can break the complete graph into a 3-colorable one, as Figure 2(e) and 2(f) show. If a stitch is overlapped with the projection of a polygon, both of the two split nodes of the stitch will still conflict with that polygon. In this example, because the stitch is overlapped with the projection of c, both d_1 and d_2 will conflict with c after inserting the stitch.

Algorithm 1 Stitch Finding in TPL

Input: Polygon P, conflict graph G
Output: All candidate stitches on P
for Every pairwise edges a and b with the lengths of a and b not less than $2 * fs_{min}$ **do**
  for Every polygon Q conflicting with P **do**
    Calculate projections from Q onto a and b
  end for
  Segment *op(a,b)* according to the endpoints of all the projections
  for Every segmentation *SE* **do**
    Choose the position of stitch s within *SE* so that the distances between s and the projections on the two sides of *SE*, d_1 and d_2, are equal, split the polygon P and get two generated polygons
    if (1) no fs_{min} violations occur in the generated polygons **and** (2) both d_1 and d_2 are not less than m_o **and** (3) s is not near a corner **and** (4) neither of the generated polygons is such that it has no conflict edges **then**
      Store the stitch
    end if
  end for
end for

We first give some definitions for the terms used in our stitch finding method. The rectilinear polygon of a feature consists of one or more rectangles. Two parallel edges a and b of a polygon are called *pairwise edges* if they belong to the same rectangle. Note that projections on an edge from the same polygon may be separated (Figure 3(a)), but they need to be merged (Figure 3(b)), since the projection has actually covered the whole edge. The overlapping part of two pairwise horizontal (vertical) edges a and b in the x (y) direction is denoted as *op(a,b)*. Polygon cannot be split near a corner (distance within m_o) since corner stitch may cause significant side effect on printability [1, 6]. The pseudocode of our stitch finding method is listed in Algorithm 1. Here we prove the correctness of our method.

OBSERVATION 1. *Within a segment in Algorithm 1, every point has the same property of overlapping with projections from other polygons (the property refers to the projections it overlaps with), it will thus result in the same effect by splitting at any point in a segment, if no stitch constraints are considered.*

THEOREM 1. *The stitches found by Algorithm 1 are complete.*

Proof: According to Observation 1, Algorithm 1 can exhaust all possible stitch positions by checking every segment delimited by the endpoints of the projections. Besides, by checking conditions (1)-(3), it can prune away stitches that violate constraints about m_o, fs_{min} or corner stitch. By checking condition (4), it can prune away useless stitches that will not help to resolve any conflict. (The reason is that adding isolated nodes into a conflict graph will not affect the 3-colorability of the graph). Therefore, we can conclude that Algorithm 1 can find all legal and effective candidate TPL stitches. Q.E.D.

Note that we have to test the conflict properties of the two generated polygons directly, instead of inferring them from the projections on the two sides of the stitch, because of the existence of U-shape polygon. As Figure 3(c) shows, both the generated polygon P_1 and P_2 conflict with Q if we split at s, although the projections of Q only appear on the lower side of s.

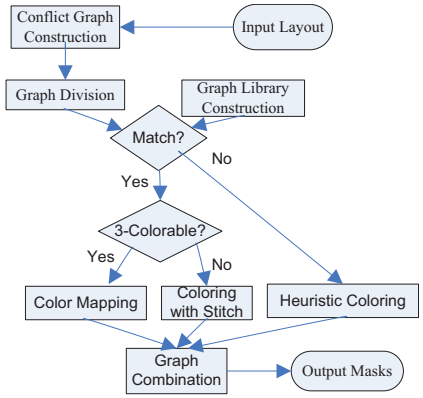

Figure 4: The flow of the decomposition approach

4. THE DECOMPOSITION APPROACH

In this section, we give details of our decomposition approach for TPL.

4.1 Overview

The flow of our decomposition approach for TPL is shown in Figure 4. First, conflict graph is constructed according to the input layout. Then, five graph simplification techniques are applied to divide the graph into a set of simple graphs. A graph library consisting of all possible graphs (with some constraints) of a limited number of nodes has been constructed beforehand. After that, we try to match all the simple graphs with the graphs in the library. For the matched ones, if they are 3-colorable, we simply map the colors from the graphs in the library to them; otherwise we will color them after stitch insertion. For the unmatched ones, we will color them by an effective heuristic as described below.

4.2 Graph Division

We construct the conflict graph, as shown in Section 2.1, and incorporate five graph decomposition techniques to divide the constructed graph. Three of which were proposed

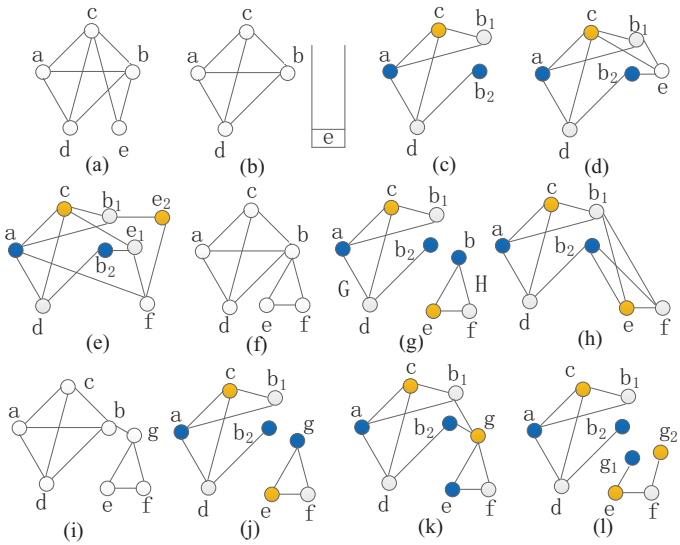

Figure 5: Examples for graph division

in [14]: *Independent Component Computation, Nodes with Degree Less than Three Removal* and *2-Edge-Connected Component Computation*. The other two were proposed in [3]: *Biconnected Component Computation and 3-Edge-Connected Component Computation*. In the following, we will describe them briefly and point out their potential problems.

4.2.1 Independent Component Computation

Independent component of a graph is a maximal subgraph in which any two nodes are connected by at least one path. Independent component computation has been applied to many previous studies [4, 13, 15]. It is obvious that two independent components are coloring-independent.

4.2.2 Nodes with Degree Less than Three Removal

When we 3-color a graph, nodes with degree less than three can be removed temporarily and pushed to a stack. We can continue removing until every remaining node is with degree at least three. After coloring these remaining nodes, we can pop out all the nodes in the stack one by one, color them and add them back to the graph. Since every node has at most two neighbours when it is removed, there is at least one available color for it when it is added back. If all the nodes of a graph are removed, the graph can be colored easily. This method was adopted to reduce the graph in [14]. However, when some nodes of the graph cannot be removed, the removal process is not guaranteed to be safe. As Figure 5(a) and 5(b) show, node e is removed and pushed to the stack. The remaining nodes are colored with a stitch inserted into b that splits b into two nodes (Figure 5(c)). After that, the degree of node e is increased to 3 when it is being added back, and there is no available color for it (Figure 5(d)). A possible solution is to insert a stitch into e (Figure 5(e)) but this may create chain effect to other nodes like node f in Figure 5(e).

4.2.3 Biconnected Component Computation

A node is called an articulation node if its removal disconnects the graph. Biconnected component of a graph is a maximal connected subgraph without articulation nodes. Biconnected components can be 3-colored independently and combined together with color rotation. This method may also be unsafe if stitch is inserted into the articulation node in at least one of the two components. As Figure 5(f) and 5(g) show, articulation node b is split in component G. After that, we will fail to combine the two components. As Figure 5(h) shows, there will be conflict if we just insert a stitch into the combined articulation node.

4.2.4 2-edge-connected and 3-edge-connected Components Computations

An edge is called a bridge if its removal disconnects the graph. A 2-edge-connected component of a graph is a maximal connected subgraph without bridges. A pair of edges is called a cut-pair if its removal disconnects the graph. A 3-edge-connected component of a 2-edge-connected graph is a maximal connected subgraph without cut-pairs. Both 2-edge-connected component and 3-edge-connected component can be 3-colored independently and combined together with color rotation to make the colors of the two nodes connected by the bridge or the cut-pair different. These two computations are safe if at most one of the two nodes connected by the bridge or an edge of the cut-pair is split. For example, two components of the graph in Figure 5(i) are computed and colored with a stitch inserted into node b, as shown in Figure 5(j). The two components can be combined with color rotation to make the color of node g different from the colors of nodes b_1 and b_2, as shown in Figure 5(k). However, if both nodes connected by the bridge or the cut-pair are split, the two components may not be able to be combined. As shown in Figure 5(l) (obtained from Figure 5(i)), both nodes b and g are split and combining them will cause a conflict.

4.2.5 Flow of Graph Division

The flow of our graph division is shown in Figure 6. First, we compute independent components, and remove nodes with degree less than three in every component. As this node removal can further disconnect some components, we will apply the independent component computation again. Then we compute 2-edge-connected components, biconnected components and 3-edge-connected components one after another. Note that endpoints of a bridge are also articulation nodes. For example, in Figure 5(i), nodes b and g, which are endpoints of bridge *(b,g)*, are also articulation nodes. Recall that both nodes b and g cannot be split (by inserting stitch) if we divide the graph with these articulation nodes, whereas one of them can be split if we divide the graph with the bridge *(b,g)*. Therefore we will compute 2-edge-connected components first. These component computations may generate nodes with degree one or two. For example, the degree of node g in Figure 5(j) becomes two after the 2-edge-connected component computation. Therefore we will always try to remove nodes with degree less than 3 again after each computation.

We compute independent components with depth-first search. Bridges, 2-edge connected components, articulation nodes and biconnected components are computed according to the algorithms in [9, 10]. Cut-pairs and 3-edge connected components are computed by the algorithm in [12]. All of them can be done in linear time $O(|E|+|V|)$, where $|E|$ and $|V|$ are the numbers of edges and the number of nodes in the conflict graph $G=(E,V)$ respectively.

4.3 Graph Library Construction

We construct a graph library that contains all biconnected graphs with four, five or six nodes according to the algo-

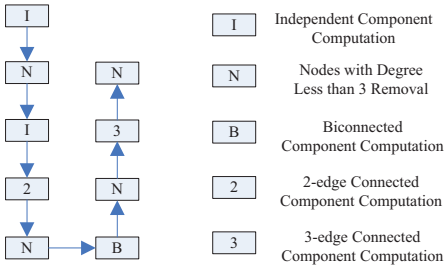

Figure 6: Flow of graph division

rithm in [7], and remove those with nodes of degree less than three. There are 23 such graphs in total. Obviously, all of them have no bridges nor cut-pairs. Nine of them can be 3-colored easily[1]. For example, the graphs in Figure 7(a) and 7(b) are 3-colorable ones in the library with five and six nodes respectively. For those non-3-colorable ones, it is also easy to decide which nodes need to split (by inserting stitch). For example, the graph in Figure 7(c) is not 3-colorable, but it can be 3-colored with a stitch properly inserted into a degree-4 node, such as node a, to split it into nodes a_1 and a_2 as shown in Figure 7(d). Whether a stitch can be properly inserted depends on which nodes a_1 and a_2 are connected with. For example, if node a in Figure 7(c) is split to give Figure 7(e), the graph is still not 3-colorable. Some graphs need two or more stitches to be 3-colorable. For example, the graph in Figure 7(f) is 3-colorable with two stitches inserted into nodes a and b respectively. With this library, after dividing a conflict graph, the subgraphs obtained will be matched with the graphs in the library. From the experiments, we found that almost all the subgraphs will be covered. For each matched subgraph, we will get the splitting and coloring information from the library directly. The library, once constructed, can be used for all benchmarks.

Here we present an example to show that the assumption in [3] is not always true. Consider Figure 7(g) that is an actual layout of the conflict graph in Figure 7(c). In Figure 7(g), different projections are shown with different colors. When we consider the conflict between e and a, the maximum overlap density of a is 3, since all the projections from b, c and d overlap with each other on a. Similarly, when we consider the conflict between e and b, the maximum overlap density of b is 2. According to the assumption, splitting b is better than splitting a. However, by splitting a, we can get a one-stitch solution (Figure 7(d)), but no matter how we split b, the graph remains non-3-colorable (Figure 7(h)).

4.4 Graph Matching and Coloring

We adopt a polynomial-time algorithm on graph isomorphism in [2] to match graphs. If a subgraph is matched with a 3-colorable one, we can color it simply by mapping the colors according to the nodes rearrangement. Otherwise, we will try inserting stitches. We will first try all possible 1-stitch solutions. If it fails, we will try to solve it with more stitches.

Because the graph division may not be safe, as explained in Section 4.2, we *disallow* stitches to be inserted at articulation nodes and at both endpoints of bridges or cut-pairs. Besides,

[1]We observed from experiments that these 3-colorable components cover most of the subgraphs obtained after dividing the conflict graphs.

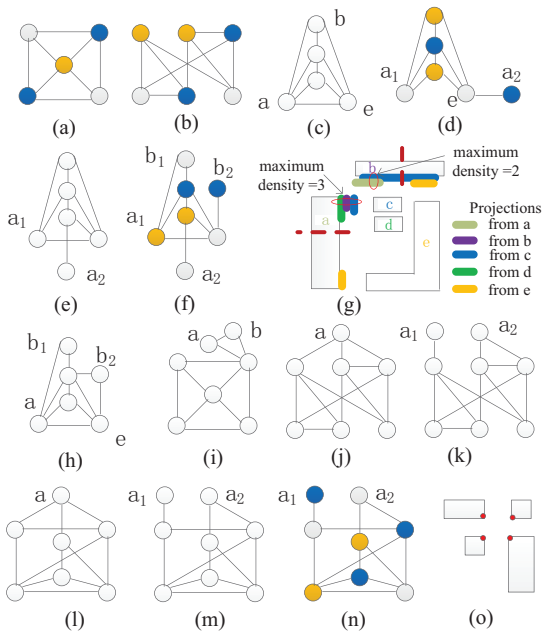

Figure 7: Examples for coloring of graphs in the library, heuristic coloring and native conflict

if the degree of a node is two when it is being removed in the "*nodes with degree less than three removal*" process, we will not insert a stitch that overlaps with its projection. For example, we will not insert a stitch to split node b into nodes b_1 and b_2, as shown in Figure 5(c), because they both conflict with node e (note that node e has degree two when it is being removed). To achieve this, when we find stitches by projection onto a node a with algorithm 1, we also consider the projections of the removed nodes that have degree two and have conflict with a when being removed.

Note that the order of removing nodes will have some impacts on the result. As shown in Figure 7(i), for the two degree-two nodes, a and b, if a is removed first, the degree of a when it is being removed is two, whereas if b is removed first, the degree is one. In our current implementation, we will just follow the order of the features given in the benchmark.

4.5 Heuristic Coloring

Subgraphs with seven or more nodes will not be matched and we use a heuristic to color them. We observed that there are not many such cases and the subgraphs are typically sparse, i.e., many nodes have low-degree. In our heuristics, we will try to split (by inserting stitches) those low-degree nodes in order to generate nodes with degree less than three. Since nodes with degree less than three can be removed temporarily, we may be able to convert the graph to one in the library, or simply all the nodes are removed step by step (because of degree less than three) and thus can be colored easily. An example is shown in Figure 7(j) and 7(k). We can split node a into two nodes a_1 and a_2. After the splitting and nodes removal, this graph is converted to the graph shown in Figure 7(b), which is 3-colorable. Another example is shown in Figure 7(l). Similarly, we can split node a into two nodes, as shown in Figure 7(m). After that, all nodes of the graph can be removed, thus we can 3-color it as shown in Figure 7(n). Similar to the coloring of matched subgraphs, we forbid unsafe stitches in our heuristics.

Table 1: Statistics and decomposition results

	Statistics						From [11]		From [3]				Ours			
Circuit	G#	M#	Ratio	3C#	Ratio	NC	C#	S#	C#	S#	Cost	cpu(s)	C#	S#	Cost	cpu(s)
C432	4	4	1.000	0	0.000	0	-	-	0	6	6	0.01	0	4	4	0.01
C499	3	3	1.000	3	1.000	0	0	0	0	0	0	0.01	0	0	0	0.01
C880	7	7	1.000	0	0.000	0	0	7	1	15	25	0.01	0	7	7	0.01
C1355	4	3	0.750	1	0.250	0	0	3	1	7	17	0.02	0	3	3	0.01
C1908	3	3	1.000	2	0.667	0	0	1	1	0	10	0.04	0	1	1	0.01
C2670	9	8	0.889	3	0.333	0	0	6	2	14	34	0.06	0	6	6	0.04
C3540	14	14	1.000	5	0.357	1	-	-	2	15	35	0.08	1	8	18	0.05
C5315	15	15	1.000	6	0.400	0	-	-	3	11	41	0.11	0	9	9	0.05
C6288	213	213	1.000	8	0.038	2	-	-	19	341	531	0.13	14	191	331	0.25
C7552	34	34	1.000	12	0.353	0	-	-	3	46	76	0.17	0	22	22	0.10
S1488	8	8	1.000	6	0.750	0	0	2	0	4	4	0.03	0	2	2	0.01
S38417	627	625	0.997	553	0.882	19	-	-	20	122	322	0.62	19	55	245	0.42
S35932	1831	1829	0.999	1746	0.954	44	-	-	46	103	563	2.13	44	41	481	0.82
S38584	1819	1818	0.999	1667	0.916	36	-	-	36	280	640	2.26	36	116	476	0.77
S15850	1576	1573	0.998	1443	0.916	34	-	-	36	201	561	2.14	34	100	440	0.76
Avg.	411.13	410.47	0.998	363.67	0.885	-	-	-	1.14	2.07	1.39	2.36	1.00	1.00	1.00	1.00

4.6 Native Conflicts

In order to understand the difficulty of each benchmark, we devised a simple method to compute a lower bound on the number of native conflicts. One common case of native conflict is the existence of a K_4 graph between four points in the layout (e.g., red corners in Figure 7(o)). It is obvious that this kind of conflict subgraph can never be resolved by inserting stitches.

5. EXPERIMENTAL RESULTS

We implemented the proposed approach in C++, on a 2.39 GHz Linux machine with 48 GB memory. To evaluate the approach, we tested the ISCAS-85 & 89 benchmarks provided by the authors of [14]. We used the same setting of cs_{min} as previous studies [3, 11]. Since the setting of m_o is not clearly stated in [3, 11], we set it to 10nm, which is the same as that in [14] and is believed to be the same m_o used by all these works since comparisons with each other have been shown in these papers.

The statistics and decomposition results of the benchmarks are shown in Table 1. G#, M#, 3C#, NC denote subgraph number, matched subgraph number, 3-colorable subgraph number and lower bound of native conflicts, respectively. It can be seen that most of the subgraphs can be matched and 3-colored and the total ratios of matched ones and 3-colorable ones are nearly 0.998 and 0.885 respectively. We compare our results with previous studies. C# denotes the number of conflicts, S# is for the number of stitches, cpu is the running time for the decomposition process and cost is computed as $10 \times C\# + S\#$ (same as in [3]) since conflicts typically cause much higher manufacturing cost. As shown in the table, we can always achieve as good as [11] for the benchmarks that they can solve and we have actually solved three more benchmarks (C432, C5315 and C7552). Note that the method in [11] will only give a solution when there is a non-conflict solution and that is why no solutions are reported for the other benchmarks. For the additionally solved benchmarks, since the work [11] considers candidate stitches using DPL projections, some of the possible TPL stitches are actually missed. Compared with [3], we can reduce the number of conflicts by 12%, number of stitches by 52%, and cost by 28%. Note that our numbers of conflicts are actually very close to the native conflict numbers. For running time, we can achieve a 2.36× speed-up, which clearly demonstrates our efficiency.

6. CONCLUSION

In this paper, we propose an approach of decomposition for TPL. Based on graph division and matching, our approach can be very efficient. To solve unmatched subgraphs, we propose an effective heuristic. Our approach can find all legal stitches in TPL to resolve more conflicts. Experimental results verify its effectiveness. We expect this result to benefit the industry on TPL. In the near future, we will study on a good ordering of removing nodes with degree less than three in order to give better results.

7. REFERENCES

[1] D. Abercrombie, P. Lacour, O. El-Sewefy, A. Volkov, K. Arb, C. Reid, Q. Li, and P. Ghosh. Double patterning from design enablement to verification. In *Pro. SPIE*, 2011.

[2] A. Dharwadker and J. Tevet. The graph isomorphism algorithm. In *Pro. The Structure Semiotics Research Group S.E.R.R.*, 2009.

[3] S. Fang, Y. Chang, and W. Chen. A novel layout decomposition algorithm for triple patterning lithography. In *Pro. DAC*, 2012.

[4] A. B. Kahng, C. H. Park, X. Xu, and H. Yao. Layout decomposition for double patterning lithography. In *Pro. ICCAD*, pages 465–472, 2008.

[5] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Pro. Israel Symposium on the Theory and Computing Systems*, pages 250–260, 1993.

[6] Q. Ma, H. Zhang, and D. F. Wong. Triple patterning aware routing and its comparision with double patterning aware routing in 14nm technology. In *Pro. DAC*, 2012.

[7] D. Stolee. Isomorph-free generation of 2-connected graphs with applications. *CSE Technical Reports, University of Nebraska - Lincoln*, 2011.

[8] X. Tang and M. Cho. Optimal layout decomposition for double patterning technology. In *Pro. ICCAD*, 2011.

[9] R. Tarjan. Depth-first search and linear graph algorithms. In *Pro. SWAT*, pages 114–121, 1971.

[10] R. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 40:125–142, 2007.

[11] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and D. F. Wong. A polynomial time triple patterning algorithm for cell based row-structure layout. In *Pro. ICCAD*, 2012.

[12] Y. H. Tsin. A simple 3-edge-connected component algorithm. *Theory of Computation Systems*, 2:160–161, 1974.

[13] J.-S. Yang, K. Lu, M. Cho, K. Yuan, and D. Pan. A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography. In *Pro. ASPDAC*, 2010.

[14] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan. Layout decomposition for triple patterning lithography. In *Pro. ICCAD*, 2011.

[15] K. Yuan, J.-S. Yang, and D. Pan. Double patterning layout decomposition for simultaneous conflict and stitch minimization. In *Pro. ISPD*, 2009.