Layout Decomposition for Triple Patterning Lithography

Bei Yu*, Kun Yuan†, Boyang Zhang*, Duo Ding*, David Z. Pan*
* ECE Dept. University of Texas at Austin, Austin, TX USA 78712
† Cadence Design Systems, Inc., San Jose, CA USA 95134
Email: {bei, dpan}@cerc.utexas.edu

***Abstract*—As minimum feature size and pitch spacing further decrease, triple patterning lithography (TPL) is a possible 193nm extension along the paradigm of double patterning lithography (DPL). However, there is very little study on TPL layout decomposition. In this paper, we show that TPL layout decomposition is a more difficult problem than that for DPL. We then propose a general integer linear programming formulation for TPL layout decomposition which can simultaneously minimize conflict and stitch numbers. Since ILP has very poor scalability, we propose three acceleration techniques without sacrificing solution quality: independent component computation, layout graph simplification, and bridge computation. For very dense layouts, even with these speedup techniques, ILP formulation may still be too slow. Therefore, we propose a novel vector programming formulation for TPL decomposition, and solve it through effective semidefinite programming (SDP) approximation. Experimental results show that the ILP with acceleration techniques can reduce 82% runtime compared to the baseline ILP. Using SDP based algorithm, the runtime can be further reduced by 42% with some tradeoff in the stitch number (reduced by 7%) and the conflict (9% more). However, for very dense layouts, SDP based algorithm can achieve $140\times$ speed-up even compared with accelerated ILP.**

1. Introduction

As minimum feature size further scales, the semiconductor industry is greatly challenged of patterning sub-22nm half-pitch due to the delay of viable next generation lithography such as Extreme Ultra Violet (EUV). Double patterning lithography (DPL) is widely recognized as a promising solution for 32nm, 22nm, and possibly 16nm volume chip production.

As shown in Fig. 1, the key challenge of DPL lies in the decomposition process by which the original layout is divided into two masks. Then, there are two exposure/etching steps, through which the layout can be produced. The advantage of this approach is that the effective pitch can be doubled, which improves the lithography resolution. During the decomposition, when the distance between the two patterns is less than minimum colorable distance min_s, they need to be assigned to different masks to avoid a conflict. Sometimes conflict can be solved by splitting a pattern into two touching parts, called stitches. In Fig. 1(b), polygon a is split into two polygons a_1 and a_2 in order to resolve the decomposition conflicts. However the introduced stitches lead to yield loss due to overlay error [1]. Therefore, two of the main challenges in layout decomposition are conflict and stitch minimization.

The paradigm of double patterning may be further extended to triple patterning lithography (TPL). Industry has already explored the test-chip patterns with triple patterning or even quadruple patterning [2]. By using TPL, we can achieve further feature-size scaling through pitch-tripling.

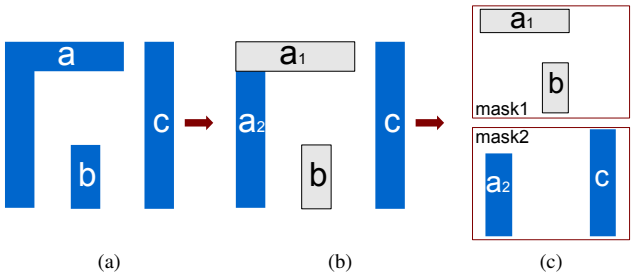

Fig. 1: In DPL, a single layer is decomposed into two masks and the pitch can be increased effectively.

Fig. 2: (a) In DPL, even stitch insertion can not avoid native conflict. (b) The native conflicts can be resolved by TPL.

It shall be noted that in DPL, even with stitch insertion, there may be native conflicts [3]. Fig. 2(a) shows a three-way conflict cycle between features a, b and c, where any two of them are within the min_s. As a consequence, there is no chance to produce a conflict-free solution using DPL decomposition. However, we can easily resolve this problem if the layout is decomposed into three masks as shown in Fig. 2(b). Yet this does not mean TPL layout decomposition problem becomes easier. Actually since the features can be packed closer, the problem turns out to be more difficult, as to be shown in Section 2.

Much previous research focuses on the double patterning layout decomposition problem, which is generally regarded as a two-coloring problem on a conflict graph. Integer Linear Programming (ILP) is adopted in [4][5] to minimize the stitch number and/or the conflict number. Xu et al. [6] propose an efficient graph reduction-based algorithm for stitch minimization, and Yang et al. [7] propose a fast min-cut based approach. A matching based decomposer is proposed to minimize both the conflict and the stitch numbers [8]. To resolve the native

978-1-4577-1400-9/11/$26.00 ©2011 IEEE

conflict, several works introduce layout modification to further minimize the conflict number [9][10][11]. However, layout modification may cause new problems, i.e., timing closure, hotspot.

Until now, there are very few investigations on TPL layout decomposition. [12] proposes a triple patterning coloring algorithm, which adopts SAT Solver. However, their work only deals with contact arrays, not general layout structures with wires, contacts, and so on. Besides, it does not involve any stitch minimization. [13][14] propose a self-aligned triple patterning (SATP) process to extend 193nm immersion lithography to half-pitch 15nm patterning. But the SATP process cannot insert any stitch, which would greatly constrain the possible layout patterns that are decomposable [15]. To our best knowledge, there is no study so far on layout decomposition on TPL for general layout styles.

In this paper, we propose the first systematic study on layout decomposition for triple patterning lithography. We first formulate a general ILP formulation for TPL layout decomposition to simultaneously minimize conflict and stitch. To improve scalability, we further propose three acceleration techniques without loss of solution quality: layout graph simplification, independent component computation and bridges computation. A semidefinite programming based approximation algorithm is further proposed to improve scalability. Semidefinite programming is an extension of linear programming to approximately solve NP-hard problems and it has been successfully applied to many combinatorial problems [16][17]. Our main contributions of this paper include:

- General ILP formulation to simultaneously minimize conflict and stitch for TPL layout decomposition;
- Three acceleration techniques to improve ILP scalability;
- A novel vector programming formulation for TPL decomposition and its semidefinite programming based approximation algorithm which can further deal with very dense layouts where even accelerated ILP becomes too slow;
- Our experimental results are very promising in terms of quality of results and runtime tradeoff.

The rest of the paper is organized as follows: in Section 2, we discuss the problem formulation and then analyze problem complexity. The basic algorithm and some acceleration techniques are described in section 3. Section 4 proposes a semidefinite programming based algorithm to further accelerate the basic algorithm. Section 5 presents the experiment results, followed by conclusion in Section 6.

2. Problem Formulation and Complexity

Some preliminaries on TPL are provided in this section, including some definitions and the problem formulation. We also demonstrate the complexity of the problem.

2.1. Problem Formulation

Given a layout which is specified by features in polygonal shapes, a layout graph [4] and a decomposition graph [9] are constructed.

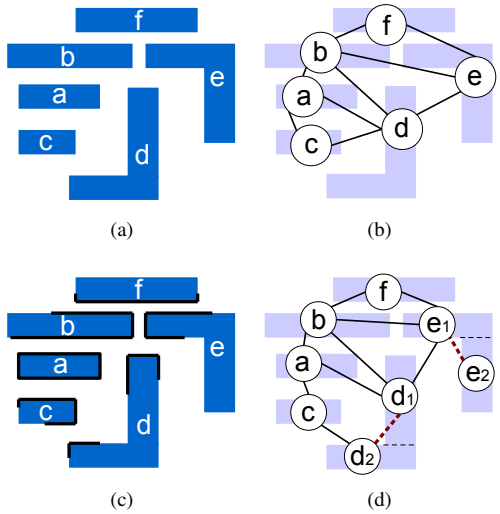

Fig. 3: Layout graph construction and decomposition graph construction (a) Input layout representing as irregular polygons. (b)Corresponding layout graph, where all edges are conflict edges. (c)The node projection. (d)Corresponding decomposition graph, where dash edges are stitch edges.

***Definition 1 (Layout Graph)**:* The *layout graph* (LG) is an undirected graph whose nodes are the given layout's polygonal shapes and where an edge exists if and only if the two polygonal shapes are within minimum coloring distance min_s of each other.

Fig. 3(a) gives an example of an input layout; the corresponding layout graph is shown in Fig. 3(b). All the edges in a layout graph are called Conflict Edges (CE). A conflict exists if and only if two nodes are connected by a CE and are in the same mask. In other words, each conflict edge is a conflict candidate.

***Definition 2 (Decomposition Graph)**:* Given a layout represented by a set of polygonal shapes, the *decomposition graph* (DG) is an undirected graph with a single set of nodes V, and two sets of edges, CE and SE, which contain the *conflict edges* and *stitch edges*, respectively. V has one or more nodes for each polygonal shape and each node is associated with a polygonal shape. An edge is in CE iff the two polygonal shapes are within minimum coloring distance min_s of each other. An edge is in SE iff there is a stitch between the two nodes which are associated with the same polygonal shape.

On the layout graph, the node projection is first performed, where projected segments are highlighted by bold lines in Fig. 3(c). Based on the projection result, all the legal splitting locations are computed. Then the decomposition graph is constructed, as shown in Fig. 3(d). Note that the conflict edges are marked as black edges, while stitch edges are marked as dash edges.

***Problem 1 (TPL layout decomposition)**:* Given a layout which is specified by features in polygonal shapes, the layout graph and the decomposition graph are constructed. Our goal

TABLE I: Notation

Notation used in Mathematical programming	
CE	set of conflict edges
SE	set of stitch edges.
V	the set of polygens.
r_i	the i_{th} layout polygons
x_i	variable denoting the coloring of r_i
c_{ij}	0-1 variable, $c_{ij} = 1$ when a conflict between r_i and r_j
s_{ij}	0-1 variable, $s_{ij} = 1$ when a stitch between r_i and r_j
Notation used in ILP formulation	
x_{i1}, x_{i2}	two 1-bit 0-1 variables to represents 3 colors
c_{ij1}, c_{ij2}	two 1-bit 0-1 variables to determine c_{ij}
s_{ij1}, s_{ij2}	two 1-bit 0-1 variables to determine s_{ij}

is to assign all the nodes in the decomposition graph to three masks to minimize the stitch number and the conflict number.

2.2. Problem Complexity

At first glance, the layout decomposition is similar to graph coloring problem. However, since stitch edges are introduced, the problem to minimize conflict and stitch is more complicated. For double patterning case, deciding whether a graph is 2-colorable is easy by determining if there exists odd cycles. For conflict and stitch minimization, if a decomposition graph is planar DPL layout decomposition can be solved in polynomial time [8]. In order to solve the triple patterning issue, the problem becomes more complicated.

Lemma 1: Deciding whether a *planar* graph is 3-colorable is NP-complete [18].

Lemma 1 can be naturally extended to general graph. Based on Lemma 1, the methodology in [12] is not suitable for TPL decomposition: SAT solver can only work for 3-colorable layout graph, which cannot be checked in polynomial time.

Lemma 2: Coloring a 3-colorable graph with 4 colors is NP-complete [19].

3-coloring problem is to assign the nodes in one 3-colorable graph to 3 colors. Since coloring the graph with 4 colors cannot be finished in polynomial time, it can be shown 3-coloring problem is NP-hard. Based on above lemmas, even the decomposition graph is planar, we reach the following theorem:

Theorem 1: **TPL layout decomposition problem is NP-hard.**

We can prove this theorem by reducing 3-Coloring problem to the TPL decomposition problem. Due to page limit, the detailed proof is skipped here.

3. Basic Algorithm

In this section, we will present our basic algorithms, which are based on the Integer Linear Programming (ILP). Since the timing complexity for ILP is very high, we propose three acceleration techniques to divide the whole problem into several smaller ones. The entire flow is shown in Fig. 4.

3.1. Mathematical Formulation for TPL Decomposition

The mathematical formulation for TPL layout decomposition is shown in (1). For convenience, some notations in mathematical programming and ILP formulation are listed in Table I. The objective is to simultaneously minimize both the conflict number and the stitch number. The parameter α is a user-defined parameter for assigning relative importance between the conflict number and the stitch number.

$$\min \sum_{e_{ij} \in CE} c_{ij} + \alpha \sum_{e_{ij} \in SE} s_{ij} \tag{1}$$

$$\text{s.t.} \quad c_{ij} = (x_i == x_j) \qquad \forall e_{ij} \in CE \tag{1a}$$

$$s_{ij} = x_i \oplus x_j \qquad \forall e_{ij} \in SE \tag{1b}$$

$$x_i \in \{0, 1, 2\} \qquad \forall i \in V \tag{1c}$$

where x_i is a variable for the three colors of rectangles r_i, c_{ij} is a binary variable for conflict edge $e_{ij} \in CE$ and s_{ij} is a binary variable for stitch edge $e_{ij} \in SE$. Constraint $(1a)$ is used to evaluate the conflict number when touch nodes r_i and r_j are assigned different colors (masks). Constraint $(1b)$ is used to calculate the stitch number. If node r_i and node r_j are assigned the same color (mask), stitch s_{ij} is introduced.

3.2. ILP Formulation for TPL Layout Decomposition

We will now show how to formulate (1) with Integer Linear Programming. Note that eqs. $(1a)$ and $(1b)$ can be linearized only when x_i is a 0-1 variable [4], which cannot represent three different colors. To handle this problem, we represent the color of each node using two 1-bit 0-1 variables x_{i1} and x_{i2}. In order to limit the number of colors for each node to 3, for each pair (x_{i1}, x_{i2}) the value $(1, 1)$ is not permitted. In other words, only values $(0, 0), (0, 1)$ and $(1, 0)$ are allowed.

Thus, (1) can be formulated as follows:

$$\min \sum_{e_{ij} \in CE} c_{ij} + \alpha \sum_{e_{ij} \in SE} s_{ij} \tag{2}$$

$$\text{s.t.} \quad x_{i1} + x_{i2} \leq 1 \tag{2a}$$

$$x_{i1} + x_{j1} \leq 1 + c_{ij1} \qquad \forall e_{ij} \in CE \tag{2b}$$

$$(1 - x_{i1}) + (1 - x_{j1}) \leq 1 + c_{ij1} \qquad \forall e_{ij} \in CE \tag{2c}$$

$$x_{i2} + x_{j2} \leq 1 + c_{ij2} \qquad \forall e_{ij} \in CE \tag{2d}$$

$$(1 - x_{i2}) + (1 - x_{j2}) \leq 1 + c_{ij2} \qquad \forall e_{ij} \in CE \tag{2e}$$

$$c_{ij1} + c_{ij2} \leq 1 + c_{ij} \qquad \forall e_{ij} \in CE \tag{2f}$$

$$x_{i1} - x_{j1} \leq s_{ij1} \qquad \forall e_{ij} \in SE \tag{2g}$$

$$x_{j1} - x_{i1} \leq s_{ij1} \qquad \forall e_{ij} \in SE \tag{2h}$$

$$x_{i2} - x_{j2} \leq s_{ij2} \qquad \forall e_{ij} \in SE \tag{2i}$$

$$x_{j2} - x_{i2} \leq s_{ij2} \qquad \forall e_{ij} \in SE \tag{2j}$$

$$s_{ij} \geq s_{ij1}, s_{ij} \geq s_{ij2} \qquad \forall e_{ij} \in SE \tag{2k}$$

The objective function is the same as that in (1), which minimizes the weighted summation of the conflict number and the stitch number. Constraint $(2a)$ is used to limit the number of colors for each node to 3.

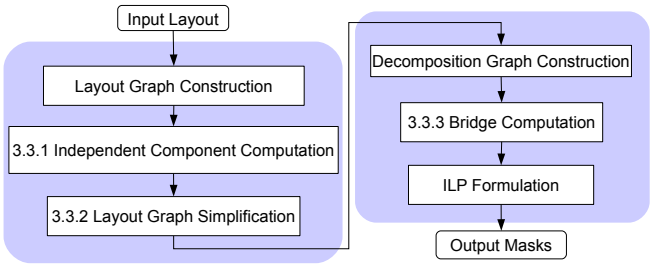

Fig. 4: Basic Algorithms Flow

Constraints $(2b)$ to $(2f)$ are equivalent to constraint $(1a)$, where 0-1 variable c_{ij1} demonstrates whether x_{i1} equals to x_{j1}, and c_{ij2} demonstrates whether x_{i2} equals to x_{j2}. 0-1 variable c_{ij} is true only if two nodes connected by conflict edge e_{ij} are in the same color, e.g. both c_{ij1} and c_{ij2} are true.

Similarly, constraints $(2g)$ to $(2k)$ are equivalent to constraint $(1b)$. 0-1 variable s_{ij1} demonstrates whether x_{i1} is different from x_{j1}, and s_{ij2} demonstrates whether x_{i2} is different from x_{j2}. Stitch s_{ij} is true if either s_{ij1} or s_{ij2} is true.

3.3. Acceleration Techniques

Since ILP is an NP-hard problem, its runtime increases dramatically with the size of a decomposition graph. We propose three acceleration techniques to simplify the layout graph and the decomposition graph in order to reduce the time complexity of ILP. As shown in Fig.4, our acceleration flow consists of three steps: Independent Component Computation, Layout Graph Simplification and Bridges Computation.

3.3.1) Independent Component Computation: We propose independent component computation on the decomposition graph to reduce the ILP problem size without losing optimality. In a layout graph of real design, we observe many isolated clusters. Therefore, we can break down the whole design into several independent components, and apply basic ILP formulation for each one. The overall solution can be taken as the union of all the components without affecting the global optimality. The runtime of ILP formulation decreases dramatically with the reduction of variables and constraints, and the coloring assignment can be effectively accelerated. Independent component computation is a well-known technique which has been applied in many previous studies [4][5][7][9].

3.3.2) Layout Graph Simplification: We can simplify the layout graph by removing all nodes with degree less than or equal to two. At the beginning, all nodes with degree less than or equal to two are detected and removed temporarily from the layout graph. This removing process will continue until all the nodes are at least degree-three. The layout graph simplification algorithm is shown in Algorithm 1.

If all the nodes in the layout graph can be pushed onto the stack, Algorithm 1 can solve TPL layout decomposition optimally in linear time. As an example shown in Fig.5, every node can be pushed onto stack and finally be colored when is popped off. Note that even when some nodes cannot be simplified, layout graph simplification can reduce problem size dramatically. Additionally, we observe that this algorithm can also partition the layout graph into several sub-graphs.

Algorithm 1 Layout Graph Simplification and Color Assignment

Require: Layout Graph G to be simplified, stack S
1: **while** $\exists n \in G$ s.t. $degree(n) \leq 2$ **do**
2: $S.push(n)$;
3: $G.delete(n)$;
4: **end while**
5: Decomposition graph construction.
6: TPL layout decomposition for nodes not be simplified.
7: **while** $!S.empty()$ **do**
8: $n = S.pop()$;
9: $G.add(n)$;
10: Assign n a legal color.
11: **end while**

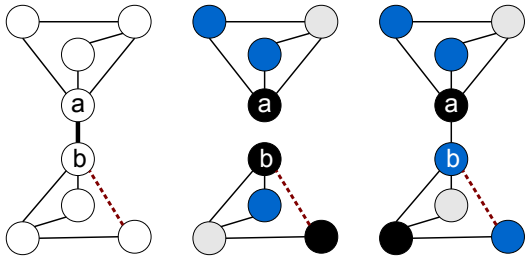

Fig. 6: Bridges Computation. (a) After bridges computation, label edge e_{ab} as bridge. (b) In two sub-graphs carry out ILP formulation. (c) Rotate colors in one sub-graph to add bridge.

3.3.3) Bridges Computation: A bridge of a graph is an edge whose removal disconnects the graph into two components. If the two components are independent, removing the bridge can divide the whole ILP into two independent ILP formulations.

Theorem 2: **Partitioning decomposition graph by removing bridges does not introduce new stitches.**

An example of the bridges computation is shown in Fig. 6. First of all, conflict edge e_{ab} is found to be a bridge. Removing the bridge divides the decomposition graph into two sides. After ILP based color assignment, if node a and node b are assigned the same color, we can rotate colors of all nodes in one sub-graph. Similar method can be adopted when bridge is a stitch edge. We adopt an $O(|V| + |E|)$ algorithm [20] to find bridges in decomposition graph.

Using above three acceleration techniques, the ILP formulation can still achieve optimal solutions. In other words, our acceleration algorithms can keep optimality. Due to page limit, we skip the detailed discussion here.

4. SDP Based Algorithm

Although the accelerated algorithms can simplify the problem size in many ways, ILP may still be too slow for

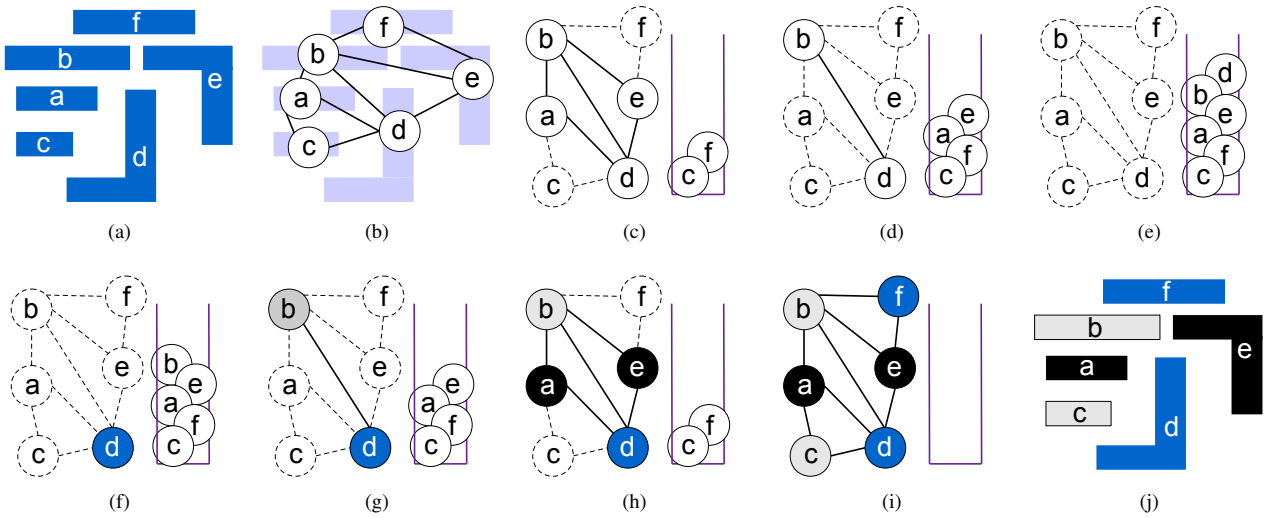

Fig. 5: This layout can be directly decomposed by layout graph simplification. (a) Input layout. (b) Corresponding layout graph. (c)(d)(e) Iteratively remove and push in nodes with edges no more than 2. (f)(g)(h)(i) Iteratively pop up and recover node, and assign any legal color. (j) Final decomposition result.

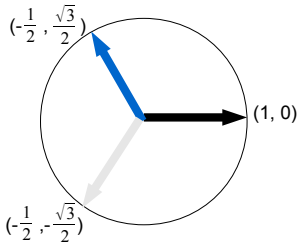

Fig. 7: Three vectors $(1,0),(-\frac{1}{2},\frac{\sqrt{3}}{2}),(-\frac{1}{2},-\frac{\sqrt{3}}{2})$ represent three different colors.

large problems which cannot be simplified effectively. In this section we provide an approximation algorithm to obtain more rapid solutions. First, a novel vector programming for TPL layout decomposition is formulated. Then we relax the vector programming into Semidefinite Programming (SDP). Given this solution from Semidefinite Programming, we can obtain the TPL decomposition results in polynomial time.

4.1. Vector Programming for TPL Layout Decomposition

In TPL decomposition, there are three possible colors. We set a unit vector $\vec{v_i}$ for every node i. If e_{ij} is a conflict edge, we want nodes $\vec{v_i}$ and $\vec{v_j}$ to be far apart. If e_{ij} is a stitch edge, we hope for nodes $\vec{v_i}$ and $\vec{v_j}$ to be the same. As shown in Fig. 7, we associate all the nodes with three different unit vectors: $(1,0),(-\frac{1}{2},\frac{\sqrt{3}}{2})$ and $(-\frac{1}{2},-\frac{\sqrt{3}}{2})$. Note that the angle between any two vectors of the same color is 0, while the angle between vectors with different colors is $2\pi/3$.

Additionally, we define the inner product of two m-dimension vectors $\vec{v_i}$ and $\vec{v_j}$ as follows:

$$\vec{v_i}\cdot\vec{v_j}=\sum_{k=1}^{m}v_{ik}v_{jk}$$

where each vector $\vec{v_i}$ can be represented as $(v_{i1},v_{i2},\dots v_{im})$. Then for the vectors $\vec{v_i},\vec{v_j}\in\{(1,0),(-\frac{1}{2},\frac{\sqrt{3}}{2}),(-\frac{1}{2},-\frac{\sqrt{3}}{2})\}$, we have the following property:

$$\vec{v_i}\cdot\vec{v_j}=\begin{cases}1, & \vec{v_i}=\vec{v_j}\\ -\frac{1}{2} & \vec{v_i}\neq\vec{v_j}\end{cases}$$

Based on the above property, we can formulate the TPL layout decomposition as the following vector program:

$$\min \sum_{e_{ij}\in CE}\frac{2}{3}(\vec{v_i}\cdot\vec{v_j}+\frac{1}{2})+\frac{2\alpha}{3}\sum_{e_{ij}\in SE}(1-\vec{v_i}\cdot\vec{v_j}) \quad (3)$$

$$\text{s.t.}\quad \vec{v_i}\in\{(1,0),(-\frac{1}{2},\frac{\sqrt{3}}{2}),(-\frac{1}{2},-\frac{\sqrt{3}}{2})\} \quad (3a)$$

Formula (3) is equivalent to mathematical formula (1). Since the TPL decomposition is NP-hard, this vector programming is also NP-hard. In the next part, we will relax (3) to semidefinite programming, which can be solved in polynomial time.

4.2. Semidefinite Programming Approximation

Constraint (3a) requires solutions of (3) be discrete. After removing this constraint, we generate formula (4) as follows:

$$\min \sum_{e_{ij}\in CE}\frac{2}{3}(\vec{y_i}\cdot\vec{y_j}+\frac{1}{2})+\frac{2\alpha}{3}\sum_{e_{ij}\in SE}(1-\vec{y_i}\cdot\vec{y_j}) \quad (4)$$

$$\text{s.t.}\quad \vec{y_i}\cdot\vec{y_i}=1,\quad \forall i\in V \quad (4a)$$

$$\vec{y_i}\cdot\vec{y_j}\geq-\frac{1}{2},\quad \forall e_{ij}\in CE \quad (4b)$$

This formula is a relaxation of (3) since we can take any feasible solution $\vec{v_i} = (v_{i1}, v_{i2})$ to produce a feasible solution of (4) by setting $\vec{y_i} = (v_{i1}, v_{i2}, 0, 0, \cdots, 0)$, i.e. $\vec{y_i} \cdot \vec{y_j} = 1$ and $\vec{y_i} \cdot \vec{y_j} = \vec{v_i} \cdot \vec{v_j}$ in this solution. Thus if Z_R is the value of an optimal solution of formula (4) and OPT is an optimal value of formula (3), it must satisfy: $Z_R \leq OPT$. In other words, solution of (4) is an approximation to that in (3). Since we only care about the value of $\vec{y_i}$, program (4) can be further simplified by eliminating the constants in the objective function:

$$\begin{aligned} &\min \sum_{e_{ij} \in CE} (\vec{y_i} \cdot \vec{y_j}) - \alpha \sum_{e_{ij} \in SE} (\vec{y_i} \cdot \vec{y_j}) \qquad (5) \\ &\text{s.t.} \quad (4a) - (4b) \end{aligned}$$

Without discrete constraint $(3a)$, programs (4) and (5) are not NP-hard now. To solve (5) in polynomial time, we will show that it is equivalent to a semidefinite programming. Semidefinite programming (SDP) is similar to linear programming which has a linear objective function and linear constraints. However, a square symmetric matrix of variables can be constrained to be positive semidefinite. Although semidefinite programs are more general than linear programs, both of them can be solved in polynomial time. Besides, the relaxation based on the semidefinite programming has better theoretical results than those based on LP [16].

Consider the following standard semidefinite program:

$$\begin{aligned} \text{SDP:} \quad \min \quad & A \bullet X \qquad (6) \\ & X_{ii} = 1, \quad \forall i \in V \qquad (6a) \\ & X_{ij} \geq -\frac{1}{2}, \quad \forall e_{ij} \in CE \qquad (6b) \\ & X \succeq 0 \qquad (6c) \end{aligned}$$

where $A \bullet X$ is the inner product between two matrices A and X, i.e. $\sum_i \sum_j A_{ij} X_{ij}$. Here A_{ij} is the entry that lies in the i-th row and the j-th column of matrix A. Constraint $(6c)$ means matrix X should be positive semidefinite.

$$A_{ij} = \begin{cases} 1, & \forall e_{ij} \in CE \\ -\alpha, & \forall e_{ij} \in SE \\ 0, & \text{otherwise} \end{cases} \qquad (7)$$

Similarly, X_{ij} is the i-th row and the j-th column entry of X. Note that the solution of SDP is represented as a positive semidefinite matrix X, while solutions of vector programming are stored in a list of vectors. However, we can show that they are equivalent.

Lemma 3: A symmetric matrix X is positive semidefinite if and only if $X = VV^T$ for some matrix V.

Given a positive semidefinite matrix X, using the Cholesky decomposition we can find corresponding matrix V in $O(n^3)$ time.

Theorem 3: **The semidefinite program (6) and the vector program (5) are equivalent.**

Proof: Given solutions of (5) $\{\vec{v_1}, \vec{v_2}, \cdots \vec{v_m}\}$, the corresponding matrix X is defined as $X_{ij} = \vec{v_i} \cdot \vec{v_j}$. In the other direction, based on Lemma 3, given a matrix X from (6), we can find a matrix V satisfying $X = VV^T$ by using the Cholesky decomposition. The rows of V are vectors $\{v_i\}$ that form the solutions of (5). ■

4.3. Mapping Algorithm

Solutions of program (6) are continuous, while optimal solutions in (3) are discrete. In this subsection we map the continuous solutions into discrete ones.

In the matrix X generated by SDP, if X_{ij} is close to 1, then nodes i and j should be in the same mask, while if X_{ij} is close to -0.5, node i and node j tend to be in different masks. Our mapping algorithm is given in Algorithm 2, which finds the relative relationships among the nodes and maps them into three different masks. First some triplets are constructed and sorted to store all X_{ij} information. Then, we carry out our mapping algorithm in two steps. In the first step, if X_{ij} is close to 1, the node i and the node j will be in the same mask, while if X_{ij} is close to -0.5, they will be labeled to be in different masks. Here the vectors UnionLevel[k] and SepaLevel[k] are some user defined parameters: UnionLevel[] are close to 1 and SepaLevel[] are close to -0.5. In the second step, we continue to union the node i and the node j with maximum X_{ij} until all nodes are assigned into three masks.

We use the disjoint-set data structure to group nodes into three masks. Implemented with "union by rank" and "path compression", the running time per operation of disjoint-set is almost constant [21]. Let n be the number of nodes, and the number of triplets is n^2. Sorting all the triplets requires $O(n^2 logn)$. Since all triplets are sorted, each of them can be visited at most once. Because the runtime of each operation can be finished almost in constant time, the complexity of Algorithm 2 is $O(n^2 logn)$.

Algorithm 2 Mapping Algorithm

```
1: Solve the program (6), get a matrix X.
2: Label each non-zero entry X_{i,j} as a triplet (X_ij, i, j).
3: sort all (X_ij, i, j) by X_ij.
4: for k = 1 to R do
5:    for each triple (X_ij, i, j) do
6:       if X_ij > UnionLevel[k] && Compatible(i, j) then
7:          Union(i, j);
8:       end if
9:    end for
10:   for each triple (x_ij, i, j) do
11:      if X_ij < SepaLevel[k] then
12:         Seperate(i, j);
13:      end if
14:   end for
15: end for
16: while Masks number > 3 do
17:   Pick triple with maximum X_ij and Compatible(i, j);
18:   Union (i, j);
19: end while
```

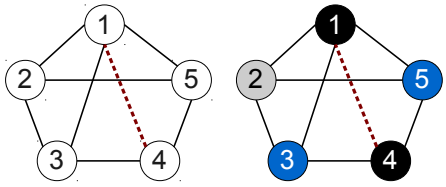

Fig. 8: Example to color decomposition graph. (a)Input decomposition graph. (b)Using semidefinite programming and Algorithm 2, we assign nodes into 3 different colors (masks).

4.4. An Example of the SDP Based Algorithm

Fig. 8 shows an example of the decomposition graph. This graph includes 7 conflict edges and 1 stitch edge, and can be colored with 3 colors. Moreover, the graph is not 2-colorable since it contains odd cycles. Here we show how the semidefinite programming can be used to solve TPL layout decomposition problem.

If we set $\alpha = 0.1$, then matrix A is as follow:

$$A = \begin{pmatrix} 0 & 1 & 1 & -0.1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ -0.1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

After solving the semidefinite programming (6), we can get a matrix X as following:

$$X = \begin{pmatrix} 1.0 & -0.5 & -0.5 & 1.0 & -0.5 \\ & 1.0 & -0.5 & -0.5 & -0.5 \\ & & 1.0 & -0.5 & 1.0 \\ & \cdots & & 1.0 & -0.5 \\ & & & & 1.0 \end{pmatrix}$$

here we only show the upper part of the matrix X.

From the matrix X we can find that node 1 and node 4 should be in the same color (because $X_{14} = 1.0$), and node 3 and node 5 should also be in the same color (because $X_{35} = 1.0$). However, since X_{12}, X_{13} and X_{15} are close to -0.5, nodes 2, 3 and 5 cannot be assigned in the same color as node 1. Using Algorithm 2, we can map all the nodes into three colors: $\{1, 4\}, \{2\}$ and $\{3, 5\}$. The final mapping result is shown in Fig. 8(b).

5. Experimental Results

We implement our algorithm in C++ and test it on an Intel Core 3.0GHz Linux machine with 32G RAM. OpenAccess2.2 [22] is used for interfacing with GDSII directly. We choose CBC [23] as our solver for the integer linear programming, and CSDP [24] as the solver for the semidefinite programming.

ISCAS-85 & 89 benchmarks are scaled down and modified to reflect the 16nm technology node. The metal one layer is used for experimental purposes, because it is one of the most complex layers in terms of layout decomposition. The minimum width and spacing become 25nm and 30nm. The minimum colorable distance is set as 85nm, and the minimum overlapping margin for stitch insertion is 10nm. The parameter α is set as 0.1.

5.1. Comparison

First, we show the effectiveness of the layout graph simplification and the bridges computation. Table II compares the Normal ILP and the Accelerated ILP, where the Normal ILP only uses independent component computation technique, while the Accelerated ILP uses all three acceleration techniques. Columns "SE#" and "CE#" denote the stitch edge number and conflict edge number respectively. From Table II we can see that layout graph simplification and bridges computation are quite effective: the stitch edge number can be reduced by 90%, while the conflict number can be reduced by 93%. The columns "st#" and "cn#" show the stitch number and the conflict number in the final design. "CPU(s)" is computational time in seconds. Compared with the Normal ILP, the Accelerated ILP can achieve the same results in around 18% of the runtime. Note that if no accelerative technique is used, the runtime for ILP is unacceptable even for small circuits like C432. From Table II we can see that both ILP formulations can achieve the optimal solutions, because of the same conflict number and stitch number.

Second, we verify the quality and efficiency of our approximation algorithm based on semidefinite programming. Table II also compares the Accelerated ILP and the SDP based algorithm, where "SDP Based" denotes the semidefinite programming based algorithm. Note that these two methods share the same decomposition graph, i.e. both the stitch edge number and the conflict edge number in their decomposition graph are equal. As we can see, using SDP based method the runtime can be further reduced by 42% and the stitch number can be reduced by 7%. The tradeoff for this acceleration is the 9% more conflicts.

5.2. Efficiency

In order to further evaluate the scalability of our SDP based method, we create four additional benchmarks (C1-C4) to test two decomposition algorithms on very dense layouts. Table III lists the comparison of the Speed-up ILP and the SDP based method on these very dense layouts. As we can see, compared with the Speed-up ILP, SDP based method can reduce stitch number by 10% while introduces 5% more conflicts. Furthermore, SDP based method can achieve $140\times$ speed-up. The reason for the dramatically acceleration is that: for a low density layout, the decomposition problem can be divided into many sub-problems, and typically each sub-problem contains no more than 20 nodes. While for high density layout, there are more nodes in each sub-problem, where SDP can be much faster than ILP.

6. Conclusion

In this paper, we propose a general integer linear programming (ILP) formulation for the TPL layout decomposition to simultaneously minimize the conflicts and stitches. To

TABLE II: Runtime and Performance Comparisons

Circuit	Comp#	Normal ILP					Accelerated ILP					SDP Based		
		SE#	CE#	st#	cn#	CPU(s)	SE#	CE#	st#	cn#	CPU(s)	st#	cn#	CPU(s)
C432	261	300	930	0	1	38.01	32	136	0	1	1.11	0	1	0.26
C499	418	566	2239	0	0	34.15	171	838	0	0	6.34	0	4	1.01
C880	516	844	2219	1	3	25.91	6	24	1	3	0.49	1	3	0.06
C1355	872	1008	2543	0	1	23.27	2	14	0	1	0.12	0	1	0.03
C1908	1132	1332	4480	0	1	48.66	1	14	0	1	0.10	0	1	0.03
C2670	1501	2186	7469	0	4	101.98	6	40	0	4	0.50	0	4	0.10
C3540	1964	3197	9283	2	2	3975.25	9	40	2	2	0.43	2	2	0.11
C5315	2767	4644	13211	5	0	173.86	20	70	5	0	0.70	5	0	0.18
C6288	3740	5319	11394	?	?	> 7200	259	509	9	72	26.05	9	72	1.36
C7552	4164	6591	19187	?	?	> 7200	64	180	10	6	2.19	7	9	0.46
S1488	588	1932	4284	0	1	73.37	31	54	0	1	0.50	0	1	0.1
S38417	9385	15912	40734	?	?	> 7200	1617	2724	3	19	20.56	3	21	9.33
S35932	23565	25252	71198	?	?	> 7200	3776	6317	3	18	49.87	3	22	33.55
S38584	24724	28808	74968	?	?	> 7200	3657	6066	4	26	44.16	4	26	34.52
S15850	20881	33134	87358	?	?	> 7200	3877	6504	6	34	49.18	6	39	35.92
avg.	-	8735	23433	-	-	-	901.8	1568.7	2.87	12.53	13.49	2.67	13.73	7.80
ratio	-	1	1	-	-	-	0.10	0.07	1	1	1	**0.93**	**1.09**	**0.58**

TABLE III: Comparison on Very Dense Layouts

Circuit	SE#	CE#	Accelerated ILP			SDP Based		
			st#	cn#	CPU(s)	st#	cn#	CPU(s)
C1	16	247	1	5	5.5	0	6	0.29
C2	38	289	0	15	17.32	0	16	0.77
C3	24	381	0	14	33.41	0	15	0.32
C4	56	437	9	32	203.17	9	32	0.49
avg.	-	-	2.5	16.5	64.9	2.25	17.3	0.468
ratio	-	-	1	1	1	**0.9**	**1.05**	**0.007**

improve scalability, we develop three acceleration techniques without losing solution quality: layout graph simplification, independent component computation and bridges computation. Furthermore, we propose a novel semidefinite programming algorithm to improve scalability for very dense layouts. Experimental results show that our methods are very effective. Since this is the first systematic attempt on TPL layout decomposition for general layouts, we expect to see a lot of researches as TPL may be adopted by industry in the near future.

REFERENCES

[1] J.-S. Yang and D. Z. Pan, "Overlay aware interconnect and timing variation modeling for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2008, pp. 488 –493.

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

[3] V. O. Anton, N. Peter, H. Judy, G. Ronald, and N. Robert, "Pattern split rules! a feasibility study of rule based pitch decomposition for double patterning," in *Proc. of SPIE*, 2007.

[4] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *ACM/IEEE International Conference on Computer Aided Design (ICCAD)*, 2008, pp. 465–472.

[5] K. Yuan, J.-S. Yang, and D. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," in *ACM International Symposium on Physical Design (ISPD)*, 2009, pp. 107–114.

[6] Y. Xu and C. Chu, "GREMA: graph reduction based efficient mask assignment for double patterning technology," in *ACM/IEEE International Conference on Computer Aided Design (ICCAD)*, 2009, pp. 601–606.

[7] 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 *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2010.

[8] Y. Xu and C. Chu, "A matching based decomposer for double patterning lithography," in *ACM International Symposium on Physical Design (ISPD)*, 2010, pp. 121–126.

[9] K. Yuan and D. Pan, "WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 32 –38.

[10] S.-Y. Chen and Y.-W. Chang, "Native-conflict-aware wire perturbation for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 556 –561.

[11] C.-H. Hsu, Y.-W. Chang, and S. R. Nassif, "Simultaneous layout migration and decomposition for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2009, pp. 595–600.

[12] C. Cork, J.-C. Madre, and L. Barnes, "Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns," in *Proc. of SPIE*, 2008.

[13] Y. Chen, P. Xu, L. Miao, Y. Chen, X. Xu, D. Mao, P. Blanco, C. Bencher, R. Hung, and C. S. Ngai, "Self-aligned triple patterning for continuous ic scaling to half-pitch 15nm," in *Proc. of SPIE*, 2011.

[14] B. Mebarki, H. D. Chen, Y. Chen, A. Wang, J. Liang, K. Sapre, T. Mandrekar, X. Chen, P. Xu, P. Blanko, C. Ngai, C. Bencher, and M. Naik, "Innovative self-aligned triple patterning for 1x half pitch using single "spacer deposition-spacer etch" step," in *Proc. of SPIE*, 2011.

[15] Q. Li, "NP-completeness result for positive line-by-fill sadp process," in *Proc. of SPIE*, 2010.

[16] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Rev.*, vol. 38, pp. 49–95, March 1996.

[17] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *J. ACM*, vol. 45, pp. 246–265, March 1998.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeye, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, pp. 237–267, 1976.

[19] S. Khanna, N. Linial, and S. Safra, "On the hardness of approximating the chromatic number," in *Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the*, Jun. 1993, pp. 250 –260.

[20] R. E. Tarjan, "A note on finding the bridges of a graph," *Information Processing Letters*, vol. 2, pp. 160–161, 1974.

[21] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. Cambridge, MA, USA: MIT Press, 1990.

[22] [Online]. Available: http://www.si2.org/?page=69

[23] [Online]. Available: http://www.coin-or.org/projects/Cbc.xml

[24] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613 – 623, 1999.