Layout Decomposition for Triple Patterning Lithography

Bei Yu, *Member, IEEE*, Kun Yuan, Duo Ding, and David Z. Pan, *Fellow, IEEE*

***Abstract*—As minimum feature size and pitch spacing further scale down, triple patterning lithography is a likely 193 nm extension along the paradigm of double patterning lithography for 14-nm technology node. Layout decomposition, which divides input layout into several masks to minimize the conflict and stitch numbers, is a crucial design step for double/triple patterning lithography. In this paper, we present a systematic study on triple patterning layout decomposition problem, which is shown to be NP-hard. Because of the NP-hardness, the runtime required to exactly solve it increases dramatically with the problem size. We first propose a set of graph division techniques to reduce the problem size. Then, we develop integer linear programming (ILP) to solve it. For large layouts, even with the graph-division techniques, ILP may still suffer from serious runtime overhead. To achieve better trade-off between runtime and performance, we present a novel semidefinite programming (SDP)-based algorithm. Followed by a mapping process, we can translate the SDP solutions into the final decomposition solutions. Experimental results show that the graph division can reduce runtime dramatically. In addition, SDP-based algorithm can achieve great speedup even compared with accelerated ILP, with very comparable results in terms of the stitch number and the conflict number.**

***Index Terms*—Graph division, integer linear programming (ILP), layout decomposition, semidefinite programming (SDP), triple patterning lithography (TPL).**

I. Introduction

AS THE feature size of semiconductor process technology nodes further scales-down, the industry is greatly challenged in printing sub-22 nm half-pitch patterns under the 193 nm lithographic wavelength. Double patterning lithography has been widely used by industry for 22 nm volume chip production [1]. For the 14-nm technology node development, there are several lithography candidates. Some candidates, such as electric beam lithography (EBL) and extreme ultraviolet (EUV), suffer from the delay due to some technical problems: EBL has a serious limitation due to low throughput, while EUV is challenged by tremendous technical barriers such as lack of power sources, resists, and defect-free masks [2], [3]. Triple patterning lithography, which is a natural extension from double patterning lithography, is one of most viable solutions for 14 nm node [4]. In addition, industry has already explored the test-chip patterns with triple patterning or even quadruple patterning [5].

In triple patterning lithography manufacturing process, there are three exposure/etching steps, through which the layout can be produced. The advantage of this process is that the effective pitch can be improved, which enhances the lithography resolution. Like in double patterning, the key challenge of triple patterning lithography is the layout decomposition, where input layout is divided into three masks. When the distance between two input features is less than minimum coloring distance $\min_s$, they need to be assigned to different masks to avoid a coloring conflict. Sometimes coloring conflict can be also resolved by inserting stitch to split a pattern into two touching parts. However, this introduces stitches, which lead to yield loss because of overlay error. Therefore, two of the main objectives in layout decomposition are conflict minimization and stitch minimization. An example of triple patterning layout decomposition (TPLD) is shown in Fig. 1, where all features in input layout are divided into three masks (colors).

In double patterning lithography, layout decomposition is generally regarded as a two-coloring problem [6]–[11]. A complete flow was proposed in [6] to optimize splitting locations with integer linear programming (ILP). Xu and Chu [8] provided an efficient graph reduction-based algorithm for stitch minimization. Yang *et al.* [9] and Tang and Cho [11] proposed min-cut-based approaches to reduce stitch number. To enable simultaneous conflict and stitch minimization, ILP was adopted by [6] and [7] with different feature preslicing techniques. A matching-based decomposer was proposed to minimize both the conflict number and the stitch number [10].

There are investigations on triple patterning aware design [12]–[14] and triple patterning layout decomposition [15]–[25]. Cork *et al.* [15] proposed a three-coloring algorithm adopting SAT formulation. Ghaida *et al*. [17] reused the double patterning techniques. For row-based layout design, Tian *et al*. [19], [21] presented polynomial time decomposition algorithms.

In this paper, we propose a systematic study on TPLD problem to simultaneously minimize conflict and stitch. Our contributions are highlighted as follows.

1) We prove that TPLD problem is NP-hard, therefore, to reduce the problem size, we propose a set of graph-division techniques.

Manuscript received January 14, 2014; revised May 22, 2014, July 25, 2014, and October 8, 2014; accepted December 17, 2014. Date of publication January 6, 2015; date of current version February 17, 2015. This work was supported in part by the NSF under Grant CCF-0644316 and Grant CCF-1218906, in part by SRC Task 2414.001, in part by the NSFC under Grant 61128010, and in part by IBM Scholarship. This paper was recommended by Associate Editor Prof. P. Gupta.

B. Yu and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78731 USA.

K. Yuan is with Facebook Inc., San Jose, CA 94025 USA.

D. Ding is with Oracle Corporation, Austin, TX 78727 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2014.2387840

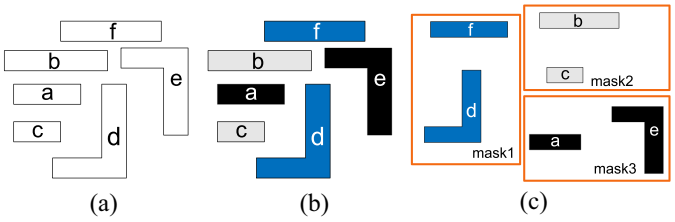


Fig. 1. Example of TPLD. (a) Input layout. (b) Features with different colors mean that they are assigned into different masks. (c) Features on each mask.

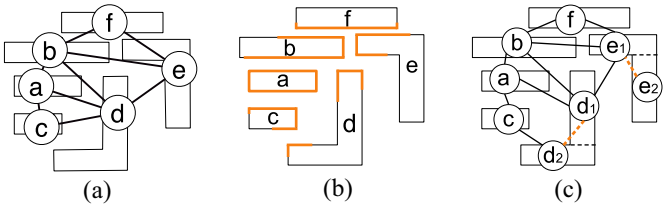


Fig. 2. Layout graph construction and DG construction. (a) Layout graph for given input, where all edges are conflict edges. (b) Vertex projection. (c) Corresponding DG, where dash edges are stitch edges.

2) We present an ILP formulation, which can achieve optimal solution theoretically, to assign color for each input feature.
3) To overcome the runtime overhead, we propose a novel vector programming, which can be relaxed to an effective semidefinite programming (SDP) formulation. We develop a mapping process to transform the SDP solution to the final TPLD results.
4) A post-stage conflict removal is proposed to further improve layout decomposition results.

The rest of this paper is organized as follows. Section II introduces the problem formulation, and the overall decomposition flow. Section III discusses how to generate stitch candidates for triple patterning. Section IV provides the general mathematical formulation and the ILP formulation, while Section V proposes a novel SDP-based algorithm to accelerate the basic algorithm. Section VI introduces some graph-division techniques to further reduce the problem size. Section VII presents the details of post conflict removal. Section VIII presents the experiment results, followed by the conclusion in Section IX.

II. Preliminaries

In this section, we provide some preliminaries on TPLD, including problem formulation, complexity analysis, and the introduction to our decomposition flow.

A. Problem Formulation

Given an input layout which is specified by features in polygonal shapes, at first a layout graph (LG) [6] is constructed by Definition 1.

Definition 1 (LG): A LG is an undirected graph whose vertex set represents polygonal shapes and edge set represents the connection if and only if two corresponding polygonal shapes are within minimum coloring distance $\min_s$.

Given an input layout, the corresponding LG is illustrated in Fig. 2(a). All the edges in a LG are called conflict edges. A conflict exists if and only if two vertices are connected by a conflict edge and are in the same mask. In other words, each conflict edge is a conflict candidate. On the LG, vertex projection [6] is performed, where projected segments are highlighted by bold lines in Fig. 2(b). Based on the projection result, all the legal splitting locations are computed. Then, a decomposition graph (DG) [26] is constructed by Definition 2.

Definition 2 (DG): A DG is an undirected graph with a single set of vertices V, and two sets of edges, conflict edges (CE) and stitch edges (SE), respectively. V has one or more vertices for each polygonal shape and each vertex is associated with a polygonal shape. An edge is in CE iff the two corresponding vertices are within minimum coloring distance $\min_s$. An edge is in SE iff there is a stitch between the two vertices which are associated with the same polygonal shape.

An example of DG is shown in Fig. 2(c). Note that the conflict edges are marked as black edges, while stitch edges are marked as dash edges. Here, each stitch edge is a stitch candidate. We define the TPLD problem as follows.

Problem 1 (Triple Patterning Layout Decomposition): Given a layout which is specified by features in polygonal shapes, the DG is constructed. TPLD assigns all the vertices of DG into one of three colors (masks) to minimize the costs of the stitches and the conflicts.

In this paper, we set the cost for each conflict is 1, and the cost for each stitch is α.

B. Problem Complexity

TPLD problem is an extension of double patterning layout decomposition (DPLD) problem, and both of them simultaneously minimize the conflict number and the stitch number. Xu and Chu [10] showed that if the DG is planar, DPLD can be resolved in polynomial time. At first glance, compared with DPLD, TPLD seems easier as there is one more color (mask). However, it turns out to harder. On one hand, since the goal of triple patterning is to achieve finer pitches, there will actually be more features to be packed closer to each other which will form multiway conflicts. That is, DGs for triple patterning will become much denser than those in double patterning. On the other hand, in double patterning the conflict detection (two-colorable) is equivalent to odd-cycles checking, which can be resolved in linear time through a breadth-first search. However, in triple patterning the conflict minimization, or even the conflict detection, is not straightforward.

To demonstrate the hardness of TPLD problem, we first introduce the planer graph 3-coloring (PG3C) problem. Given a planer graph, the PG3C problem is to assign three colors to all vertices. A conflict exists if and only if two vertices connected by an edge are in the color. The target of PG3C problem is to minimize the coloring conflict number. We have the following lemma.

Lemma 1: The PG3C problem is NP-hard.

The correctness of Lemma 1 stems from the conclusion that deciding whether a planar graph is 3-colorable is NP-complete [27]. For a planar graph, checking whether it is

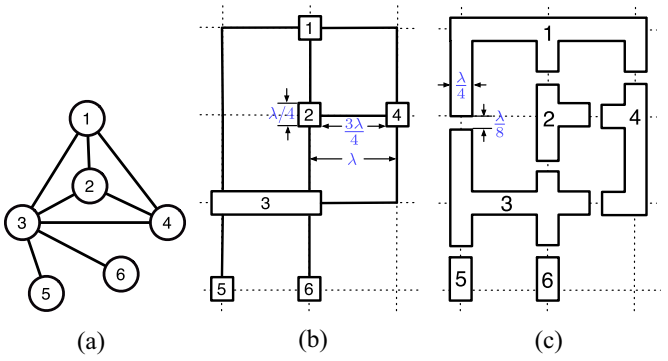


Fig. 3. Reducing PG3C to TPLD. (a) Instance of PG3C. (b) Transferred orthogonal drawing. (c) Corresponding TPLD instance.

three-colorable cannot be finished in polynomial time; therefore, three-coloring a planar graph with minimum cost cannot be finished in polynomial time.

Theorem 1: TPLD problem is NP-hard.

Proof: We prove this theorem by showing PG3C $\leq_P$ TPLD, i.e., PG3C can be reduced to TPLD. Given an instance of PG3C, its planar graph $G = (V, E)$ can be transferred to an orthogonal drawing [28], where the drawing plane is subdivided by horizontal and vertical gridlines of unit spacing λ. The vertices $\in V$ are represented by boxes occupying grid point with width $\lambda/4$. The edges $\in E$ are mapped to nonoverlapping paths in the gridlines. If each vertex is associated with a grid, the degree of each vertex should be at most four. For the vertices with degree more than four, they can be drawn as boxes occupying more than one grid point [29]. We construct the corresponding TPLD instance through the following two steps.

1) The width of each path is extended to $\lambda/4$.
2) Break each path in the middle through gap with length $\lambda/8$.

If we set $\min_s$ to $\lambda/8$, then we can get a TPLD instance, whose DG is isomorphic to the planar graph of PG3C instance. Since an orthogonal drawing can be constructed in polynomial time [28], the whole reduction can be finished in polynomial time. Thus, minimizing conflict number in the original PG3C instance is equal to minimizing conflict number in the constructed TPLD instance, which completes the proof. ■

For example, given a PG3C instance in Fig. 3(a), the corresponding orthogonal drawing and TPLD instance are illustrated in Fig. 3(b) and (c), respectively. Here, no stitch candidate is introduced.

C. Overall Decomposition Flow

The overall decomposition flow is illustrated in Fig. 4. First, we construct LG to translate the original layout into graph representations. Two graph division techniques are developed to the LG: independent component computation (ICC) and iterative vertex removal (IVR). Second, after vertex projection, we transform the LG into DG and propose two other graph division methods: bridge edge detection/removal and bridge vertex detection/duplication. Third, after these graph-based techniques, the DG is divided into a set of components.

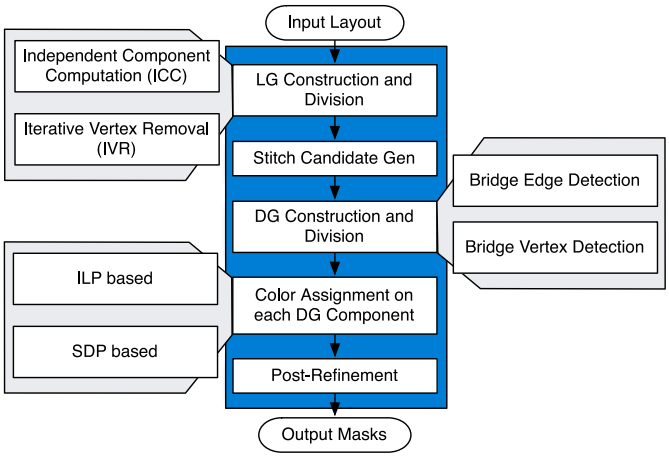


Fig. 4. Overview of our decomposition flow.

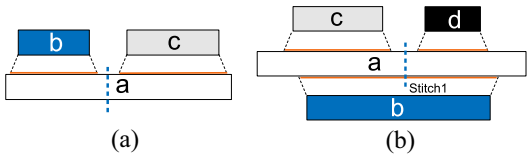


Fig. 5. Examples of (a) redundant stitch and (b) lost stitch.

To solve the color assignment on each DG component, two approaches are proposed. One is based on ILP, which can resolve the problem exactly, but it may suffer from runtime overhead. Another one is SDP-based algorithm: instead of using ILP, we formulate the problem into a vector programming, then its relaxed version can be resolved through SDP. Followed by a mapping stage, the SDP solution can be translated into a color assignment solution. At last, we merge all DG components together to achieve the final TPLD result.

III. Stitch Candidate Generation

Stitch candidate generation is one of the most important steps to parse a layout, as it not only determines the vertex number in the DG, but also affects the decomposition result. We use DP candidates to represent the stitch candidates generated by all previous double patterning research. Kahng *et al.* [6] and Xu and Chu [8] proposed different methodologies to generate the DP candidates. DP candidates may lose some legal candidates, thus they cannot be directly applied in TPLD problem [17], [18], [20]. Besides, we will show that DP candidates may be redundant. In this section, we provide a procedure to generate appropriate stitch candidates for triple patterning lithography.

A. Limitations of DP Candidates

We provide two examples to demonstrate that DP candidates are not appropriate for triple patterning. First, because of an extra color choice, some DP candidates may be redundant. As shown in Fig. 5(a), the stitch can be removed because no matter what color is assigned to features *b* and *c*, the feature *a* can always be assigned a legal color. We denote this kind

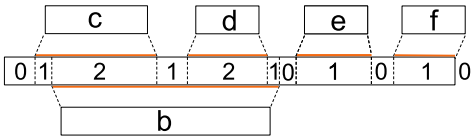


Fig. 6. Projection sequence of the feature is 01212101010, and the last 0 is a default terminal zero.

of stitch as a redundant stitch. After removing these redundant stitches, some extra vertices in the DG can be merged. In this way, we can reduce the problem size. Besides, DP candidates may cause the stitch loss problem, i.e., some useful stitch candidates cannot be detected and inserted in layout decomposition. In DPL, the stitch candidate has one precondition: it cannot intersect with any projection. For example, as shown in Fig. 5(b), because this stitch intersects with the projection of feature b, it cannot belong to the DP candidates. However, if features b, c, and d are assigned with three different colors, only introducing this stitch can resolve the conflict. In other words, the precondition in DPL limits the ability of stitches to resolve the triple patterning conflicts and may result in unnoticed conflicts. We denote the useful stitches forbidden by the DPL precondition as a lost stitch.

B. Stitch Candidate Generation for TPL

Given the projection results, we propose a new stitch candidate generation. Compared with the DP candidates, our methodology can remove some redundant stitches and systematically solve the stitch loss problem. For a better explanation, we define the projection sequence as follows.

Definition 3 (Projection Sequence): After the projection, the feature is divided into several segments each of which is labeled with a number representing how many other features are projected onto it. The sequence of numbers on these segments is the projection sequence.

Instead of analyzing each feature and all its neighboring features, we can directly carry out stitch candidate generation based on the projection sequence. For convenience, we provide a terminal zero rule, i.e., the beginning and the end of the projection sequence must be 0. To maintain this rule, sometimes a default 0 needs to be added. An example of projection sequence is shown in Fig. 6, where the middle feature has five conflict features, b, c, d, e, and f. Based on the projection results, the feature is divided into ten segments. Through labeling each segment, we can get its projection sequence: 01212101010. Here, a default 0 is added at the end of the feature.

Based on the definition of projection sequence, we summarize the rules for redundant stitches and lost stitches. First, motivated by the case in Fig. 5(a), if the projection sequence begins with "01010," then the first stitch in DP candidates is redundant. Since the projection of a feature can be symmetric, if the projection sequence ends with "01010," then the last stitch candidate is also redundant. Besides, the rule for lost stitches is as follows, if a projection sequence contains the sub-sequence "xyz," where $x, y, z > 0$ and $x > y, z > y$, then there is one lost stitch at the segment labeled as y. For example, the stitch candidate in Fig. 5(b) is contained in the sub-sequence "212," so it is a lost stitch.

Algorithm 1 Stitch Candidate Generation for TPL

Require: Projection results on features.

```
Decompose multiple-pin features;
for each feature w_i do
    Calculate the projection sequence ps_i;
    if ps_i begins or ends with "01010" then
        Remove redundant stitch(es);
    end if
    for each sequence bunch of ps_i do
        Search and insert at most one stitch candidate;
    end for
end for
```

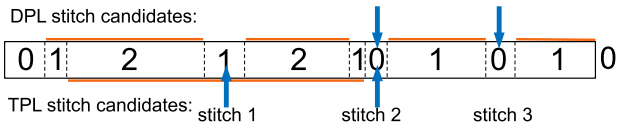


Fig. 7. Stitch candidates generated for DPL and TPL.

The details of stitch candidate generation for TPL are shown in Algorithm 1. If necessary, at first each multiple-pin feature is decomposed into several two-pin features. Then for each feature, we can calculate its projection sequence. We remove the redundant stitches by checking if the projection sequence begins or ends with "01010." Next, we search for and insert stitches, including the lost stitches. Here, we define a sequence bunch. A sequence bunch is a sub-sequence of a projection sequence, and contains at least three nonzero segments.

An example of the stitch candidate generation is shown in Fig. 7. In the DP candidate generation, there are two stitch candidates generated (stitches 2 and 3). Through our stitch candidate generation, stitch 3 is labeled as a redundant stitch. Besides, stitch 1 is identified as a lost stitch candidate because it is located in a sub-sequence "212." Therefore, stitches 1 and 2 are chosen as stitch candidates for TPL. It shall be noted that sub-sequence "101" is not necessarily a redundant stitch. That is, "101" is a redundant stitch only when it is inside "01010." For example, as illustrated in Fig. 7, one stitch candidate would be introduced inside sub-sequence "21010." In other words, sub-sequence "101" inside "21010" is not a redundant stitch location.

IV. ILP-Based Color Assignment

On DG, we carry out color assignment, which is a critical step in the layout decomposition flow. In color assignment, each vertex is assigned one of three colors (masks). In this section, first, we will give a general mathematical formulation for the color assignment. Then, we will show that it can solved through an ILP, which is commonly used before for DPLD problem [6], [7].

A. Mathematical Formulation

For convenience, some notations used in this section are listed in Table I. The general mathematical formulation for the

TABLE I
NOTATIONS

Notations used in Mathematical formulation	
CE	set of conflict edges
SE	set of stitch edges
V	the set of features
r_i	the i_{th} layout feature
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
Notations used in ILP formulation	
x_{i1}, x_{i2}	two 1-bit 0-1 variables to represents 3 colors of r_i
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}

TPLD problem is shown in (1), where the objective is to simultaneously minimize the cost of both the conflict number and the stitch number. α is a user-defined parameter for assigning relative importance between the conflict and the stitch

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

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

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

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

In (1), 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 vertices r_i and r_j are assigned the same color (mask). Constraint (1b) is used to calculate the stitch number. If vertices r_i and r_j are assigned different colors (masks), stitch s_{ij} is introduced.

B. ILP Formulation

We will now show how to implement (1) with ILP. Note that (1a) and (1b) can be linearized only when x_i is a 0–1 variable [6], which is hard to represent three different colors. To handle this problem, we represent the color of each vertex using two 1-bit 0–1 variables x_{i1} and x_{i2}. In order to limit the number of colors for each vertex 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

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

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

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

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

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

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

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

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

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

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

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

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

$$x_{ij} \quad \text{is binary.} \tag{2l}$$

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 vertex to 3. In other words, only three bit-pairs $(0, 0)$, $(0, 1)$, $(1, 0)$ are legal. Constraints (2b)–(2f) are equivalent to (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 vertices connected by conflict edge e_{ij} are in the same color, e.g., both c_{ij1} and c_{ij2} are true. Constraints (2g)–(2k) are equivalent to (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.

V. SDP-BASED COLOR ASSIGNMENT

Although ILP formulation (2) can optimally solve the color assignment problem theoretically, for practical design it may suffer from runtime overhead problem. In this section, we show that instead of expensive ILP, the color assignment can be also formulated as a vector programming, with three unit vectors to represent three different colors. Then, the vector programming is relaxed and solved through SDP. SDP is a subfield of convex optimization, where one minimizes a linear function subject to the constraint that an affine combination of symmetric matrices is positive semidefinite [30]. Although SDP is more general than linear programming, both of them can be solved in polynomial time. Besides, efficient solvers for SDP have been developed in recent years [31]. Given the solutions of SDP, we develop a mapping process to obtain the final color assignment solutions. Note that our algorithm is fast that both SDP formulation and mapping process can be finished in polynomial time.

A. Vector Programming

In color assignment, there are three possible colors. We set a unit vector $\vec{v}_i$ for every vertex i. If e_{ij} is a conflict edge, we want vertices $\vec{v}_i$ and $\vec{v}_j$ to be far apart. If e_{ij} is a stitch edge, we hope vertices $\vec{v}_i$ and $\vec{v}_j$ to be the same. As shown in Fig. 8, we associate all the vertices with three different unit vectors: $(1, 0)$, $(-1/2, \sqrt{3}/2)$, and $(-1/2, -\sqrt{3}/2)$. 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 $(v_{i1}, v_{i2}, \ldots, v_{im})$. Then, for the vectors $\vec{v}_i, \vec{v}_j \in \{(1, 0), (-1/2, \sqrt{3}/2),$

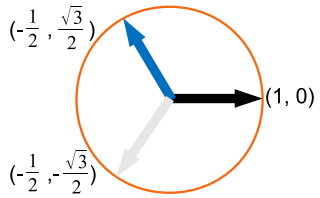


Fig. 8. Three vectors $(1, 0)$, $(-1/2, \sqrt{3}/2)$, $(-1/2, -\sqrt{3}/2)$ represent three different colors.

$(-1/2, -\sqrt{3}/2)\}$, 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 color assignment as the following vector program [32]:

$$\min \sum_{e_{ij} \in \text{CE}} \frac{2}{3}\left(\vec{v}_i \cdot \vec{v}_j + \frac{1}{2}\right) + \frac{2\alpha}{3} \sum_{e_{ij} \in \text{SE}} \left(1 - \vec{v}_i \cdot \vec{v}_j\right) \tag{3}$$

$$\text{s.t.} \quad \vec{v}_i \in \left\{(1, 0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\right\}. \tag{3a}$$

Formula (3) is equivalent to mathematical formula (1): the left part is the cost of all conflicts, and the right part gives the total cost of the stitches. Since the TPLD problem is NP-hard, this vector programming is also NP-hard. In the next part, we will relax (3) to a SDP, which can be solved in polynomial time.

B. SDP Relaxation

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

$$\min \sum_{e_{ij} \in \text{CE}} \frac{2}{3}\left(\vec{y}_i \cdot \vec{y}_j + \frac{1}{2}\right) + \frac{2\alpha}{3} \sum_{e_{ij} \in \text{SE}} \left(1 - \vec{y}_i \cdot \vec{y}_j\right) \tag{4}$$

$$\text{s.t.} \quad \vec{y}_i \cdot \vec{y}_i = 1, \qquad \forall i \in V \tag{4a}$$

$$\vec{y}_i \cdot \vec{y}_j \geq -\frac{1}{2}, \qquad \forall e_{ij} \in \text{CE}. \tag{4b}$$

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 $\vec{y}_i = (v_{i1}, v_{i2}, 0, 0, \ldots, 0)$, 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. Here, the dimension of vector $\vec{y}_i$ is $|V|$, that is, the vertex number in current DG component. 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: $Z_R \leq \text{OPT}$. In other words, solution of (4) provides a lower bound solution to that in (3). After removing the constant in objective function, we redraw the following vector programming:

$$\min \sum_{e_{ij} \in \text{CE}} \left(\vec{y}_i \cdot \vec{y}_j\right) - \alpha \sum_{e_{ij} \in \text{SE}} \left(\vec{y}_i \cdot \vec{y}_j\right) \tag{5}$$

$$\text{s.t.} \quad \text{(4a)–(4b)}.$$

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 SDP.

Consider the following standard SDP:

$$\text{SDP:} \quad \min \quad A \bullet X \tag{6}$$

$$x_{ii} = 1, \qquad \forall i \in V \tag{6a}$$

$$x_{ij} \geq -\frac{1}{2}, \qquad \forall e_{ij} \in \text{CE} \tag{6b}$$

$$X \succeq 0 \tag{6c}$$

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 ith row and the jth column of matrix A

$$a_{ij} = \begin{cases} 1, & \forall e_{ij} \in \text{CE} \\ -\alpha, & \forall e_{ij} \in \text{SE} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Constraint (6c) means matrix X should be positive semidefinite. Similarly, x_{ij} is the ith row and jth column entry of X. Note that the solution of SDP is represented as a positive semidefinite matrix X, while solutions of relaxed vector programming are stored in a list of vectors. However, we can show that they are equivalent.

Lemma 2: 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 2: The semidefinite program (6) and the vector program (5) are equivalent.

Proof: Given solutions $\{\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_m\}$ of (5), the corresponding matrix X is defined as $x_{ij} = \vec{y}_i \cdot \vec{y}_j$. In the other direction, based on Lemma 2, 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). ■

After solving the SDP formulation (6), we get a set of continuous solutions in matrix X. Since each value x_{ij} in matrix X corresponds to $\vec{y}_i \cdot \vec{y}_j$, and $\vec{y}_i \cdot \vec{y}_j$ is an approximative solution of $\vec{v}_i \cdot \vec{v}_j$ in (3), we can draw the conclusion that x_{ij} is an approximation to $\vec{v}_i \cdot \vec{v}_j$. Instead of trying to calculate all $\vec{v}_i$ through Cholesky decomposition, we pay attention to x_{ij} value itself. Essentially, if x_{ij} is close to 1, then vertices i and j tend to be in the same color; if x_{ij} is close to -0.5, vertices i and j tend to be in different colors.

For most of cases, SDP can provide reasonable solutions that each x_{ij} is either close to 1 or close to -0.5. A DG example is illustrated in Fig. 9. It contains seven conflict edges and one stitch edge. Moreover, the graph is not two-colorable since it contains several odd cycles. To solve the corresponding color assignment through SDP formulation, we construct matrix A as (7) as follows:

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

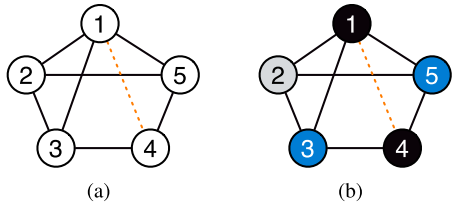


Fig. 9. Simple example of SDP. (a) Input DG component. (b) Color assignment result with 0 conflict and 0 stitch.

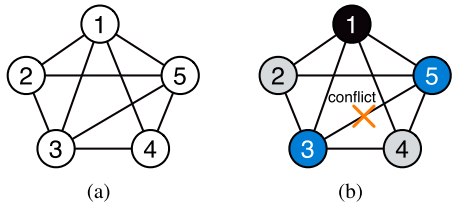


Fig. 10. Complex example. (a) Input DG component. (b) Color assignment result with one conflict.

Note that, here, we set α to 0.1. After solving the SDP (6), we can get a matrix X as follows:

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

Here, we only list the upper part of the matrix X. Because X_{14} is 1.0, vertices 1 and 4 should be in the same color. Similarly, vertices 3 and 5 should also be in the same color. In addition, because of all other -0.5 values, we know that no more vertices can be in same color. Thus, the final color assignment result for this example is shown in Fig. 9(b).

C. Mapping: SDP to Color Assignment

It shall be noted that the SDP formulation is a relaxation to the initial color assignment problem or vector programming (3). The matrix X generated from Fig. 9 is an ideal case. That is, all values are either 1 or -0.5. Therefore, from X we can derive the final color assignment easily. Our preliminary results show that with reasonable threshold such as $0.9 < x_{ij} \leq 1$ for same mask, and $-0.5 \leq x_{ij} < -0.4$ for different mask, more than 80% of vertices can be decided by the global SDP optimization. However, for practical layout, especially those essentially contain conflicts and stitches, some values in the matrix X are not so clear. We use Fig. 10 for illustration. The DG in Fig. 10(a) contains a four-clique structure $\{1, 3, 4, 5\}$, therefore at least one conflict would be reported. Through solving the SDP formulation (6), we can achieve matrix X as

$$X = \begin{pmatrix} 1.0 & -0.5 & -0.13 & -0.5 & -0.13 \\ & 1.0 & -0.5 & 1.0 & -0.5 \\ & & 1.0 & -0.5 & -0.13 \\ & \cdots & & 1.0 & -0.5 \\ & & & & 1.0 \end{pmatrix}. \quad (8)$$

From X, we can see that $x_{24} = 1.0$, therefore, vertices 2 and 4 should be in the same color. x_{13}, x_{15}, and x_{35} are not so clear (-0.13). For those vague values, we propose a mapping process to find the final color assignment solutions. In the following, we will explain the mapping algorithm. All x_{ij} values in matrix X are divided into two types: 1) clear and 2) vague. If x_{ij} is close to 1 or -0.5, it is denoted as a clear value; otherwise it is a vague value. The mapping uses all the x_{ij} values as the guideline to generate the final decomposition results, even when some x_{ij}s are vague.

Here, we propose a partition-based mapping, which can solve the assignment problem for the vague X_{ij}s in a more effective way. The main ideas are as follows. If a X_{ij} is vague, instead of only relying on the SDP solution, we also take advantage of the information in DG. The information is captured through constructing a graph, denoted by G_M. Through formulating the mapping as a three-way partitioning on the graph G_M, our mapping can provide a global view to search better solutions.

Algorithm 2 Partition-Based Mapping

Require: Solution matrix X of the program (6).
1: Label each nonzero entry $X_{i,j}$ as a triplet (X_{ij}, i, j);
2: Sort all (X_{ij}, i, j) by X_{ij};
3: **for** all triples with $X_{ij} > th_{\text{unn}}$ **do**
4: Union(i, j);
5: **end for**
6: **for** all triples with $X_{ij} < th_{\text{sp}}$ **do**
7: Separate(i, j);
8: **end for**
9: Construct graph G_M;
10: **if** graph size ≤ 3 **then**
11: return;
12: **else if** graph size ≤ 7 **then**
13: Backtracking-based three-way partitioning;
14: **else**
15: FM-based three-way partitioning;
16: **end if**

Algorithm 2 shows our partition-based mapping procedure. Given the solutions from program (6), some triplets are constructed and sorted to maintain all nonzero X_{ij} values (lines 1 and 2). The mapping incorporates two stages to deal with the two different types. The first stage (lines 3–8) is similar to that in [16]. If X_{ij} is clear then the relationship between vertices r_i and r_j can be directly determined. Here, th_{unn} and th_{sp} are user-defined threshold values. For example, if $X_{ij} > th_{\text{unn}}$, which means that r_i and r_j should be in the same color, then function Union (i, j) is applied to merge them into a large vertex. Similarly, if $X_{ij} < th_{\text{sp}}$, then function Separate (i, j) is used to label r_i and r_j as incompatible. In the second stage (lines 9–16), we deal with the vague X_{ij} values. During the previous stage, some vertices have been merged, therefore the total vertex number is not large. Here, we construct a graph G_M to represent the relationships among all the remanent vertices (line 9). Each edge e_{ij} in this graph has a weight representing the cost if vertices i and j are assigned into same color. Therefore, the color assignment problem can be formulated as a maximum-cut partitioning problem on G_M (lines 10–16).

It is well known that the maximum-cut problem, even for a two-way partition, is NP-hard. However, we observe that in many cases, after the global SDP optimization, the graph size of G_M could be quite small, i.e., less than 7. For these small cases, we develop a backtracking-based method to search the entire solution space. Note that, here backtracking can quickly find the optimal solution even through three-way partitioning is NP-hard. If the graph size is larger, we propose a heuristic method, motivated by the classic FM partitioning algorithm [33], [34]. Different from the classic FM algorithm, we make the following modifications.

1) In the first stage of mapping, some vertices are labeled as incomparable, therefore before moving a vertex from one partition to another, we should check whether it is legal.
2) Classical FM algorithm is for min-cut problem, we need to modify the gain function of each move to achieve a maximum cut.

We use the disjoint-set data structure to group vertices into three colors. Implemented with union by rank and path compression, the running time per operation of disjoint-set is almost constant [35]. Let n be the number of vertices, then the number of triplets is $O(n^2)$. Sorting all the triplets requires $O(n^2 logn)$. Since all triplets are sorted, each of them can be visited at most once. Besides, the runtime complexity of graph construction is $O(m)$, where m is the vertex number in G_M. The runtime of three-way maximum-cut partitioning algorithm is $O(mlogm)$. Since m is much smaller than n, the complexity of partition balanced mapping is $O(n^2 logn)$.

VI. Graph Division

To further achieve some speedup, instead of solving color assignment in one DG, we propose several techniques to divide the graph into a bunch of components. Then, each component can be solved color assignment independently.

A. LG Division

Given input layout, LG is constructed first. We propose two methods to divide/simplify the LG in order to reduce the problem size.

1) ICC: The first division technique is called ICC. In a LG of real design, we observe many isolated clusters. By breaking down the whole LG into several independent components, we partition the initial LG into several small ones. After solving the TPLD problem for each isolated component, the overall solution can be taken as the union of all the components without affecting the global optimality. It shall be noted that ICC is a well-known technique which has been applied in many previous studies.

2) IVR: We can further simplify the LG by iteratively removing all vertices with degree less than or equal to two. This technique is called IVR, as described in Algorithm 3. At the beginning, all vertices with degree no more than two are detected and removed temporarily from the LG. After each vertex removal, we need to update the degrees of other vertices. This removing process will continue until all the vertices are at least degree-three. All the vertices that are temporarily removed are stored in stack S. Then, DG are constructed for the remanent vertices. After solving the color assignment on each DG component, the removed vertices are recovered one-by-one.

Algorithm 3 IVR and Color Assignment

Require: LG G, 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: Construct DG for the remanent vertices;
6: **for** each component in DG **do**
7: Apply color assignment;
8: **end for**
9: **while** !$S.empty()$ **do**
10: $n = S.pop()$;
11: $G.add(n)$;
12: Assign n a legal color;
13: **end while**

If all the vertices in one LG can be temporarily removed (pushed onto the stack S), TPLD problem is solved optimally in linear time. An example is illustrated in Fig. 11, where all the vertices can finally be pushed onto stack. Even there are still some vertices remained, our IVR technique can minimize problem size dramatically. Additionally, we observe that this technique can further partition the LG into several independent components.

B. DG Division

On the LG simplified by ICC and IVR, projection is carried out to calculate all the potential stitch positions. Then, we construct the DG, which includes the conflict edges and the stitch edges. Here, the stitch edges are based on the projection result. Note that ICC can be still applied here to partition a DG into several smaller ones. We further propose three new techniques to reduce the size of each DG.

1) Bridge Edge Detection and Removal: A bridge edge of a graph is an edge whose removal disconnects the graph into two components. Removing the bridge edge can divide the whole problem into two independent sub-problems.

An example of the bridge edge detection is shown in Fig. 12. Conflict edge e_{ab} is identified as a bridge edge. Removing the bridge edge divides the DG into two sides. After layout decomposition for each component, if vertices a and b are assigned the same color, without loss of generality, we can rotate colors of all vertices in the lower side. Similar method can be adopted when bridge is a stitch edge. We adopt an $O(|V| + |E|)$ algorithm [36] to detect all bridge edges in DG.

2) Bridge Vertex Detection and Duplication: This technique is also defined as "two-connected component computation" in [18]. A bridge vertex of a graph is a vertex whose removal disconnects the graph into two or more components. Similar to bridge edge detection, we can further simplify the DG by removing all the bridge vertices. An example of bridge

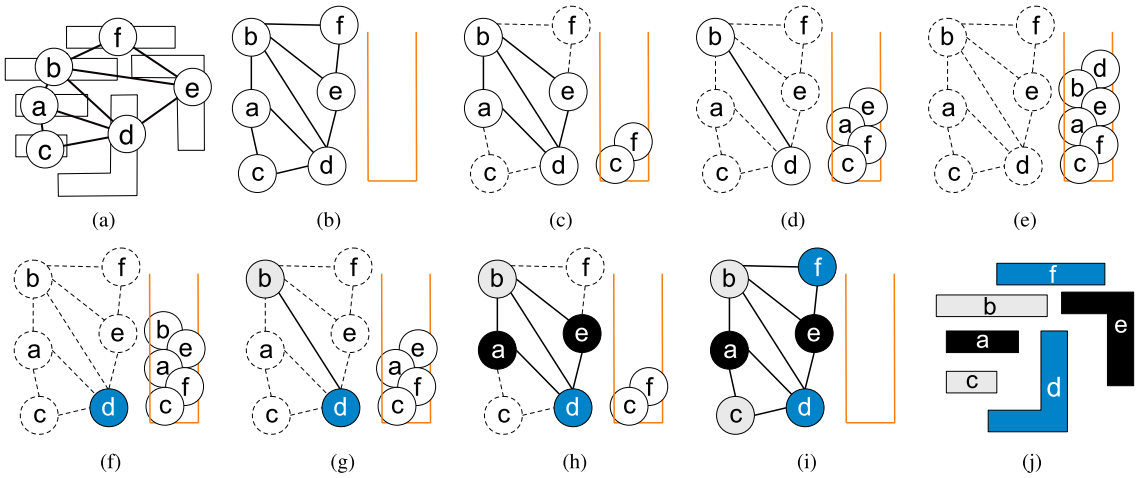


Fig. 11. Example of IVR, where the TPLD problem can be solved in linear time. (a) LG. (b)–(e) Iteratively remove and push in vertices with degree less than three. (f)–(i) After color assignment for the remanent vertices, iteratively pop-up and recover vertices, and assign any legal color. (j) TPLD can be finished after the iterative vertex recover.

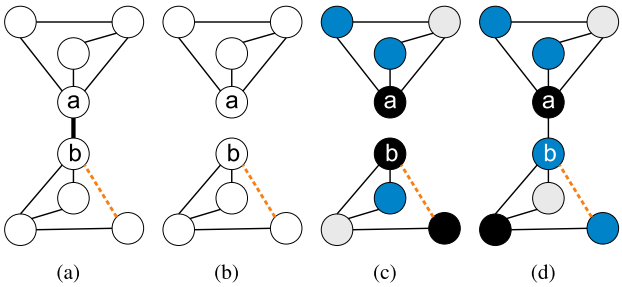


Fig. 12. Bridge edge detection and removal. (a) Initial DG. (b) After bridge edge detection, remove edge e_{ab}. (c) In two components, we carry out layout decomposition. (d) Rotate colors in the lower component to add bridge.

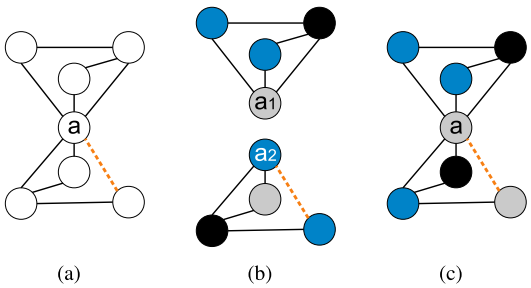


Fig. 13. Bridge vertex detection and duplication. (a) Initial DG. (b) After bridge vertex detection, duplicate vertex a. (c) Rotate the colors in lower sub-graph to merge vertices a_1 and a_2.

vertex computation is illustrated in Fig. 13. This simplification method is effective because for standard cell layouts, usually we can choose the power and ground lines as the bridge vertices. By this way, we can significantly partition the layouts by rows. All bridge vertices can be detected using an $O(|V|+|E|)$ search algorithm.

3) Fast Color Assignment Trial: Although the SDP and the partition-based mapping can provide high performance for color assignment, it is still expensive to be applied to all the DG components. We derive a fast color assignment trial before calling SDP-based method. If no conflict or stitch is introduced, our trial solves the color assignment problem in linear time. Note that SDP method is skipped only when DG can be colored without stitch or conflict, our fast trial does not lose any solution quality. Besides, our preliminary results show that more than half of the DGs can be decomposed using this fast method. Therefore, the runtime can be dramatically reduced.

Algorithm 4 Fast Color Assignment Trial

Require: DG G, stack S.

1: **while** $\exists n \in G$ s.t. $d_{\text{conf}}(n) < 3$ & $d_{\text{stit}}(n) < 2$ **do**
2: $S.push(n)$; $G.delete(n)$;
3: **end while**
4: **if** G is not empty **then**
5: Recover all vertices in S; **return** *FALSE*;
6: **else**
7: **while** !$S.empty()$ **do**
8: $n = S.pop()$; $G.add(n)$;
9: Assign n a legal color;
10: **end while**
11: **return** *TRUE*;
12: **end if**

The details of fast color assignment trial is shown in Algorithm 4. First, we iteratively remove the vertex with conflict degree (d_{conf}) less than 3 and stitch degree (d_{stit}) less than 2 (lines 1–3). If some vertices cannot be removed, we recover all the vertices in stack S, then return false; otherwise, the vertices in S are iteratively popped (recovered) (lines 7–11). For each vertex n popped, since it is connected with at most one stitch edge, we can always assign one color without introducing conflict. However, several additional stitches may be introduced when some vertices are recovered from stack.

VII. Post Refinement

Although the graph-division techniques can dramatically reduce the computational time to solve the TPLD problem,

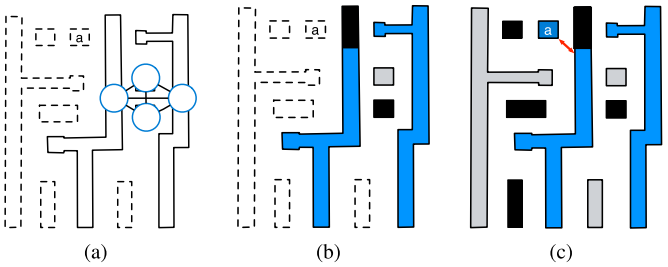


Fig. 14. IVR may introduce additional conflicts. (a) LG after IVR. (b) Stitch generation and color assignment on the graph. (c) After adding back the simplified vertices, one additional conflict is introduced to vertex a.

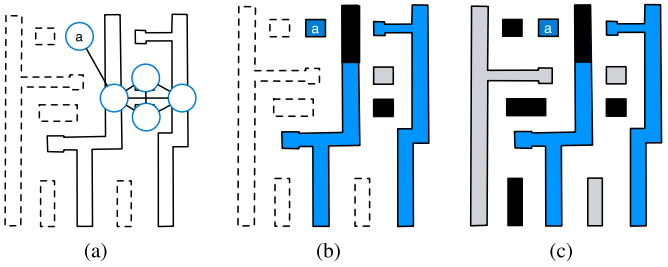


Fig. 15. Example of post-refinement. (a) Extend LG to include a. (b) Stitch generation and color assignment on the new graph. (c) No additional conflict at final solution.

Kuang and Young [20] pointed out that for some cases IVR may loss some optimality. One example is illustrated in Fig. 14. The simplified LG [Fig. 14(b)] can be inserted stitch candidates and assigned legal colors [see Fig. 14(b)]. However, when recover removed vertices, the vertex degree of a is increased to 3, and there is no available color for it [see Fig. 14(c)]. The reason for this conflict is that during stitch candidate generation, vertex a is not considered.

We propose a post refinement to resolve the conflicts caused by IVR. First, we check whether one conflict can be cleared by splitting a, if yes, one or more stitches would be introduced to a, then stop. Otherwise, we recalculate the color assignment on this portion, as illustrated in Fig. 15. The initial LG would be extended to include vertex a [Fig. 15(a)], thus the position of vertex a would be considered during stitch candidate generation. As shown in Fig. 15(c), no additional conflict would be introduced after recovering all vertices from stack.

During post-refinement, after solving the new DG, the simplified vertices would be pushed back and assigned colors one-by-one. If there is still conflicts caused by the simplified vertices, we will check whether one conflict can be removed through stitch insertion. If yes, then one more stitch would be reported. Otherwise, if we cannot find stitch to split the vertex, one conflict would be reported. Then, this vertex would not be considered during the following color assignment. Although resolving color assignment requires more computational time, our initial results show that only small part of DG components need to apply the post-stage.

VIII. Experimental Results

We implement our algorithm in C++ and test it on an Intel Core 2.9 GHz Linux machine. We choose GUROBI [37] as the ILP solver, while CSDP [31] as the SDP solver. ISCAS benchmarks from [7] and [9] are scaled down and modified as our test cases. 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 coloring spacing $\min_s$ is set to 120 for the first ten cases and as 100 for the last five cases, as in [16] and [18]. Parameter α is set to 0.1, thus the decomposition cost is calculated by $cn\# + 0.1 \cdot st\#$, where $cn\#$ and $st\#$ denote the conflict number and the stitch number, respectively. Fig. 16 illustrates part of the decomposition result for case S1488, which can be decomposed in 0.1 s.

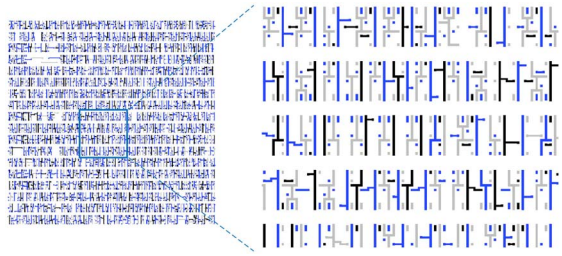


Fig. 16. Part of S1488 decomposition result.

TABLE II
DP Stitch Versus TP Stitch

Circuit	ILP w. DP Stitch				ILP w. TP Stitch			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
C432	0	4	4.0	0.3	4	0	0.4	0.3
C499	0	0	0	0.2	0	0	0	0.2
C880	4	3	3.4	0.3	7	0	0.7	0.3
C1355	1	3	3.1	0.2	3	0	0.3	0.3
C1908	0	1	1.0	0.3	1	0	0.1	0.2
C2670	1	5	5.1	0.5	6	2	2.6	0.4
C3540	4	5	5.4	0.8	9	3	3.9	0.8
C5315	4	5	5.4	1.1	9	1	1.9	0.8
C6288	121	87	99.1	9.2	212	4	25.2	14.3
C7552	7	15	15.7	2.5	27	3	5.7	2.1
S1488	2	0	0.2	0.1	3	0	0.3	0.1
S38417	47	29	33.7	3.8	59	20	25.9	4.1
S35932	62	45	51.2	7.3	62	45	51.2	9.3
S38584	106	49	59.6	7.0	131	40	53.1	6.9
S15850	82	52	60.2	6.7	117	35	46.7	8.2
avg.	29.4	20.2	23.1	2.7	43.3	10.2	14.5	3.2
ratio	1.0	1.0	**1.0**	**1.0**	1.47	0.50	**0.63**	**1.20**

A. DP Stitch Versus TP Stitch

First, we demonstrate the effectiveness of our stitch candidate generation. Table II compares the performance and runtime of ILP on two different stitch candidates, i.e., DP stitch and TP stitch. "ILP w. DP stitch" and "ILP w. TP stitch" apply DP stitch and TP stitch, respectively. Note that, here all graph-division techniques are applied here. The columns "st#" and "cn#" denote the stitch number and the conflict number. Column "CPU(s)" is computational time in seconds. As discussed in Section III, through applying TP stitch more stitch candidates would be generated, therefore, we can see from Table II that 20% more runtime would be introduced. However, TP stitch overcomes the lost stitch problem in DP stitch, thus the decomposition cost is reduced by 37%. In other words, compared with DP stitch, TP stitch can provide higher performance in terms of the conflict number and the stitch number.

TABLE III
EFFECTIVENESS OF GRAPH DIVISION

Circuit	ILP w. ICC						ILP w. 4SPD					
	TCE#	TSE#	DG#	st#	cn#	CPU(s)	TCE#	TSE#	DG#	st#	cn#	CPU(s)
C432	2710	934	123	4	0	15.1	69	25	4	4	1	0.62
C499	7670	2426	175	0	0	29.9	61	17	3	0	0	0.24
C880	5132	1871	270	7	0	29.3	97	40	8	8	1	0.49
C1355	6640	2522	467	3	0	48.3	86	38	6	3	0	0.14
C1908	11346	4214	507	1	0	55.4	62	21	3	1	0	0.31
C2670	19716	7026	614	6	0	72.7	148	56	10	7	1	0.5
C3540	24076	8849	827	8	1	100.6	218	90	20	8	2	1.31
C5315	34668	12789	1154	9	0	134.4	263	111	18	9	1	0.94
C6288	31943	11276	2325	215	1	340.7	3389	1386	293	217	18	16.2
C7552	49496	18407	1783	22	0	209.5	616	267	54	27	2	2.37
S1488	9108	4232	274	2	0	31	123	58	16	2	0	0.09
S38417	122120	52726	5298	56	19	653.2	10432	5255	1510	76	19	5.14
S35932	288171	115739	15804	N/A	N/A	>7200	32073	16237	4713	99	47	13.15
S38584	299613	127063	16235	N/A	N/A	>7200	30663	15515	4517	157	43	11.6
S15850	298423	125722	13226	N/A	N/A	>7200	26468	13312	3807	130	40	10.7
avg.	80722	33053	-	-	-	>1554.7	6984.5	3495.2	-	49.9	11.7	4.25
ratio	**11.6**	**9.46**	-	-	-	**>365.5**	**1.0**	**1.0**	-	-	-	**1.0**

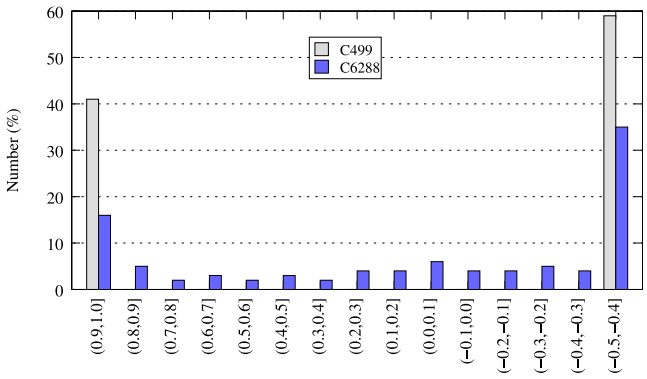


Fig. 17. Value distribution in matrix X for cases C499 and C6288.

B. Effectiveness of Graph Division

Secondly, we show the effectiveness of the graph division. Through applying these division techniques, the DG size can be reduced. Generally speaking, smaller size of DG, less runtime the ILP needs. Table III compares the performance and runtime of ILP on two different DGs. Here, "ILP w. ICC" means the DGs are only simplified by the ICC, while "ILP w. 4SPD" means all the division techniques are used. Columns "TCE#" and "TSE#" denote the total conflict edge number and total stitch edge number, respectively. From Table III, we can see that compared with only using ICC technique, further applying IVR and bridges detection is more effective: the stitch edge number can be reduced by 92%, 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 decomposition results. "CPU(s)" is computational time in seconds. Compared with the "ILP w. ICC," the "ILP w. 4SPD" can achieve the same results with much less of the runtime for some smaller cases. For some big circuits, the runtimes of "ILP w. ICC" are unacceptable, i.e., longer than 2 h. Note that if no ICC technique is used, even for small circuits like C432, the runtime for ILP is unacceptable.

C. Effectiveness of SDP

Here, we show some more details of solutions in SDP. As discussed before, if the value X_{ij} is close to 1 or -0.5, it can be directly rounded to an integer value. Otherwise, we have to rely on some mapping methods. Fig. 17 illustrates the X_{ij} value distributions in circuit C499 and C6288. As we can see that all for C499, the values are either in the range of [0.9, 1.0] or in $[-0.5, -0.4]$. In other words, here SDP is effective and its results can be directly used as final decomposition results. For the case C6288, since its result consists of several stitches and conflicts, some X_{ij} values are vague. But most of the values are still distinguishable.

TABLE IV
EFFECTIVENESS OF POST-REFINEMENT

Circuit	SDP w/o. Refinement				SDP w. Refinement			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
C432	4	0	0.4	0.1	4	0	0.4	0.1
C499	0	0	0	0.1	0	0	0	0.1
C880	7	0	0.7	0.1	7	0	0.7	0.1
C1355	3	0	0.3	0.1	3	0	0.3	0.1
C1908	1	0	0.1	0.1	1	0	0.1	0.1
C2670	6	1	1.6	0.2	6	0	0.6	0.2
C3540	9	2	2.9	0.3	9	1	1.9	0.4
C5315	9	0	0.9	0.3	9	0	0.9	0.4
C6288	213	5	26.3	2.1	213	1	22.3	1.7
C7552	26	2	4.6	0.6	26	0	2.6	0.6
S1488	3	0	0.3	0.1	3	0	0.3	0.1
S38417	59	21	26.9	1.5	57	20	25.7	1.8
S35932	60	46	52	3.9	60	46	52	6.4
S38584	131	40	53.1	3.7	131	36	49.1	5.8
S15850	118	36	47.8	3.9	119	34	45.9	5.8
avg.	43.3	10.2	14.6	1.14	43.2	9.2	13.52	1.58
ratio	1.0	1.0	**1.0**	**1.0**	0.99	0.91	**0.93**	**1.39**

D. Effectiveness of Post-Refinement

We further demonstrate the effectiveness of the post-refinement. Table IV lists the decomposition results of two SDP-based algorithms. Columns "SDP w/o. Refinement" and "SDP w. Refinement" mean SDP without and with post-refinement, respectively. As shown in Table IV, through additional post-refinement stage, the decomposition costs can be reduced by 7%, while 39% more computational time is introduced.

E. Comparison With Other Decomposers

Finally, we compare our decomposition algorithms with the state-of-the-art layout decomposers [18], [20], [22], as

TABLE V
COMPARISON WITH OTHER DECOMPOSERS

Circuit	TCAD [18]			DAC'13 [20][1]			ICCAD'13 [22]			ILP w. All			SDP w. All		
	st#	cn#	CPU(s)	st#	cn#	CPU(s)	st#	cn#	CPU(s)	st#	cn#	CPU(s)	st#	cn#	CPU(s)
C432	6	0	0.11	4	0	0.01	4	0	11.6	4	0	0.5	4	0	0.1
C499	0	0	0.02	0	0	0.01	0	0	24.7	0	0	0.3	0	0	0.1
C880	8	1	0.12	7	0	0.01	11	0	21.5	7	0	0.4	7	0	0.1
C1355	4	1	0.14	3	0	0.01	10	0	35	3	0	0.4	3	0	0.1
C1908	0	1	0.08	1	0	0.01	4	0	54.9	1	0	0.3	1	0	0.1
C2670	11	2	0.1	6	0	0.04	25	0	100.6	6	0	0.6	6	0	0.2
C3540	11	3	0.11	8	1	0.05	43	1	122.8	9	1	1.2	9	1	0.4
C5315	11	3	0.17	9	0	0.05	69	0	203	9	0	1.1	9	0	0.4
C6288	243	20	0.17	191	14	0.25	240	21	183.4	212	0	15.4	213	1	1.7
C7552	37	3	0.28	22	0	0.1	80	2	337.6	26	0	2.5	26	0	0.6
S1488	4	0	0.1	2	0	0.01	15	0	39.3	3	0	0.1	3	0	0.1
S38417	82	20	0.74	55	19	0.42	170	20	682	57	19	4.3	57	20	1.8
S35932	63	46	2.1	41	44	0.82	260	48	1877	62	44	10.1	60	46	6.4
S38584	176	36	2.3	116	36	0.77	313	46	1881	131	36	8.6	131	36	5.8
S15850	146	36	2	100	34	1.0	412	48	2222	117	34	10.1	119	34	5.8
avg.	53.5	11.5	0.57	37.7	9.9	0.24	110.4	12.4	519.76	43.1	8.9	3.7	43.2	9.2	1.58
ratio		**1.25**	**0.36**		**1.07**	**0.15**		**1.35**	**329.0**		**0.97**	**2.36**		**1.0**	**1.0**

TABLE VI
COMPARISON WITH OTHER DECOMPOSERS ON VERY DENSE LAYOUTS

Circuit	TCE#	TSE#	TCAD [18]			ICCAD'13 [22]			ILP w. All			SDP w. All		
			st#	cn#	CPU(s)	st#	cn#	CPU(s)	st#	cn#	CPU(s)	st#	cn#	CPU(s)
c5_total	248	0.16	374	248	0.15	487	45	183	493	8	105.2	497	11	6.8
c6_total	522	0.3	869	521	0.26	1120	84	381	N/A	N/A	>3600	1149	65	17.8
c7_total	623	0.3	938	624	0.33	1257	122	458	1315	58	451.4	1304	75	13.5
c8_total	727	0.53	1671	727	0.51	1788	183	778	2004	84	260.1	1964	115	20.9
c9_total	525	0.4	635	525	0.4	916	46	574	889	68	79.8	882	74	11.4
c10_total	1144	0.86	2120	1146	0.88	2527	255	1302	2649	162	802.6	2592	195	31.7
avg.	631.5	0.43	1101.2	631.8	0.42	1349.2	122.5	612.7	-	-	>883.2	1398	89.2	17.0
ratio	-	-		**7.08**	**0.03**		**1.37**	**36.0**		-	**>51.9**		**1.0**	**1.0**

shown in Table V. We have obtained executable programs from [18] and [22]. Since we cannot get the decomposer binary from [20], the results are directly from their paper. Columns "ILP w. All" and "SDP w. All" denote ILP-based algorithm and SDP-based algorithm, respectively. Here, "w. All" means all other techniques, e.g., TP stitch, graph division, and post-refinement, are applied. The decomposer [18] is fast due to several graph-based division and coloring heuristic. However, the nature of heuristic cannot guarantee good solution quality. Thus compared with [18], ILP-based methods and SDP-based methods can reduce conflict number by 20%. The decomposer [20] presents library-based approach, thus it can provides fast solution when input layout is not complex, otherwise heuristic methods would be applied. Compared with [20], our ILP-based method and SDP-based method can reduce the conflict number by 10% and 7%, respectively. For weighted sum cost which is $cn\# + 0.1 \cdot st\#$, ILP-based method and SDP-based method can reduce the cost by 3% and 1%, respectively. The decomposer [22] relies on randomized coloring method and SPQR-tree-based graph division. However, due to the nature of randomized algorithms, the performance of [22] is not good, and our methods can achieve around 30% cost reduction. Besides, since we provide a set of powerful graph division techniques, ILP-based algorithms and SDP-based algorithms can achieve 140× and 300× speed-up, respectively. ILP-based algorithm has best performance in terms of conflict number, but it may suffer from longer runtime problem. Compared with ILP, SDP-based algorithm provides much better trade-off between runtime and performance, i.e., it can achieve very comparable results (3% of conflict difference) and more than 2× speed-up.

In order to further evaluate the scalability of all the decomposers, we create six additional benchmarks ("c5_total" – "c10_total") to compare different algorithms on very dense layouts. Table VI lists the comparison results. Columns "TCE#" and "TSE#" denote the total conflict edge number and total stitch edge number, respectively. It shall be noted that Kuang and Young [20] cannot provide us the binary, since their program cannot handle these cases. The reason is that they assume that each DG component has at most six nodes. Otherwise, the library-based approach, or even their heuristic coloring method may fail. Unfortunately, such assumptions cannot be held any more for the new benchmark suite. As we can see, compared with SDP-based method, although ILP can achieve the best decomposition results, its high runtime complexity makes it impossible to solve one large dense layout, even all the graph-division techniques are adopted. Although the decomposer [18] is faster that all the cases can be finished in 1 s, it introduces hundreds of additional conflicts for this new benchmark suite. Each conflict may require manual layout modification or high ECO efforts, which are very time consuming. In addition, compared with [22], our SDP-based algorithms can achieve 36× speed-up and around 30% cost reduction. Therefore, we can see that for these dense layouts,

[1]The results are directly from [20].

SDP-based algorithm can achieve good trade-off in terms of runtime and performance.

IX. Conclusion

In this paper, we have proved that TPLD problem is NP-hard, and the runtime required to solve it exactly increases dramatically with the problem size. To reduce the problem size, we presented a set of graph-division techniques. Then we proposed a general ILP formulation to simultaneously minimize the conflicts and stitches. Furthermore, we proposed a novel vector program, and its SDP relaxation to improve scalability for very dense layouts. Experimental results showed that our methods are very effective. Triple patterning or even quadruple patterning may be a promising manufacturing solution for sub-10 nm nodes. We believe the SDP-based methods are generic and robust to be extended to quadruple patterning problem. This paper will stimulate more future research into this field.

Acknowledgment

The authors would like to thank Dr. K. Lucas and Dr. G. Luk-Pat at Synopsys for helpful discussions, as well as Dr. S.-Y. Fang and Y. Zhang for providing their layout decomposer binaries [18], [22].

References

[1] K. Lucas *et al.*, "Double-patterning interactions with wafer processing, optical proximity correction, and physical design flows," *J. Micro/Nanolith. MEMS MOEMS (JM3)*, vol. 8, no. 3, 2009, Art. ID 033002.

[2] D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nanolithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 10, pp. 1453–1472, Oct. 2013.

[3] B. Yu *et al.*, "Dealing with IC manufacturability in extreme scaling," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 240–242.

[4] K. Lucas *et al.*, "Implications of triple patterning for 14 nm node design and patterning," *Proc. SPIE*, vol. 8327, Mar. 2012, Art. ID 832703.

[5] Y. Borodovsky, "Lithography 2009 overview of opportunities," in *Proc. Semicon West*, San Francisco, CA, USA, 2009.

[6] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 939–952, Jun. 2010.

[7] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 2, pp. 185–196, Feb. 2010.

[8] Y. Xu and C. Chu, "GREMA: Graph reduction based efficient mask assignment for double patterning technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2009, pp. 601–606.

[9] J.-S. Yang, K. Lu, M. Cho, K. Yuan, and D. Z. Pan, "A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Taipei, Taiwan, 2010, pp. 637–644.

[10] Y. Xu and C. Chu, "A matching based decomposer for double patterning lithography," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, San Francisco, CA, USA, 2010, pp. 121–126.

[11] X. Tang and M. Cho, "Optimal layout decomposition for double patterning technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2011, pp. 9–13.

[12] Q. Ma, H. Zhang, and M. D. F. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14 nm technology," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 591–596.

[13] Y.-H. Lin, B. Yu, D. Z. Pan, and Y.-L. Li, "TRIAD: A triple patterning lithography aware detailed router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 123–129.

[14] B. Yu, X. Xu, J.-R. Gao, and D. Z. Pan, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 349–356.

[15] C. Cork, J.-C. Madre, and L. Barnes, "Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns," *Proc. SPIE*, vol. 7028, May 2008, Art. ID 702839.

[16] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2011, pp. 1–8.

[17] R. S. Ghaida, K. B. Agarwal, L. W. Liebmann, S. R. Nassif, and P. Gupta, "A novel methodology for triple/multiple-patterning layout decomposition," *Proc. SPIE*, vol. 8327, Mar. 2012, Art. ID 83270M.

[18] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 3, pp. 397–408, Mar. 2014.

[19] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 57–64.

[20] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, Austin, TX, USA, 2013, pp. 1–6.

[21] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. Wong, "Constrained pattern assignment for standard cell based triple patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 57–64.

[22] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 170–177.

[23] B. Yu *et al.*, "A high-performance triple patterning layout decomposer with balanced density," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 163–169.

[24] B. Yu, J.-R. Gao, and D. Z. Pan, "Triple patterning lithography (TPL) layout decomposition using end-cutting," *Proc. SPIE*, vol. 8684, Mar. 2013, Art. ID 86840G.

[25] Z. Chen, H. Yao, and Y. Cai, "SUALD: Spacing uniformity-aware layout decomposition in triple patterning lithography," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, Santa Clara, CA, USA, 2013, pp. 566–571.

[26] K. Yuan and D. Z. Pan, "WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2010, pp. 32–38.

[27] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[28] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," *SIAM J. Comput.*, vol. 16, no. 3, pp. 421–444, 1987.

[29] R. Tamassia, G. Di Battista, and C. Batini, "Automatic graph drawing and readability of diagrams," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 1, pp. 61–79, Jan./Feb. 1988.

[30] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Rev.*, vol. 38, no. 1, pp. 49–95, 1996.

[31] B. Borchers, "CSDP, a C library for semidefinite programming," *Optim. Methods Softw.*, vol. 11, no. 1, pp. 613–623, 1999.

[32] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[33] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, 1982, pp. 175–181.

[34] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 62–81, Jan. 1989.

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

[36] R. E. Tarjan, "A note on finding the bridges of a graph," *Inf. Process. Lett.*, vol. 2, no. 6, pp. 160–161, 1974.

[37] Gurobi Optimization Inc. (2014). *Gurobi Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com

Bei Yu (S'11–M'14) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, in 2014.

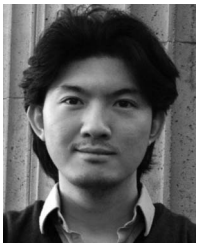
He is currently a Post-Doctoral Scholar with the Department of Electrical and Computer Engineering, University of Texas at Austin. His current research interests include design for manufacturability and optimization algorithms.

Dr. Yu was the recipient of the Best Paper Awards at the International Conference on Computer Aided Design (ICCAD) 2013 and the Asia and South Pacific Design Automation Conference (ASPDAC) 2012, three other Best Paper Award Nominations at Design Automation Conference'14, ASPDAC'13, and ICCAD'11, the Chinese Government Award for Outstanding Students Abroad in 2013, the Society of Photo-Optical Instrumentation Engineers Education Scholarship in 2013, the Silver Medal in ACM Student Research Contest at ICCAD'13, and the IBM Ph.D. Scholarship in 2012.

Kun Yuan received the B.S. degree in electronic engineering information science from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2004 and 2010, respectively.

He is currently a Senior Engineer with Facebook Inc., San Jose, CA, USA.

Dr. Yuan was the recipient of the International Symposium on Physical Design (ISPD) Routing Contest Award in 2007, and the three Best Paper Awards at Asia and South Pacific Design Automation Conference'10, ISPD 2011, and the IBM Research 2010 Pat Goldberg Memorial Best Paper Award in CS/EE/Math.



Duo Ding received the Ph.D. degree from the Department of Electrical and Computer Engineering, the University of Texas at Austin, Austin, TX, USA, in 2011.

He is currently a Principal Hardware Engineer with Oracle, Austin.

Dr. Ding was the recipient of the 2013 ACM Special Interest Group on Design Automation Best Ph.D. Dissertation Award, the Best Paper Award in the 2012 Asia and South Pacific Design Automation Conference, and the Best Student Paper Award in the 2009 International Conference on IC Design Technology, among many other peer-reviewed conference and journal publications.

David Z. Pan (S'97–M'00–SM'06–F'14) received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from University of California, Los Angeles (UCLA), Los Angeles, CA, USA.

From 2000 to 2003, he was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Engineering Foundation Endowed Professor with the Department of Electrical and Computer Engineering, the University of Texas at Austin, Austin, TX, USA. His current research interests include cross-layer nanometer IC design for manufacturability/reliability, new frontiers of physical design, and CAD for emerging technologies such as 3-D-IC, bio, and nanophotonics. He has published over 200 papers in refereed journals and conferences, and holds eight U.S. patents.

Prof. Pan was the recipient of several awards including the SRC 2013 Technical Excellence Award, the DAC Top Ten Author in Fifth Decade, the DAC Prolific Author Award, 11 Best Paper Awards, and several International CAD Contest Awards, Communications of the ACM Research Highlights'14, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award thrice, the IBM Faculty Award four times, the UCLA Engineering Distinguished Young Alumnus Award in 2009, and the UT Austin RAISE Faculty Excellence Award in 2014. He has served as a Senior Associate Editor of the *ACM Transactions on Design Automation of Electronic Systems*, an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART-I, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART-II, the *Science China Information Sciences*, the *Journal of Computer Science and Technology*, and the IEEE CAS Society Newsletter. He has served as the Chair of the IEEE CANDE Committee and the ACM/SIGDA Physical Design Technical Committee, Program/General Chair of ISPD, TPC Subcommittee Chair for DAC, ICCAD, ASPDAC, ISLPED, ICCD, Tutorial Chair for DAC 2014, and a Workshop Chair for ICCAD 2015, among others.