Discrete Relaxation Method for Triple Patterning Lithography Layout Decomposition

Xingquan Li, Ziran Zhu, and Wenxing Zhu

Abstract—In this paper, we consider the triple patterning lithography layout decomposition problem. To address the problem, a discrete relaxation theory is built. For designing a discrete relaxation based decomposition framework, we propose a surface projection method for identifying native conflicts in a layout, and then constructing the conflict graph. Guided by the theory, the conflict graph is reduced to small size subgraphs by vertex removals, which is a discrete relaxation. Furthermore, by ignoring stitch insertions and assigning weights to features, the layout decomposition problem on the small subgraphs is further relaxed to a 0-1 program, which is solved by the Branch-and-Bound method. To obtain a feasible solution of the original problem, legalization methods are introduced to legalize a relaxation solution. At the legalization stage, we prior utilize one-stitch insertion to eliminate conflicts, and use a backtrack coloring algorithm to obtain a better solution. We test our decomposition approach on the ISCAS-85 & 89 benchmarks. Comparisons of experimental results show that our approach finds solutions of some benchmarks better than those by the state-of-the-art decomposers. Especially, according to our discrete relaxation theory, some optimal decompositions are obtained.

Index Terms—Triple patterning lithography, layout decomposition, discrete relaxation, surface projection, 0-1 program

1 Introduction

Relaxation is an important approach for NP-hard combinatorial optimization problems. The minimum value of a relaxation problem provides a lower bound on the minimum value of an NP-hard combinatorial optimization problem. Moreover, by legalizing a minimum solution of the relaxation problem to a feasible one of the original problem, we can get a possibly least upper bound on the minimum value of the original problem. There are two categories of relaxation methods, i.e., continuous relaxation and discrete relaxation. Continuous relaxation includes convex relaxations [1], e.g., linear programming relaxation and semidefinite programming relaxation. Discrete relaxation refers to some discrete optimization problem based relaxation [2]. The discrete relaxation problem should be much easier to solve. In this paper, we propose a discrete relaxation method for layout decomposition for triple patterning lithography (TPL), which is a 0-1 program relaxation.

With the need of industry for more and more smaller cells, the current 193 nm ArF lithography technology cannot meet the need of the IC industry. Extreme Ultra-Violet (EUV) lithography is considered as a promising technology for next-generation lithography. However, due to mask material, light source and other device problems, EUV still cannot be put into IC manufacture. Consequently, multiple patterning lithography (MPL) technology has become the preferred, which decomposes an initial layout into multiple masks for use in multiple exposure lithography. MPL is considered of great values in research and industrial applications [3], [4], [5]. In MPL, layout decomposition is a key step. In this paper, we focus on the layout decomposition for triple patterning lithography.

Layout decomposition for triple patterning lithography is that, an initial layout is decomposed into three masks. Fig. 1a shows an example of decomposing a layout to three masks. For TPL, the rule of minimum coloring spacing is that, if two features are within the minimum coloring spacing $\min_{cs}$, then they should be assigned to different masks; otherwise a conflict occurs between the two features. As shown in Fig. 1b, according to the rule of minimum coloring spacing, a conflict occurs between features a and d. Conflicts can be eliminated by inserting stitches. That is, a feature may be split into several touching sub-features by inserting stitches. As shown in Fig. 1c, a stitch is inserted into feature d. Since both conflict and stitch will affect the effect of lithography, especially the conflict, a crucial issue in TPL is to achieve the minimum numbers of conflicts and stitches.

Before introducing previous works on TPL layout decomposition, it is necessary to introduce some researches on double pattering lithography (DPL). Generally, DPL layout decomposer converts the layout decomposition problem to the 2-coloring problem. Kahng et al. [5] and Yuan et al. [6] constructed integer linear programming (ILP) models, where the number of conflicts and the number of stitches are minimized simultaneously. Xu et al. [7] and Tang et al. [8] used many graph division methods in DPL for reducing the scale of the problem. A method for identifying the native conflicts was proposed by Fang et al. [9]. The method by Tang et. al [8] achieved the most up-to-date results, which are based on a polynomial time minimum cut algorithm.

The TPL layout decomposition problem and the problem with balanced density are NP-complete, which were proved by Yu et al. [10], [11]. Hence most of algorithms for the TPL layout decomposition problem belong to heuristic methods.

• *The authors are with the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350108, China. E-mail: {982670199, 406187567}@qq.com, wxzhu@fzu.edu.cn.*

Manuscript received 7 Jan. 2016; revised 8 June 2016; accepted 11 June 2016. Date of publication 15 June 2016; date of current version 20 Jan. 2017. Recommended for acceptance by K. Chakrabarty. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2016.2582154

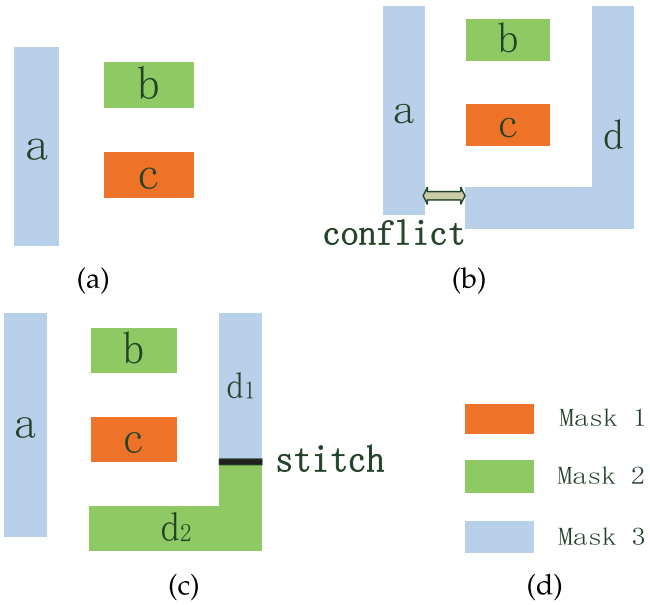


Fig. 1. TPL layout decomposition. (a) An example of three masks assignment. (b) An example of layout with a conflict. (c) An example of layout with inserting a stitch to eliminate conflict.

A special case of the TPL layout decomposition problem is the problem where only row structure and standard cells are considered. A polynomial time heuristic algorithm was proposed by Tian et al. [12] for this problem. This was improved by Chien et al. [13].

For the general TPL layout decomposition problem, there are two kinds of common approaches. The first is the methods operating in the feasible solution space only, and trying to find a least possible upper bound on the minimum value of the TPL layout decomposition problem. Among this kind of methods, Cork et al. [14] proposed an algorithm based on the 3-SAT problem. The heuristic method by Fang et al. [15] uses the line projection method to identify conflicts, and then all edges of the conflict graph are weighted for designing algorithms. At present, the heuristic graph matching method proposed by Kuang et al. [16] achieves the fastest running time, in which a number of graph division methods are introduced for reducing the graph vertex number.

The second approach is the semidefinite program relaxations of the TPL layout decomposition problem by Yu et al. [10], [11], which are continuous relaxations. In [10], Yu et al. formulated the TPL problem as a linear binary program and a vector program. The vector program was further transformed to a semidefinite program for finding a relaxation solution. Generally, a solution of the relaxation problem is infeasible for the TPL problem, and is rounded to a feasible one. An advantage of this method is that, a solution of the relaxation problem provides a lower bound, and the feasible solution by rounding provides an upper bound on the minimum value of the TPL problem. Hence, the solution quality may be estimated. However, since conflicts and stitches are considered simultaneously, the size of the relaxation problem is much larger, and the solution time is much more.

Discrete relaxation has not been proposed for the TPL layout decomposition problem. By considering the advantage of the relaxation method, we propose in this paper a discrete relaxation theory and the theory based layout decomposition framework for TPL. In the framework, conflicts and stitch insertions are considered separately. Our decomposition framework obtains a decomposition solution in two steps. The first step focuses on finding a discrete relaxation solution. Due to our relaxation by graph reduction tricks and stitch insertions not considered at this step, the solution space is dramatically reduced, and we can quickly find an optimal solution of the relaxation problem. At the second step, the relaxation solution is legalized to a feasible solution of the TPL layout decomposition problem. Experimental results and comparisons indicate that our discrete relaxation based method is fast and effective. Specifically, for some test benchmarks, optimal solutions are obtained according to our discrete relaxation theory.

If stitch insertions are not allowed, then the TPL layout decomposition problem is the 3-coloring problem, and our decomposition method still works by deleting stitch insertion operations. Although the techniques used in our method look like heuristic methods, they are carefully adopted or designed for the discrete relaxation purpose. For example, the surface projection method developed is used for finding some features in a layout which will be colored prior. We believe our discrete relaxation idea and techniques could be applied to other problems.

The rest of this paper is organized as follows. Problem formulation, the discrete relaxation theory, and the overview of the TPL layout decomposition framework are stated in Section 2. Section 3 shows the surface projection technique for identifying conflicts and the defined conflict features. This technique is important for our discrete relaxation method. Section 4 introduces the discrete relaxation method. Legalization process is explained in Section 5. Section 6 details the graph reduction methods used in our decomposition framework. The graph reduction is also a kind of discrete relaxation of the TPL layout decomposition problem. Experimental results will be given in Section 7, and we conclude our work finally.

2 Problem Formulation, Discrete Relaxation Theory, and Layout Decomposition Framework

In this section, we present first the TPL layout decomposition problem. Then we propose the discrete relaxation theory based on which our algorithm for the TPL problem is developed. Finally, we give an overview of our TPL layout decomposition framework.

2.1 Problem Formulation

We introduce two definitions as follows:

Definition 1 (conflict graph, CG). *The conflict graph is defined as an undirected graph $G(V, E)$, where V represents the set of vertices, and $v \in V$ represents a feature. E is the set of edges, and $e_{ij} \in E$ exists between two features i and j if the distance between them is less than $\min_{cs}$.*

Definition 2 (sub-feature). *One or more stitches are inserted into a feature, and the feature is split into several parts, namely sub-features.*

The TPL layout decomposition problem is to decompose an initial layout into three masks via possible stitch

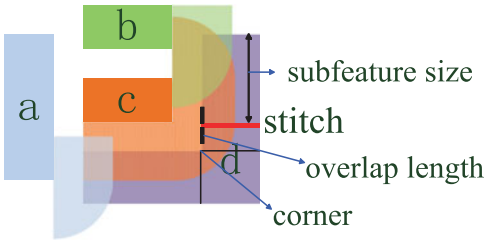


Fig. 2. A sample of legal stitch insertion.

insertions. The objective is minimizing the numbers of conflicts and inserted stitches. Formally, it can be described as follows:

Problem (P_0): *TPL Layout Decomposition Problem*

Given: Layout L, the minimum coloring spacing $\min_{cs}$, the minimum feature size $\min_{fs}$, the minimum overlap margin $\min_{om}$, a constant α ($0 \leq \alpha < 1$).

Find: Stitch insertions in features and assignment of features and sub-features to three masks, such that $|C| + \alpha|S|$ is minimized, where $|C|$ is the number of conflicts occurring and $|S|$ is the number of stitches inserted. The locations of stitches should be legal.

Inserting a stitch legally into a feature means that: (i) the size of the generated sub-features should be larger than the minimum feature size $\min_{fs}$; (ii) an inserted stitch is not near any corner of the feature; and (iii) the overlap length is greater than the minimum overlap margin $\min_{om}$. Here overlap length means that, stitch insertion position can be moved vertically or horizontally without causing any new conflict, and the move length is called the overlap length [16]. An example of a legal stitch is shown in Fig. 2.

The above constraints are due to that, a very small subfeature is easy to generate overlay error [5], [15], [16]; a corner stitch may cause significant side effect on printability; and similarly, if the overlap length of a stitch is less than $\min_{om}$, then the stitch may cause side effect too [5], [15], [16].

In the problem, assigning features to three masks is equivalent to 3-coloring of the conflict graph. Hence in the following, when saying coloring a vertex of the conflict graph, it means assignment of the corresponding feature to a mask, and vice versa.

2.2 Discrete Relaxation Theory

Definition 3 (Discrete relaxation). *Relaxation Problem* (RP)*:*

$$z^R = \min\{f^R(x) : x \in X^R\},$$

is a discrete relaxation of problem (P)*:*

$$z = \min\{f(x) : x \in X\},$$

if there exists an optimal solution $x^{R\star}$ of problem (RP)*, and there exists an optimal solution $x^\star$ of problem* (P) *such that $f^R(x^{R\star}) \leq f(x^\star)$.*

In the above definition, the function f^R and the feasible set X^R must be selected carefully. Generally, we should select an X^R such that an optimal solution of the relaxation problem could be transformed easily to a feasible solution of the original problem (P).

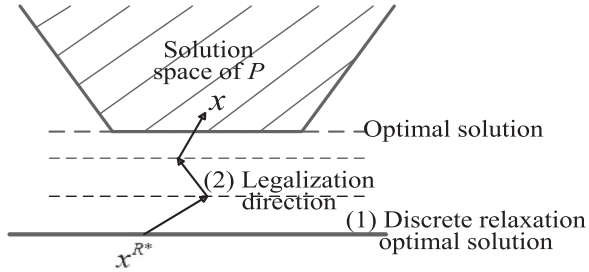


Fig. 3. Geometrical representation of discrete relaxation.

Although Definition 3 can be applied without any assumption on the feasible sets X^R and X, we restrict them to be discrete sets for our usage and call the relaxation as discrete relaxation. By Definition 3, we immediately have:

Proposition 1. *If problem (RP) is a discrete relaxation of problem (P), then $z^R \leq z$.*

Proposition 1 means that, by discrete relaxation we will obtain a lower bound on the minimum value of the original problem. Specifically, we have:

Proposition 2. *Suppose that problem (RP) is a discrete relaxation of problem (P). Let $x^{R\star}$ be an optimal solution of problem (RP). If $x^{R\star}$ can be transformed to a feasible solution x of problem (P), such that $f^R(x^{R\star}) = f(x)$, then x is an optimal solution of problem (P).*

Proof. By Proposition 1 and the assumptions of this proposition, $z^R = f^R(x^{R\star}) = f(x) \leq z$. Since $x \in X$, we have $z \leq f(x)$. So $f(x) = z$, which means that x is an optimal solution of problem (P). □

The principle of using the above discrete relaxation theory is as Fig. 3 shows. First, we formulate a discrete relaxation problem for problem (P) to obtain an optimal solution $x^{R\star}$ of (RP). And then, $x^{R\star}$ is legalized to a feasible solution x of the original problem (P), and we have $f^R(x^{R\star}) \leq f(x)$ by Proposition 1. If $f^R(x^{R\star}) = f(x)$, then by Proposition 2 we obtain an optimal solution x of problem (P).

Although the above discrete relaxation theory looks simple, we carefully design relaxation techniques for the TPL layout decomposition problem, and the computational results are promising.

2.3 Overview of Layout Decomposition Framework

In Fig. 4, we illustrate our decomposition framework for the TPL layout decomposition problem. First, according to the rule of minimum coloring spacing, we transform an initial layout to a conflict graph using our new projection method, i.e., surface projection method. And then, we adopt two techniques to reduce the conflict graph, which is a discrete relaxation. After that, the large scale TPL layout decomposition problem is reduced to numerous small scale TPL layout decomposition subproblems.

At the discrete relaxation coloring step, a 0-1 program is constructed for each reduced TPL problem, which is a discrete relaxation. Then, we solve the 0-1 program to obtain a discrete relaxation solution for each reduced TPL problem. At the discrete relaxation step, stitch insertions are ignored. At the legalization step, stitch insertion and backtrack

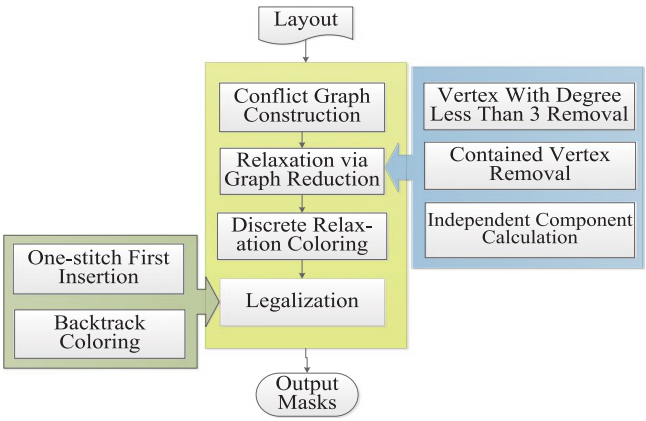


Fig. 4. Our TPL layout decomposition framework.

coloring algorithms are respectively used to legalize the discrete relaxation solutions. Finally, we color the removed features, which are removed at the relaxation by graph reduction step. Details of the decomposition framework are described in the following sections.

3 Surface Projection and K_4 Conflict Structure

In order to eliminate conflict edges in a layout, stitches might be introduced to split a feature into several sub-features. But, conflict edges between some features might not be totally eliminated by inserting stitches. We call these conflict edges as non-resolvable conflicts, and call the corresponding conflict sub-graph as non-resolvable conflict structure.

In this section, we develop a surface projection method for finding the conflict features in a layout. The conflict features will be prior considered at the discrete relaxation coloring step. Using the conflict features, the K_4 conflict structures in a layout can be identified, which have an important property that the conflict edges in the structures cannot be totally eliminated by inserting stitches.

3.1 Surface Projection

Line projection method [5] has been popularly utilized for constructing a conflict graph and finding stitch insertions for the DPL or TPL layout decompositions. Fig. 5a shows an example of line projection of feature b on feature c. However, we want to identify features which are critical and should be colored prior, and this forms a basis of our discrete relaxation method. For this purpose, we develop the surface projection method.

Note that, the relation between two neighboring features is more than their boundaries. Some useful relation may be inside the features. Using the information, we want to identify the features at which non-resolvable conflicts are more likely to exist, which might be colored prior. We develop a surface projection method to mine the information. In order to present the method accurately, some definitions are introduced as follows:

Definition 4 (conflict region, CR). *The conflict region of a feature is defined as a 2D region around the feature but within the minimum coloring spacing* $\min_{cs}$.

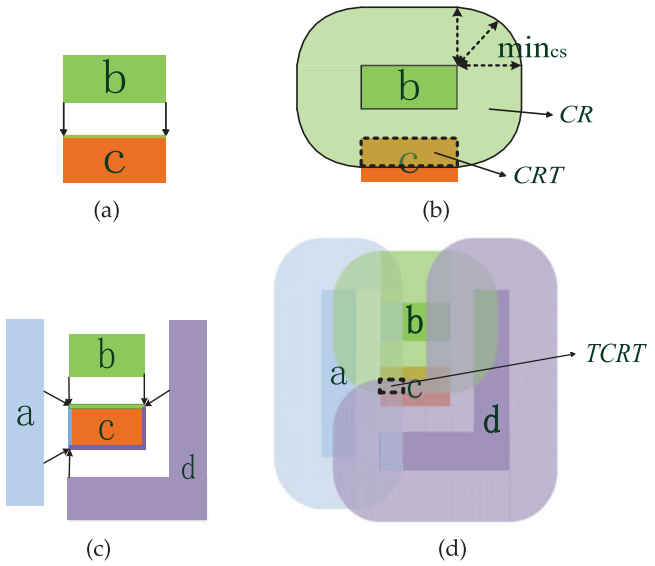


Fig. 5. Surface projection and line projection. (a) Line projection. (b) Surface projection. (c) An example shows that line projection cannot find inside *TCRT*. (d) Surface projection is used to find *TCRT*.

Definition 5 (conflict rectangle, CRT). *The conflict rectangle on feature a by feature b is defined as the minimum rectangle enclosing the intersection of feature a and the conflict region of feature b.*

As illustrated in Fig. 5b, the shaded round region around feature b is called the conflict region (CR) of b, and the dashed box on feature c is the conflict rectangle CRT resulted by b.

Definition 6 (triple conflict rectangle, $TCRT$). *The triple conflict rectangle on feature a is defined as a rectangle, which is the intersection of three or more conflict rectangles (CRTs) on feature a. Some adjacent features of feature a create these conflict rectangles, and these adjacent features are called conflict adjacent features (CAF) of feature a.*

For any feature a, the line projection method can find all possible conflict line segments on the peripheries of a, which are projections of the neighbouring features. However, for some features, line projection may not find all possible conflict regions inside the features, and then, it may not find the triple conflict rectangles inside the features. An example is illustrated as Figs. 5c and 5d.

In Fig. 5c, line projection finds three 1D conflict line segments on the four peripheries of feature c. In Fig. 5d, surface projection finds three 2D conflict regions on feature c, and a triple conflict rectangle ($TCRT$), i.e., dashed box, can be found inside c, which cannot be found by line projection. Detecting triple conflict rectangle is crucial for finding conflict features and $K4$ conflict structures, which will be presented in the next section.

3.2 Conflict Feature and K_4 Conflict Structure

Definition 7 (conflict feature, CF). *Feature a is called a conflict feature if it satisfies the two conditions: (1) there exists a triple conflict rectangle TCRT on feature a; (2) the graph composed of feature a and its conflict adjacent features (CAF) is not three-colorable.*

Definition 8 (K_4 conflict structure, K_4CS). *A graph structure is a K_4 conflict structure, if it is a K_4 structure, and all*

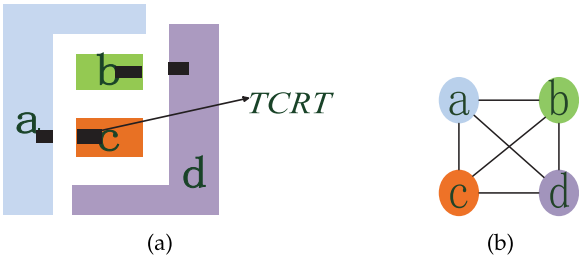


Fig. 6. Conflict features and a K_4 conflict structure.

the four features are conflict features CF and they are CAF each other.

By Definition 7, the feature c in Fig. 5d is a CF, since there is a $TCRT$ on it; and c and its conflict adjacent features a, b and d compose a K_4. Similarly, features a, b, c and d in Fig. 6a are CF too. In addition, the structure composed of features a, b, c and d in Fig. 6a is a K_4 conflict structure K_4CS, in which the $TCRT$s in features b and c cannot be identified by the line projection method. Hence, our surface projection method can be used to locate more unresolvable conflicts, since some conflict structures K_4CS cannot be identified by the line projection method.

For a conflict feature, it has a property as follows.

Lemma 1. *Suppose that feature a is a conflict feature, and features $a_1, a_2, \ldots, a_{k+1}$ are the sub-features of feature a, where k is the number of stitches inserted into feature a. Then at least one of features $a_1, a_2, \ldots, a_{k+1}$ is still a conflict feature.*

Proof. If feature a is a conflict feature, then it has at least a $TCRT$. We insert k stitches into feature a and split it into sub-features $a_1, a_2, \ldots, a_{k+1}$. Suppose that we first insert stitch s_1 into feature a and split it into two parts a_1 and a_2. Then there are two possible results: 1) if stitch s_1 cuts across the $TCRT$ on feature a, we have that both of the sub-features a_1 and a_2 contain a new $TCRT$; 2) otherwise, only one of a_1 and a_2 contains $TCRT$. Next, stitches are inserted into feature a one by one. At last, feature a is split into $k+1$ sub-features $a_1, a_2, \ldots, a_{k+1}$. Obviously, the number of sub-features containing $TCRT$ is nondecreasing. Hence, at least one of the sub-features a_i contains $TCRT$, and the sub-graph consisting of the sub-feature a_i and the conflict adjacent features CAF of feature a is not 3-colorable. By Definition 6, this completes the proof. □

Lemma 1 shows that, inserting stitches into a conflict feature cannot make it be a non-conflict feature. Hence, when coloring, conflict features should be colored prior. Especially, for a K_4 conflict structure, it has the following property.

Theorem 1. *Every K_4 conflict structure exists at least a non-resolvable conflict.*

Proof. Since a K_4CS is a K_4 structure, it is not 3-colorable, which means that for any color assignment, at least a conflict exists in the K_4CS. Furthermore, since all the four features in a K_4 structure are CFs, and by Lemma 1, stitch insertions cannot eliminate the conflict in the K_4CS. Hence a K_4 conflict structure exists at least a non-resolvable conflict. □

Theorem 1 suggests that a K_4 conflict structure exists a conflict which cannot be eliminated by inserting stitches. To find potential conflicts in a layout, it is important to identify the conflict features. Identifying the conflict features can be done by the BFS algorithm, which traverses the conflict graph. Furthermore, for every vertex v with $TCRT$, the algorithm will find all K_4 subgraphs containing v. Suppose that the maximum degree of vertices in a layout is d. Then the runtime of determining whether there exists a subgraph containing v is K_4 is in time $O(d^3)$. Hence, the total runtime complexity for identifying CFs is in time $O(nd^3)$, where n is the number of features in a layout.

4 Discrete Relaxation Method for TPL Layout Decomposition

Since it is hard to solve directly the TPL layout decomposition problem, we propose a discrete relaxation method for finding a lower bound on the minimum value of the problem, basing on the discrete relaxation theory.

In this part, we describe our discrete relaxation method for the TPL layout decomposition problem, which is to obtain a lower bound on the optimal value of the problem.

4.1 Equivalent Formulation of Problem (P_0) to (P_1)

First, we formulate the TPL layout decomposition problem (P_0) to an equivalent problem (P_1). Here, we consider the TPL layout decomposition problem by assigning features to three masks first, and then consider stitch insertions into some features.

Suppose that $G(V, E)$ is decomposed into four subgraphs $G_1(C_1, E_1)$, $G_2(C_2, E_2)$, $G_3(C_3, E_3)$, $G_4(R_4, E_4)$, where E_i is the set of edges between the features in C_i ($i = 1, 2, 3, 4$), respectively, and E_i ($i = 1, 2, 3$) is an empty set. After obtaining the decomposition C_1, C_2, C_3 and R_4, we consider stitch insertions for the features in R_4. Then, the generated sub-features and the unsplitted features in R_4 are assigned to classes C_1, C_2 and C_3 with the minimum total number of conflicts in C_1, C_2 and C_3.

Let $\mathcal{S}(R_4)$ be the set of all possible plans of inserting stitches into the features in R_4. Given C_1, C_2, C_3, $S \in \mathcal{S}(R_4)$, there are many ways of assigning the generated sub-features and the unsplitted features in R_4 to classes C_1, C_2 and C_3. And different way of assignment will produce different conflicts. Here we let $T(C_1, C_2, C_3, S)$ be the minimum total number of conflicts in C_1, C_2 and C_3 among these ways of assignment.

Mathematically, problem (P_0) can be formulated equivalently as the following problem (P_1):

$$\begin{aligned} \min \quad & T(C_1, C_2, C_3, S) + \alpha |S| \\ \text{s.t.} \quad & \text{if } i, j \in C_k, \text{then } e_{ij} \notin E,\ k = 1, 2, 3; \end{aligned} \tag{1a}$$

$$S \in \mathcal{S}(R_4), \tag{1b}$$

where constant α is a weight parameter. In this paper, we take $\alpha = 0.1$, which is as that in Yu et al. [10].

Let X_{P_0}, X_{P_1} be the solution spaces of problems (P_0) and (P_1), and let $f_0(x_{P_0})$, $f_1(x_{P_1})$ be the objective functions of problems (P_0) and (P_1), respectively.

Claim 1. *Problem (P_1) is an equivalent formulation of problem (P_0).*

Proof. Suppose that $x^{\star}_{P_0} = (M^{\star}_1, M^{\star}_2, M^{\star}_3) \in X_{P_0}$ is an optimal solution of problem (P_0), where $M^{\star}_k$ $(k = 1, 2, 3)$ includes the stitch free features and the sub-features produced by stitch insertion plan S. Move all sub-features in $M^{\star}_1$, $M^{\star}_2$, $M^{\star}_3$ to R_4. After that, if there exists a conflict in $M^{\star}_1$, $M^{\star}_2$ or $M^{\star}_3$, move a feature at which the conflict occurs to R_4. Repeating this operation will finally produce a feasible solution (C_1, C_2, C_3, S) of problem (P_1). Obviously

$$f_0(x^{\star}_{P_0}) \geq T(C_1, C_2, C_3, S) + \alpha|S|.$$

Hence the optimal value of problem (P_1) is a lower bound on the optimal value of problem (P_0).

Conversely, an optimal solution of problem (P_1) is a feasible solution of problem (P_0), which means that the optimal value of problem (P_0) is also a lower bound on the optimal value of problem (P_1). Combining the two cases implies that Claim 1 holds. □

4.2 Relaxation of Problem (P_1) to (P_2)

Next, we relax problem (P_1) to a problem (P_2) by discrete relaxation, which is a 0-1 program whose optimal value provides a lower bound on the optimal value of problems (P_1) and (P_0). Problem size of the 0-1 program is significantly less than that of the TPL layout decomposition problem, due to ignoring stitch insertions.

Let (x_{i1}, x_{i2}) be a two dimensional binary variable, which is used to express the color of vertex i. When $(x_{i1}, x_{i2}) = (0, 1)$, it means $i \in C_1$; similarly, when $(x_{i1}, x_{i2}) = (1, 0)$, it means $i \in C_2$; and when $(x_{i1}, x_{i2}) = (1, 1)$, it means $i \in C_3$. However, when $(x_{i1}, x_{i2}) = (0, 0)$, it means that vertex i is uncolored, i.e., $i \in R_4$.

Let w_i be the weight of vertex i. By Theorem 1, we know that a non-resolvable conflict is more likely to exist at a conflict feature CF. Hence a conflict feature CF should be assigned a greater weight. Specifically, let

$$w_i = \begin{cases} 1, & \text{if } i \text{ is conflict feature;} \\ \alpha, & \text{if } i \text{ is not conflict feature.} \end{cases}$$

Then we relax problem (P_1) to the 0-1 program (P_2):

$$\min \quad \sum_{i \in V} w_i(1 - x_{i1})(1 - x_{i2})$$

$$\text{s.t.} \quad x_{i2} - x_{i1} + x_{j2} - x_{j1} \leq 1, \ \forall e_{ij} \in E; \tag{2a}$$

$$x_{i1} - x_{i2} + x_{j1} - x_{j2} \leq 1, \ \forall e_{ij} \in E; \tag{2b}$$

$$x_{i1} + x_{i2} + x_{j1} + x_{j2} \leq 3, \ \forall e_{ij} \in E; \tag{2c}$$

$$(x_{i1}, x_{i2}) \in \{0, 1\} \times \{0, 1\}, \ \forall i \in V. \tag{2d}$$

In the objective function,

$$(1 - x_{i1})(1 - x_{i2}) = \begin{cases} 0, & \text{if } i \notin R_4; \\ 1, & \text{if } i \in R_4. \end{cases}$$

Hence the objective is minimizing the total weight of the vertices in R_4. Constraints (2a)-(2c) are equivalent to constraint (1a), which are used to force that, if $e_{ij} \in E$, then vertices i and j should not be in the same class C_1, C_2 or C_3, respectively. Specifically, for (2a), if vertices i and j are in C_1, then $e_{ij} \notin E$; otherwise, $(x_{i1}, x_{i2}) = (x_{j1}, x_{j2}) = (0, 1)$ violate constraint (2a).

Furthermore, for an optimal solution (C_1, C_2, C_3, R_4) of problem (P_2), the following observation is obvious. Any feature in R_4 conflicts with a feature in C_i, $i = 1, 2, 3$, respectively; otherwise it will be moved to C_i, and we get a better solution.

From the formulation of program (P_2), it can be seen that the difficulty of resolving conflicts is considered by assigning weights to vertices. This difficulty has also been addressed by Fang et al. [15], in which weights are assigned to edges.

Next, we discuss the relationship between problems (P_1) and (P_2). For any optimal solution $x^{\star}_{P_1}$=($C^{\star}_1$, $C^{\star}_2$, $C^{\star}_3$, $S^{\star}$) of problem (P_1), we suppose without loss of generality that, if a feature $i \in R^{\star}_4 = V - C^{\star}_1 \cup C^{\star}_2 \cup C^{\star}_3$ can be moved to $C^{\star}_1$, $C^{\star}_2$ or $C^{\star}_3$ without causing conflicts, then it has been moved.

For the convenience of description, we divide the features in R^*_4 into two classes, UCF and $UNCF$, where UCF is the set of uncolored conflict features, and $UNCF$ is the set of uncolored non-conflict features.

Lemma 2. *For any optimal solution $x^{\star}_{P_1} = (C^{\star}_1, C^{\star}_2, C^{\star}_3, S^{\star}) \in X_{P_1}$ of problem (P_1), if a feature $i \in UCF$, then there exists at least a conflict occurring at i which cannot be eliminated by inserting stitches.*

Proof. Suppose that feature i belongs to UCF. Then i is a conflict feature, $i \in R^{\star}_4$, and at least a conflict will occur at feature i. By Lemma1, if stitches are inserted into feature i, then one or more sub-features of feature i are conflict features, and there still exists at least a conflict. Hence, at least a conflict occurs at feature i which cannot be eliminated by inserting stitches. □

Note that, given an optimal solution $x^{\star}_{P_1} = (C^{\star}_1, C^{\star}_2, C^{\star}_3, S^{\star})$ of problem (P_1), when calculating $T(x^{\star}_{P_1})$, the features and sub-features resulted by stitch insertions $S^{\star}$ of features in $R^{\star}_4$ are assigned optimally to $C^{\star}_1$, $C^{\star}_2$ or $C^{\star}_3$. In this case, some features in R^*_4 will contribute conflicts, we call them Occurring Conflict Features (OCF); and some features in R^*_4 will not contribute conflicts, we call them Not Occurring Conflict Features ($NOCF$). Obviously, $OCF \cup NOCF = R^{\star}_4$, and $|OCF| + |NOCF| = |R^{\star}_4|$.

Let $f_1(x_{P_1})$ be the objective function of problem (P_1), and let $f_2(x_{P_2})$ be the objective function of problem (P_2). By the above assumptions, the following result can be deduced.

Theorem 2. *Given an optimal solution $x^{\star}_{P_1} = (C^{\star}_1, C^{\star}_2, C^{\star}_3, S^{\star})$ of problem (P_1), there exists an optimal solution $x^{\star}_{P_2}$ of problem (P_2) such that $f_2(x^{\star}_{P_2}) \leq f_1(x^{\star}_{P_1})$.*

Proof. Suppose that $x^{\star}_{P_1} = (C^{\star}_1, C^{\star}_2, C^{\star}_3, S^{\star}) \in X_{P_1}$ is an optimal solution of problem (P_1). It is obvious that $|UNCF| + |UCF| = |R^{\star}_4|$. By Lemma 2, we know that every feature in UCF contributes at least a conflict number. Hence $|UCF| \leq |OCF| \leq T(x^{\star}_{P_1})$. Let $|Y| = T(x^{\star}_{P_1}) - |UCF|$. Obviously, $|Y| \geq 0$.

Moreover, every feature $i \in NOCF$ must be inserted $s_i \geq 1$ stitches to eliminate conflicts. So $|NOCF| \leq |S^{\star}|$, and

$$|UNCF| + |UCF| - T(x^{\star}_{P_1}) = |R_4| - T(x^{\star}_{P_1})$$
$$\leq |R_4| - |OCF| = |NOCF| \leq |S^{\star}|.$$

Furthermore, since $0 \leq \alpha = 0.1 < 1$, we have

$$|UCF| + \alpha|UNCF|$$
$$\leq |UCF| + \alpha|UNCF| + (1-\alpha)|Y|$$
$$= |UCF| + |Y| + \alpha(|UNCF| - |Y|)$$
$$= T(x^{\star}_{P_1}) + \alpha(|UNCF| + |UCF| - T(x^{\star}_{P_1}))$$
$$\leq T(x^{\star}_{P_1}) + \alpha|S^{\star}| = f_1(x^{\star}_{P_1}).$$

According to the value of w_i, for the solution ($C_1^{\star}$, $C_2^{\star}$, $C_3^{\star}$, $R_4^{\star}$), it is evidently that

$$f_2(x_{P_1}) = \sum_{i \in V} w_i(1 - x_{i1})(1 - x_{i2}) = |UCF| + \alpha|UNCF|.$$

Hence, Theorem 2 holds. □

By Claim 1, Theorem 2 and the definition of discrete relaxation, it is obvious that problem (P_2) is a discrete relaxation of the TPL layout decomposition problem (P_0). Specifically, we have

Claim 2. *Suppose $x^{\star}_{P_2} \in X_{P_2}$ is an optimal solution of problem (P_2). If $x^{\star}_{P_2}$ is legalized to a feasible solution x_{P_0} of problem (P_0), then*

$$f_2(x^{\star}_{P_2}) \leq f_0(x_{P_0}).$$

Specially, if $f_2(x^{\star}_{P_2}) = f_0(x_{P_0})$, then x_{P_0} is an optimal solution of the TPL layout decomposition problem (P_0).

Furthermore, we have two special cases in which optimality of the TPL layout decomposition problem can be verified.

Claim 3. *Suppose N is the number of K_4 conflict structures in a layout. If there exists a solution x_{P_2} of problem (P_2) with conflict number N, then x_{P_2} is a minimum conflict solution of problem (P_0).*

Proof. Since every K_4 conflict structure contributes at least a conflict, the layout with N K_4 conflict structures exists at least N conflicts. Hence the solution x_{P_2} with conflict number N is a minimum conflict solution of problem (P_0). □

Claim 4. *Let $x^{\star}_{P_2} \in X_{P_2}$ be an optimal solution of problem (P_2). If every feature $i \in UNCF$ can be inserted at most $s_i = 1$ stitch to eliminate conflicts, and every feature $i \in UCF$ contributes at most a conflict number, then $x^{\star}_{P_2}$ together with the stitch insertion into every feature $i \in UNCF$ is an optimal solution of problem (P_0).*

Proof. For the optimal solution $x^{\star}_{P_2}$ of problem (P_2), if every feature $i \in UNCF$ can be inserted at most $s_i = 1$ stitch to eliminate conflicts, and every feature $i \in UCF$ contributes at most a conflict number, then $UCF = OCF$ and $UNCF = NOCF$. Thus the total conflict numbers $|C| = |OCF| = |UCF|$, and

$$|S| = \sum_{i \in NOCF} s_i = |NOCF| = |UNCF|.$$

So

$$|UCF| + \alpha|UNCF| = |C| + \alpha|S|,$$

and

$$|UCF| + \alpha|UNCF| = \sum_{i \in V} w_i(1 - x_{i1})(1 - x_{i2}).$$

Hence, $x^{\star}_{P_2}$ together with the stitch insertion into every feature $i \in UNCF$ is an optimal solution of problem (P_0). □

Problem (P_2) is a nonlinear 0-1 program, which is generally difficult to solve in large scale cases. However, our relaxation via graph reduction techniques proposed in Section 6 can reduce the conflict graph to many small size independent components. Thus problem (P_2) on the small size independent components can be solved easily. Specifically, we adopted the Branch and Bound method in the software package GUROBI [18] to solve the problem on the small size independent components.

Actually, our 0-1 non-linear program (P_2) can be linearized equivalently to the 0-1 linear program (P_3) as follows by introducing new variables and constraints:

$$\min \quad \sum_{i \in V} w_i x_{i3}$$
$$\text{s.t.} \quad x_{i2} - x_{i1} + x_{j2} - x_{j1} \leq 1, \qquad \forall e_{ij} \in E; \tag{3a}$$
$$x_{i1} - x_{i2} + x_{j1} - x_{j2} \leq 1, \qquad \forall e_{ij} \in E; \tag{3b}$$
$$x_{i1} + x_{i2} + x_{j1} + x_{j2} \leq 3, \qquad \forall e_{ij} \in E; \tag{3c}$$
$$1 - x_{i1} - x_{i2} \leq x_{i3}, \qquad \forall i \in V; \tag{3d}$$
$$(x_{i1}, x_{i2}) \in \{0,1\}^2, x_{i3} \in \{0,1\}, \qquad \forall i \in V. \tag{3e}$$

In the above formulation, constraint (3d) is used to force that, if $x_{i1} = 0$ and $x_{i2} = 0$ then $x_{i3} = 1$; otherwise $x_{i3} = 0$, since the objective is minimization and $w_i > 0$. However, we did not adopt the 0-1 linear program (P_3) in our decomposition flow, since it uses more variables and constraints than the 0-1 non-linear program (P_2), and the software package GUROBI [18] is faster on the small scale program (P_2) than on program (P_3).

The 0-1 linear program (P_3) is a formulation of the TPL layout decomposition problem (P_0) without stitch insertions. In [10], Yu et al. proposed an exact 0-1 linear program formulation of the TPL layout decomposition problem (P_0). If forbidding stitch insertions, their formulation is reduced to the following program (P_4),

$$\min \sum_{e_{ij} \in E} c_{ij}$$
$$\text{s.t.} \quad x_{i1} + x_{i2} \leq 1, \qquad \forall i \in V; \tag{4a}$$
$$x_{i1} + x_{j1} \leq 1 + c_{ij1}, \qquad \forall e_{ij} \in E; \tag{4b}$$

$$(1 - x_{i1}) + (1 - x_{j1}) \leq 1 + c_{ij1},\ \forall e_{ij} \in E; \quad (4c)$$

$$x_{i2} + x_{j2} \leq 1 + c_{ij2}, \quad \forall e_{ij} \in E; \quad (4d)$$

$$(1 - x_{i2}) + (1 - x_{j2}) \leq 1 + c_{ij2}, \quad \forall e_{ij} \in E; \quad (4e)$$

$$c_{ij1} + c_{ij2} \leq 1 + c_{ij}, \quad \forall e_{ij} \in E; \quad (4f)$$

$$x_{i1},\ x_{i2},\ c_{ij1},\ c_{ij2},\ c_{ij} \in \{0, 1\}, \quad \forall i, j \in V. \quad (4g)$$

However, it can be seen that program (P_3) uses $3|V|$ variables and $|V| + 3|E|$ constraints, while program (P_4) uses $2|V| + 3|E|$ variables and $|V| + 5|E|$ constraints. Hence, if forbidding stitch insertions, our 0-1 linear program (P_3) uses less variables and constraints than program (P_4).

We solve the discrete relaxation problem (P_2) to obtain a relaxation solution. And the relaxation solution will be legalized to a feasible solution (M_1, M_2, M_3) of problem (P_0) with stitch insertions. The feasible solution (M_1, M_2, M_3) may not be an optimal solution of problem (P_0). However, by Claim 2 or Claim 4, in some cases it will be.

5 Legalization

For every independent component G of the conflict graph got after graph reduction proposed in Section 6, we obtain a relaxation coloring solution from program (P_2). Then, in order to obtain a final mask assignment of TPL layout decomposition, the discrete relaxation solution should be legalized. For an independent component G, the legalization proceeds as follows.

If in the coloring solution (C_1, C_2, C_3, R_4) of problem (P_2), $R_4 \neq \emptyset$, then to eliminate conflicts, we introduce one-stitch first insertions on the features in R_4. Details of the optimal one-stitch insertion are described in Section 5.1. Furthermore, if there exits a feature $i \in UNCF$ which cannot be inserted at most $s_i = 1$ stitch to eliminate conflicts, then the solution obtained from the relaxation coloring solution may not be optimal for problem (P_0). In this case, we propose a backtrack coloring method to obtain another better relaxation solution $(C_1^\star, C_2^\star, C_3^\star, R_4^\star)$. And then we consider dichotomy stitch insertion on the new solution. The details are shown in Section 5.2.

5.1 Stitch Insertion

Due to conflicts, the features in the set R_4 of a relaxation solution are uncolored. And the features might be inserted stitches to eliminate conflicts. Since our objective is that, these features should be colored with the minimum $|C| + \alpha|S|$, we should find stitch insertions into the features in R_4, and then assign their sub-features to the three masks. An algorithm for finding all candidate one-stitch insertions on a feature a is shown as Algorithm 1. Furthermore, if we need to insert multiple stitches into feature a, then we execute Algorithm 1 on the sub-features of feature a iteratively.

In the algorithm, first we find all conflict rectangles on feature a (Line 2). A CRT consists of four edges, each edge of CRT generates a Checking Stitch Edge (CSE). A CSE of feature a is a line segment whose two endpoints are on the boundary of feature a, and feature a could be split into two sub-features as long as the split along the CSE satisfies the following three conditions:

Algorithm 1. Candidate One-Stitches Finding

Input: feature a in R_4, its adjacent features and their colors;
Output: all candidate one-stitch insertions on a;
1: **for** every adjacent feature of feature a **do**
2: calculate the CRTs on feature a caused by its adjacent features, and store CRTs in $CRTSet$;
3: **end for**
4: **for** every $CRT \in CRTSet$ **do**
5: CSEs $= genCSE(CRT)$;
6: store CSEs in $CSESet$;
7: **end for**
8: **for** $CSEs \in CSESet$ **do**
9: if splitting feature a into two sub-features along CSE satisfies **Conditions 1, 2 and 3**, then store the CSE into candidate stitch set $COSSet$;
10: **end for**

Condition 1. Sizes of the generated sub-features should be larger than the minimum feature size $\min_{fs}$;

Condition 2. A candidate stitch insertion is not near a corner of feature a. Overlap length should not be less than the overlap margin $\min_{om}$.

Condition 3. The number of conflict edges connected to generated sub-features is less than the number of conflict edges connected to feature a.

Conditions 1 and 2 are used to make the locations of stitches legal. Condition 3 indicates that inserting a stitch along the CSE will eliminate some conflicts, and then generate a better layout decomposition.

In the algorithm, the function *genCSE(CRT)* refers to generating four CSEs of a conflict rectangle CRT (Lines 4-7). If along a CSE the feature a could be split into two sub-features satisfying the three conditions, then the CSE is a candidate stitch insertion (Lines 8-10).

After obtaining all candidate stitch insertions in $COSSet$, we only keep those candidate stitches which may generate sub-features with the minimum conflict number, and the others are deleted. Then we find an optimal stitch in $COSSet$ for feature a, and split feature a along the optimal stitch into sub-features a_1 and a_2. The optimal stitch is based on a criterion cut_cost for evaluating every stitch, which is the increased number of edges of the conflict graph after splitting feature a along the stitch.

Note that, the sub-features a_1 and a_2 of feature a got by splitting feature a along stitch s will be assigned different colors. Suppose that b_i is an uncolored feature and adjacent to feature a, i.e., $e_{ab_i} \in E$. If $e_{a_1b_i} \in E$ and $e_{a_2b_i} \in E$, then splitting feature a along stitch s will increase the degree of feature b_i by 1, and we let $cut\_cost_{b_i} = 1$. So the function $cut\_cost(s)$ of stitch s is formulated as

$$cut\_cost(s) = \sum_{e_{ab_i} \in E} cut\_cost_{b_i},$$

where

$$cut\_cost_{b_i} = \begin{cases} 1, & e_{a_1 b_i} \in E \text{ and } e_{a_2 b_i} \in E; \\ 0, & \text{otherwise.} \end{cases}$$

Obviously, $cut\_cost(s) \geq 0$. An optimal one-stitch is with the minimum $cut\_cost(s)$. Our empirical experiments indicate that most of features in $UNCF$ have optimal one-stitches with $cut\_cost(s) = 0$.

For every feature in R_4, Algorithm 1 is used to find all candidate one-stitch insertions, and all optimal one-stitch insertions are found based on the above criterion. After that, the features in R_4 with optimal one-stitch insertions are split along their optimal one-stitches respectively, and the obtained sub-features are colored legally. Then the conflict graph G, the sets C_1, C_2, C_3 and R_4 are updated accordingly. If conditions of Claim 4 are satisfied, then the obtained coloring solution is optimal for the TPL layout decomposition problem; otherwise, we perform the process of backtrack coloring in Section 5.2.

5.2 Backtrack Coloring

To obtain a better legal solution, a backtrack coloring algorithm is proposed in this section. Similar backtrack method was introduced by Yu et al. [17] to address the TPL aware detailed placement problem, which is fast and effective on small size graphs. Let $x = (C_1, C_2, C_3, R_4)$ be a relaxation coloring solution of independent component G got by program (P_2). We introduce a set $WOSF$ to represent the features in $UNCF$ which cannot be inserted at most $s_i = 1$ stitch to eliminate conflicts. We try to find a better relaxation coloring solution $x^\star = (C_1^\star, C_2^\star, C_3^\star, R_4^\star)$ of G by Algorithm 2, and then insert stitches into the features in $R_4^\star$.

Algorithm 2. Backtrack Coloring

Input: independent component G, coloring solution $x = (C_1, C_2, C_3, R_4)$ by program (2), subsets X' and X'' of coloring solution space of G;
Output: another coloring solution $x^\star$ of G;
1: **for** every $x' = (C_1', C_2', C_3', R_4) \in X'$ **do**
2: **if** there exist one-stitch insertions such that $WOSF = \emptyset$ **then**
3: $x^\star = x'$;
4: break;
5: **end if**
6: **end for**
7: **if** $WOSF = \emptyset$ **then**
8: **return** $x^\star$;
9: **else**
10: **for** every $x'' = (C_1'', C_2'', C_3'', R_4'') \in X''$ **do**
11: calculate the decomposition cost of x'': $cost(x'') = |C| + \alpha|S|$;
12: **if** find a less cost solution x'' **then**
13: **if** $cost(x'') == \sum_{i \in R_4} w_i$ **then**
14: $x^\star = x''$; Break;
15: **end if**
16: $x^\star = x''$;
17: **end if**
18: **end for**
19: **return** $x^\star$;
20: **end if**

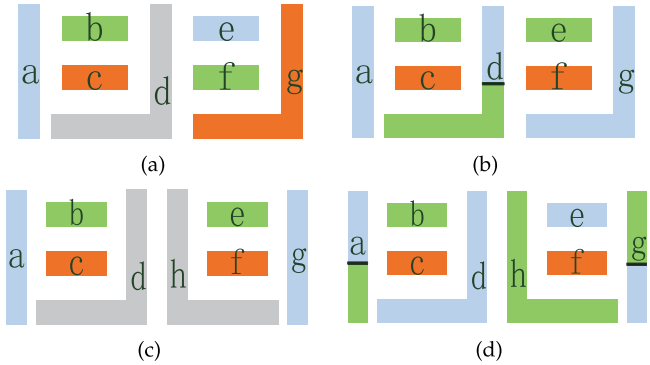


Fig. 7. Two samples of backtrack coloring. (a)(c) Initial illegal coloring solutions. (b)(d) Feasible solutions after backtracking.

In Algorithm 2, inputs X' and X'' are two subsets of coloring solutions of program (2), which are obtained and stored at the solution stage of program (2) by the Branch and Bound method. Every solution $x' = (C_1', C_2', C_3', R_4') \in X'$ satisfies that $R_4' = R_4$, here R_4 is a part of the coloring solution x of program (2), which is an input of Algorithm 2. And every solution $x'' = (C_1'', C_2'', C_3'', R_4'') \in X''$ satisfies that $R_4'' \neq R_4$ and there are no conflict features in R_4''.

Theoretically, the number of solutions in X'' generated by the Branch and Bound method may be exponential. However, there are few non-conflict features in a dense layout. Moreover, for a sparse layout, after graph reduction, the size of every resulting connected component is small. These means that $|X''|$ is not too large and the storage of X'' is tolerable. The situation for X' is similar. To make Algorithm 2 faster, we might only store solutions nearing to the coloring solution x of program (2) in the Branch and Bound process, to X' and X'' respectively.

Algorithm 2 first scan all solutions in X' of G (Lines 1-6). For every new coloring solution (C_1', C_2', C_3', R_4), we enumerate all candidate one-stitch insertions on features in R_4 by Algorithm 1, and check whether the criterion $WOSF = \emptyset$ is satisfied (Line 2). If a solution $x^\star$ with $WOSF = \emptyset$ is found, then a condition of Claim 4 is satisfied, and we stop the backtrack coloring; otherwise, all solutions in X'' will be scanned to find another better one (Lines 10-19), and we have another break criterion to end the scan (Line 13). This criterion means that, if the algorithm has found a solution with $cost(x'') = \sum_{i \in R_4} w_i$, which is the optimal value of problem (2), then it is optimal for problem (1) and it does not need to scan further.

Note that stitch insertions should be found before calculating $cost(x'') = |C| + \alpha|S|$ (Line 11). Here, we consider multiple stitch insertions more than one-stitch insertions for eliminating more conflicts. Inserting multiple stitches into a feature a means that we execute Algorithm 1 on the sub-features of feature a iteratively. Main steps for finding candidate multiple stitch insertions are similar to those in Algorithm 1, and the details are omitted here. Fig. 7 shows two examples of backtrack coloring.

6 Relaxation Via Graph Reduction

In this section we introduce several graph reduction techniques to reduce the conflict graph to small size subgraphs,

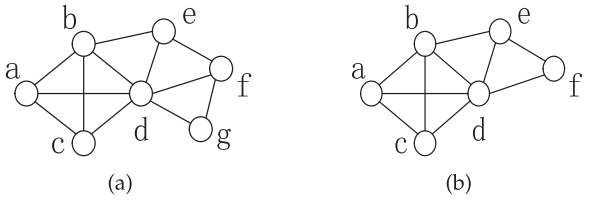


Fig. 8. An example of vertex removal. (a) The initial graph. (b) The remainder graph after removing contained vertex g.

and thus achieve a relaxation of the TPL layout decomposition problem.

Using some rule, we remove some vertices in the initial conflict graph $G(V, E)$, and finally get some small size disconnected subgraphs. After that, we solve the TPL problem on the subgraphs to obtain a mask assignment. Suppose that x is a mask assignment of $G(V, E)$. Obviously, the optimal value of the TPL problem on the subgraphs is not more than the optimal value of the original problem. Hence the mask assignment problem on the subgraphs is a relaxation problem of the mask assignment problem of $G(V, E)$. Theoretical results of this relaxation are similar to those in Section 4.1, and are omitted here.

Removed vertices should be selected carefully from the initial conflict graph. A rule is that, after mask assignment of the subgraphs, the removed vertices can be colored easily. The vertex removing techniques in this paper are listed as follows:

- Vertex with degree less than three removal [10], [15], [16];
- Contained vertex removal.

After vertex removal from the initial conflict graph, independent components must be calculated for constructing subgraphs [10], [15], [16]. Independent components calculation and vertex with degree less than three removal have been used popularly in the previous TPL layout decomposition works. Here, we focus on the contained vertex removal technique. First we introduce the definition of contained vertex [19].

Definition 9 (contained vertex). *Given a graph $G(V, E)$, for a pair of vertices $i, j \in V$, suppose that $e_{ij} \notin E$ and $A(i) \subseteq A(j)$, where $A(i)$ and $A(j)$ are the set of adjacent vertices of vertices i and j, respectively. We call that vertex i is contained in vertex j, vertex i is called a contained vertex, and j is called a containing vertex.*

In Fig. 8a, vertex g is a contained vertex, since $e_{ge} \notin E$, and $A(g) = \{d,\ f\} \subseteq A(e) = \{b,\ d,\ f\}$.

We find a contained vertex and its containing vertices in the graph after the vertex with degree less than three removal, and remove it from the graph. This process is repeated until no contained vertex found. The time complexity of removing all contained vertices is $O(d^3|V|)$, where d is the maximum vertex degree of the graph.

The vertex removals are performed before the discrete relaxation coloring stage. After obtaining the mask assignment of the remainder graph, these removed vertices are still uncolored, and need to be assigned to masks.

A contained vertex is prior assigned to the same mask as one of its containing vertices. For the other removed vertices, we color them in the reverse order of removing them at the vertex removal stage. Since the features in the remainder graph are well colored, it is easy to assign colors to the removed vertices. To achieve final coloring, if needed, we will insert stitches into them or perform backtrack coloring as in Section 5.2 again. In all, the removed features will be assigned to appropriate masks with the minimum number of conflicts or stitches.

Note that a removed vertex might have several optional colors during coloring. That is, no matter which color is assigned to the feature, no conflict will be generated. We call this feature as a color-optional-feature. Since a well balanced decomposition is benefit for manufacturing [11], [20], here we consider global density balance for layout decomposition. We define

$$\text{den} = \frac{\max\{A_1, A_2, A_3\}}{\min\{A_1, A_2, A_3\}},$$

as our density balance measurement, where A_k $(k = 1, 2, 3)$ is the total area of features on the kth mask. During the process of removed feature coloring, a color-optional-feature will be assigned a color which may keep the minimal density balance measurement. This strategy benefits to the global density balance of the three masks.

7 Experimental Results

In order to evaluate our TPL decomposition method, we tested it on the ISCAS-85 & 89 benchmarks provided by Yu et al. [10]. The circuit sizes of the benchmarks range from 1,109 to 168 K. The algorithms were programmed in C++ and run on a personal computer with 2.4 GHz CPU, 16 GB memory and the Linux operating system. For problem (P_2), we adopted the Branch-and-Bound code for nonlinear integer programming in the package GUROBI [18] as our nonlinear 0-1 program solver. For comparing with previous TPL decomposers, we set the parameter values the same as those in previous works. More precisely, the minimum feature size $\min_{fs}$ and the overlap margin $\min_{om}$ were set as 10 nm, and the weight parameter α was set as 0.1.

We performed two experiments to adequately demonstrate the effectiveness of our discrete relaxation based decomposition method. Test results of the first experiment are compared with those of the state-of-the-art TPL decomposers [11], [15], [16]. Test results of the second experiment are compared with those of the state-of-the-art TPL decomposers [10], [15], [21]. Since the binaries of the compared decomposers are not available to us, the test results of the state-of-the-art TPL decomposers are quoted from the respective publications directly.

7.1 First Experiment

In this experiment, we tested our decomposition method on the benchmarks with smaller minimum coloring spacing $\min_{cs}$. More precisely, the minimum coloring spacing $\min_{cs}$ was set as 120 nm for benchmarks C432-C7552, and as 100 nm for benchmarks S1488-S15850. The test results are listed in Table 1.

7.1.1 Analysis of Our Test Results

In Table 1, the data in the columns "#IC" and "#RIC" are the numbers of independent components of the initial conflict

TABLE 1
Test Results of Our Decomposition Method with $\min_{cs} = 120/100$ nm

Bench	#IC	#RIC	Ratio	#UCF	#UNCF	#WOSF	Den
C432	123	4	1.500	0	4	0	1.02
C499	175	0	-	0	0	0	1.37
C880	270	7	1.625	0	7	1	1.09
C1355	467	3	1.500	0	3	0	1.13
C1908	507	1	1.500	0	1	0	1.21
C2670	614	7	1.548	0	6	0	1.25
C3540	827	9	1.500	1	8	0	1.16
C5315	1,154	9	1.500	0	9	0	1.23
C6288	2,325	171	1.559	0	191	16	1.01
C7552	1,783	22	1.511	0	22	1	1.18
S1488	274	2	1.500	0	2	0	1.03
S38417	5,298	74	1.516	19	54	0	1.00
S35932	15,804	84	1.563	44	40	1	1.00
S38584	16,235	152	1.513	36	116	1	1.00
S15850	13,226	131	1.524	34	97	1	1.00

graphs, and the numbers of independent components of the remainder graphs after graph reduction, respectively. The data in the column "ratio" are the ratios between the numbers of edges and the numbers of vertices in the remainder independent components, respectively. From Table 1, the data in the column "#RIC" are small, compared with the data in the column "#IC". Specifically, the average #RIC is only 1.1 percent of #IC. Hence we can conclude that our graph reduction methods are effective.

The data in the column "#UCF" of Table 1 are the numbers of uncolored conflict features by our algorithm on the remainder graphs of the benchmarks. By Lemma 2, we know that #UCF is a lower bound on the minimum conflict number of TPL. The data in the column "#C" of Table 2 are the total conflict numbers found by our method on the benchmarks. Comparing #UCF with #C, it is obvious that both of them are equal for every benchmark. Hence, our decomposition method found results with the minimum conflict numbers for these benchmarks. Moreover, if #UCF=#C, then every feature in UCF contributes at most a conflict number.

The data in the columns "#UNCF" and "#WOSF" of Table 1 are the numbers of uncolored non-conflict features, and the numbers of uncolored non-conflict features without one-stitch insertion, respectively. Table 1 shows that all benchmarks C432-C5315 are with #WOSF=0 except C880, and benchmarks S1488 and S38417 are with #WOSF=0. Since #UCF=#C for benchmarks C432-C5315 except C880 and #UCF=#C for benchmarks S1488 and S38417, every feature in UCF contributes at most a conflict number for these benchmarks. Hence according to Claim 4, we have obtained a minimum cost solution, i.e., we have achieved optimal decomposition costs for these benchmarks.

From the column "Den" in Table 1, it can be seen that most of benchmarks have Den close to 1, i.e., the total areas of the obtained three masks are almost equal. That is, the densities of our results are well balanced.

7.1.2 Comparing with Other TPL Decomposers

In Table 2, we list the test results of our decomposition method and the state-of-the-art TPL decomposers [11], [15], [16], on the benchmarks C432-C7552 and S1488-S15850 with $\min_{cs} = 120/100$ nm. In the table, the data in the columns "#C" and "#S" denote the conflict numbers and the stitch numbers of the final results, respectively. And the data in the column "Cost" are calculated in the same way as that in problem (P_1) and references [11], [15], [16]. The data in the columns "CPU(s)" are the running times of our decomposition method and the decomposers [11], [15], [16], respectively.

We compare in Table 2 our test results with those of decomposers in [11], [15], [16]. From the table, we can see that our method achieves the best TPL decomposition result for every benchmark among the compared decomposers. The last row in Table 2 lists the average #C, #S, Cost and runtime; and lists the Cost, runtime ratios based on the results of our TPL layout decomposition method. Comparing with the results achieved by Kuang et al. [16],

TABLE 2
Experimental Result Comparisons with $\min_{cs} = 120/100$ nm

	From [15]				From [16]				From [11]				Ours			
Bench	#C	#S	Cost	CPU(s)	#C	#S	Cost	CPU(s)	#C	#S	Cost	CPU(s)	#C	#S	Cost	CPU(s)
C432	0	6	0.6	0.01	0	4	0.4	0.01	0	4	0.4	0.2	0	4	0.4	0.01
C499	0	0	0	0.01	0	0	0	0.01	0	0	0	0.2	0	0	0	0.01
C880	1	15	2.5	0.01	0	7	0.7	0.01	0	7	0.7	0.3	0	7	0.7	0.02
C1355	1	7	0.7	0.02	0	3	0.3	0.01	0	3	0.3	0.3	0	3	0.3	0.02
C1908	1	0	0	0.04	0	1	0.1	0.01	0	1	0.1	0.3	0	1	0.1	0.04
C2670	2	14	3.4	0.06	0	6	0.6	0.04	0	6	0.6	0.4	0	6	0.6	0.06
C3540	2	15	3.5	0.08	1	8	1.8	0.05	1	8	1.8	0.5	1	8	1.8	0.07
C5315	3	11	4.1	0.11	0	9	0.9	0.05	0	9	0.9	0.7	0	9	0.9	0.11
C6288	19	341	53.1	0.13	14	191	33.1	0.25	1	213	22.3	2.7	0	204	20.4	0.16
C7552	3	46	7.6	0.17	0	22	2.2	0.1	0	22	2.2	1.1	0	22	2.2	0.17
S1488	0	4	0.4	0.03	0	2	0.2	0.01	0	2	0.2	0.3	0	2	0.2	0.03
S38417	20	122	32.2	0.62	19	55	24.5	0.42	19	55	24.5	7.9	19	54	24.4	0.60
S35932	46	103	56.3	2.13	44	41	48.1	0.82	44	48	48.8	21.4	44	40	48.0	1.70
S38584	36	280	64.0	2.26	36	116	47.6	0.77	37	118	48.8	22.2	36	116	47.6	1.81
S15850	36	201	56.1	2.14	36	97	45.7	0.76	34	101	44.1	20	34	97	43.7	1.73
Avg.	11.3	77.7	19.0	0.52	10.1	37.4	13.75	0.22	9.07	39.8	13.05	5.23	8.9	38.2	12.75	0.44
Ratio			**1.49**	**1.19**			**1.08**	**0.50**			**1.02**	**10.06**			**1.00**	**1.00**

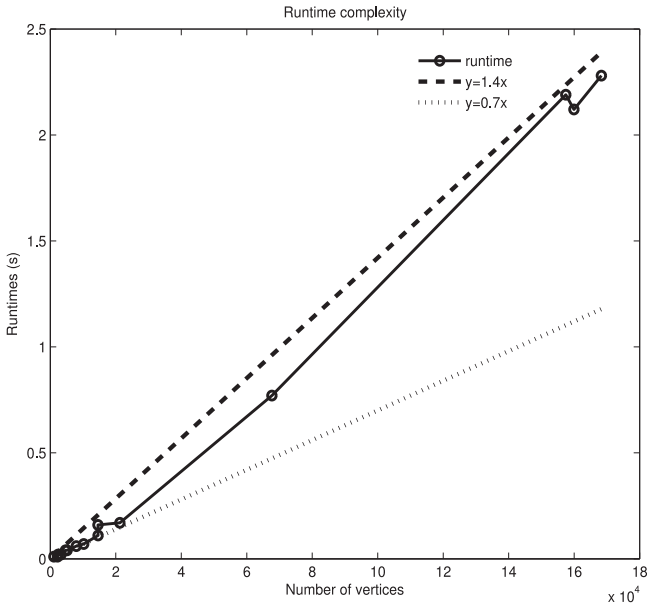


Fig. 9. Running time versus the number of vertices.

which is the fastest decomposer, we reduce the average conflict number by 12 percent, and the average cost by 8 percent. For the state-of-the-art decomposer proposed by Yu et al. [11], the average cost is 2 percent more than ours, and the running time is nine times more than ours. These comparisons validate the effectiveness of our decomposition method.

It must be remarked that the compared decomposers were run on different platforms. In [11], the platform is a personal computer with 3.0 GHz CPU and 32 GB RAM; in [15], the platform is a personal computer with 2.93 GHz CPU and 48 GB RAM; in [16], the platform is a personal computer with 2.39 GHz CPU and 48 GB RAM. But our platform is a personal computer with 2.40 GHz CPU and 16 GB RAM, which is almost the worst.

7.1.3 Scalability of Our Method

To analyze scalability of our decomposition method, Fig. 9 presents the relationship between the number of features and the running time of our decomposition method. The figure is based on our computational results on the ISCAS-85 & 89 benchmarks with $\min_{cs} = 120$ nm. In the figure, the bottom dotted line is the plot of function $y = 0.7x$; the up dotted line is the plot of function $y = 1.4x$; and our runtime picture is the middle solid line. This fully illustrates that our discrete relaxation based decomposition method is almost a linear-time decomposer for the tested benchmarks.

7.2 Second Experiment

To further evaluate effectiveness and scalability of our discrete relaxation based decomposition method, we performed additional experiments of testing our method on the benchmarks S1488-S15850 with $\min_{cs} = 120$ nm, and on the ISCAS-85 & 89 benchmarks with minimum coloring spacing $\min_{cs} = 160$ nm. It is obvious that the numbers of conflict edges of these graphs are more than those in the first experiment, and the conflict graphs of these benchmarks are more dense. The test results of our decomposition method and the compared decomposers are listed in Tables 3 and 4 respectively, where "Im(percent)" denotes the cost reduction percentage by our method comparing with the corresponding method. The data in the columns "CPU(s)" are the runtimes of the algorithm in [21] and our method, respectively. Since the binaries of [10] and [15] are not available to us, we do not list their runtimes.

From Table 3, comparing with [15], [10] and [21], our method achieves considerably cost reduction for every benchmark. Averagely, the cost reduction percentages by our method comparing with the corresponding methods in [15], [10] and [21] are 33.9, 55.6 and 59.6 percent, respectively. From Table 4, the total costs of benchmarks S1488-S15850 by our decomposition method are larger than those in Table 3 respectively, due to the dense structures. The last row of Table 4 lists the average improvements of the cost reduction percentage by our method comparing with the corresponding methods in [15], [10] and [21], which are 35.6, 18.9 and 5.3 percent, respectively. Comparing runtime between [21] and our approach, it can be found that, our approach is faster than the method in [21] on most of the benchmarks, especially on the sparse layouts. It must be remarked that the platform in [21] is a personal computer with 2.66 GHz and 4 GB RAM, which is better than ours.

Finally, for the TPL layout decomposition problem, it is obvious that $\min_{cs}$ is a main parameter to control the density of the conflict graph. In this experiment, we have achieved greater improvement than the first experiment, in which the conflict graph is sparser. This demonstrates that our discrete relaxation based decomposition method is more effective on the dense layouts.

TABLE 3
Experimental Result Comparisons on Dense Layouts with $\min_{cs} = 120$ nm

	From [15]				From [10]				From [21]					Ours			
Bench	#C	#S	Cost	Im%	#C	#S	Cost	Im%	#C	#S	Cost	Im%	CPU(s)	#C	#S	Cost	CPU(s)
S1488	0	4	0.4	50.0	1	1	1.1	81.8	0	30	3.0	93.3	2.15	0	2	0.2	0.04
S38417	43	112	54.2	32.3	61	54	66.4	44.7	47	271	74.1	50.5	45.1	30	67	36.7	0.77
S35932	108	117	119.7	26.8	127	67	133.7	72.5	119	608	179.8	51.3	122	79	86	87.6	2.19
S38584	120	251	145.1	31.1	146	113	157.3	36.4	124	689	192.9	48.2	114	85	150	100.0	2.28
S15850	85	254	110.4	29.3	123	128	135.8	42.6	96	772	173.2	55.0	127	63	150	78.0	2.12
Avg.				**33.9**				**55.6**				**59.6**					

TABLE 4
Experimental Result Comparisons on Dense Layouts with $\min_{cs} = 160$ nm

	From [15]				From [10]				From [21]					Ours			
Bench	#C	#S	Cost	Im%	#C	#S	Cost	Im%	#C	#S	Cost	Im%	CPU(s)	#C	#S	Cost	CPU(s)
C432	94	12	95.2	19.9	89	11	90.4	15.3	79	20	81.0	5.8	2.18	73	33	76.3	0.90
C499	350	17	351.7	19.5	324	29	326.9	13.4	289	49	293.9	3.7	6.15	274	90	283.0	5.49
C880	230	36	233.6	45.0	193	16	194.6	34.1	136	90	145.0	11.5	4.50	116	123	128.3	3.87
C1355	227	50	232.0	41.7	193	29	195.9	30.8	135	87	143.7	5.8	7.13	123	123	135.3	3.55
C1908	287	51	292.1	40.9	210	30	213.0	19.0	176	93	185.3	6.9	10.7	163	95	172.5	2.38
C2670	810	122	822.2	40.1	633	57	638.7	22.9	473	238	496.8	0.8	21.4	463	297	492.7	12.3
C3540	810	156	825.6	46.4	732	62	738.2	40.0	489	349	523.9	15.5	25.2	394	487	442.7	5.85
C5315	1,313	201	1333.1	33.7	1,181	94	1190.4	25.7	933	404	973.4	9.2	38.1	830	529	883.9	10.5
C6288	879	219	900.9	29.2	816	40	820.0	22.3	731	293	760.3	16.2	40.6	603	344	637.4	11.1
C7552	1,585	380	162.3	30.8	1,275	506	1325.6	15.2	1100	642	1164.2	3.5	55.9	1,054	697	1123.7	17.3
S1488	615	108	625.8	24.3	602	34	605.4	21.8	443	215	464.5	−2.0	10.4	446	277	473.7	27.8
S38417	7,748	534	7801.4	40.4	4,908	952	5003.2	7.1	4,473	2,408	4713.8	1.4	197	4,394	2,524	4646.4	66.5
S35932	23,767	1,404	23907.4	39.6	14,412	3,120	14724.0	1.9	13,101	7,137	13814.7	−3.8	722	13,751	6,921	14443.1	813
S38584	20,106	1,392	20235.2	44.7	11,622	2,639	11885.9	5.8	11,187	6,097	11796.7	5.1	654	10,564	6,301	11194.1	102
S15850	22,561	1,915	22752.5	37.8	15,196	2,613	15457.3	8.4	13,405	6,997	14104.7	−0.3	713	13,425	7,282	14153.2	591
Avg.				**35.6**				**18.9**				**5.3**					

8 Conclusion

In this paper, we have proposed a discrete relaxation theory, and have developed a discrete relaxation based decomposition framework for the TPL layout decomposition problem. Although the line projection method can construct the conflict graph of the problem, we have developed a surface projection method for identifying features which are critical and should be colored prior, and this forms a basis of our discrete relaxation method.

To solve the TPL layout decomposition problem, our discrete relaxation based decomposition method relaxes the problem in two steps. First, the conflict graph is reduced to small size subgraphs by two graph reduction techniques, which is a discrete relaxation of the TPL problem. After that, the TPL problem on the small subgraphs is relaxed to a nonlinear 0-1 programming problem by ignoring stitch insertions and assigning weights to features. To legalize an optimal solution of the relaxation problem to a feasible one of the TPL layout decomposition problem, some techniques have been carefully adopted, e.g., the one-stitch first insertion, backtrack coloring. Experiments on the tested benchmarks show that our decomposition method is efficient and effective, compared with the state-of-the-art decomposers. Moreover, by our theoretical results, we have obtained optimal decompositions for some benchmarks. The developed discrete relaxation based decomposition method for TPL is successful. We believe the idea can be applied to other problems.

Acknowledgments

The authors would like to thank the anonymous referees, whose suggestions and comments helped improving the quality of the manuscript greatly. This work was supported by the National Natural Science Foundation of China.

References

[1] E. Chlamtac and M. Tulsiani, "Convex relaxations and integrality gaps," in *Handbook on Semidefinite, Conic and Polynomial Optimization*, M. F. Anjos and J. B. Lasserre, Eds. New York, NY, USA: Springer, 2012, pp. 139–169.

[2] R. Borndorfer and R. Weismantel, "Discrete relaxations of combinatorial programs," *Discrete Appl. Math.*, vol. 112, pp. 11–26, 2001.

[3] ITRS. (2014, Oct.). [Online]. Available: http://www.itrs.net/

[4] Y. Borodovsky, "Lithography 2009: Overview of opportunities," in *SemiCon West*, 2009.

[5] A. B. Kahng, C. H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Proc IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2008, pp. 465–472.

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

[7] Y. Xu and C. Chu, "GREMA: Graph reduction based efficient mask assignment for double patterning technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2009, pp. 601–606.

[8] X. Tang and M. Cho, "Optimal layout decomposition for double patterning technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, pp. 9–13, 2011.

[9] S. Y. Fang, S. Y. Chen, and Y. W. Chang, "Native-conflict and stitch-aware wire perturbation for double patterning technology," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 5, pp. 703–716, May 2012.

[10] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE Trans. Comput.-Aided Des. Integr. Circuits. Syst.*, vol. 34, no. 3, pp. 433–446, Mar. 2015.

[11] B. Yu, Y. H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2013, pp. 163–169.

[12] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and D. F. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2012, pp. 57–64.

[13] H.-A. Chien, S.-Y. Han, Y.-H. Chen, and T.-C. Wang, "A cell-based row-structure layout decomposer for triple patterning lithography," in *Proc. Int. Symp. Physical Des.*, 2015, pp. 67–74.

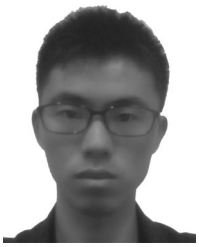
[14] C. Cork, J. C. Madre, and L. Barnes, "Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns," in *Proc. SPIE Next-Generation Lithography Mask Tech. XV*, vol. 7028, 2008, Art. no. 702839.

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

[16] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2013, pp. 1–6.

[17] B. Yu, et al., "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 5, pp. 726–739, May 2015.

[18] GUROBI. (2015, Mar.). [Online]. Available: http://www.gurobi.com/html/academic.html/
[19] C. Lucet, F. Mendes, and A. Moukrim, "Pre-processing and linear-decomposition algorithm to solve the k-colorability problem," *Lecture Notes Comput Sci.*, vol. 3059, pp. 315–325, 2004.
[20] Y. Lin, X. Xu, B. Yu, R. Baldicky. and D. Z. Pan, "Triple/quadruple pattering layout decomposition via novel linear programming and iterative rounding," in *Proc. SPIE Des.-Process-Tech. Co-optim. Manufacturability X*, 2016, Art. no. 97810M.
[21] 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 Des.*, 2013, pp. 170–177.



Xingquan Li received the BSc degree in applied mathematics from the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China, in 2013. He is currently working toward the PhD degree with the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou, China. His research interests include VLSI design for manufacture, VLSI placement, and optimization theory and methods.

Ziran Zhu received the BSc degree in computer science from the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China, in 2015. He is currently working toward the MSc degree at the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou, China. His research interests include VLSI placement and global routing.

Wenxing Zhu received the BSc degree in applied mathematics and the MSc and PhD degrees in operations research and cybernetics, all from Shanghai University, Shanghai, China, in 1989, 1992, and 1996, respectively. Since 1996, he has been with Fuzhou University, where he has been a professor with the Center for Discrete Mathematics and Theoretical Computer Science since 2004. His research interests include optimization theory and methods, algorithms for VLSI physical design automation. He is the author or coauthor of more than 40 scientific papers in refereed international journals, including the *IEEE Transactions on Computers*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *IEEE Transactions on Evolutionary Computation*, the *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, and the *INFORMS Journal on Computing*.