Layout Decomposition with Pairwise Coloring and Adaptive Multi-Start for Triple Patterning Lithography

YE ZHANG, WAI-SHING LUK, and YUNFENG YANG, Fudan University
HAI ZHOU, Northwestern University
CHANGHAO YAN, Fudan University
DAVID Z. PAN, University of Texas at Austin
XUAN ZENG, Fudan University

In this article we present a *pairwise coloring* (PWC) approach to tackle the layout decomposition problem for *triple patterning lithography* (TPL). The main idea is to reduce the problem to a set of bi-coloring problems. The overall solution is refined by applying a bi-coloring method for pairs of color sets per pass. One obvious advantage of this method is that the existing *double patterning lithography* (DPL) techniques can be reused effortlessly. Moreover, we observe that each pass can be fulfilled efficiently by integrating an SPQR-tree-graph-division-based bi-coloring method. In addition, to prevent the solution getting stuck in the local minima, an *adaptive multi-start* (AMS) approach is incorporated. Adaptive starting points are generated according to the vote of previous solutions. The experimental results show that our method is competitive with other works on both solution quality and runtime performance.

Categories and Subject Descriptors: B.7.2 [**Hardware, Integrated Circuits, Design Aids**]: Layout

General Terms: Algorithms, Design

Additional Key Words and Phrases: Adaptive multi-start, design for manufacturability, layout decomposition, pairwise coloring, triple patterning lithography

ACM Reference Format:
Ye Zhang, Wai-Shing Luk, Yunfeng Yang, Hai Zhou, Changhao Yan, David Z. Pan, and Xuan Zeng. 2015. Layout decomposition with pairwise coloring and adaptive multi-start for triple patterning lithography. ACM Trans. Des. Autom. Electron. Syst. 21, 1, Article 2 (November 2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2764904

1. INTRODUCTION

According to the International Technology Roadmap for Semiconductors (ITRS) Report 2013, the critical dimension will achieve 13nm in 2017. To meet the critical manufacture requirement, various next-generation lithography systems have been proposed.

This article extends an earlier version that appeared at the 32nd International Conference on Computer-Aided Design (ICCAD2013).
This work was supported in part by the National Natural Science Foundation of China under Project 61125401, Project 61376040, Project 61274032, in part by the National Basic Research Program of China under Grant 2011CB309701, in part by the Shanghai Science and Technology Committee under Project 13XD1401100, in part by NSF under CCF-1115550, CCF-1218906 and CNS-1441695, and in part by SRC under 2014-TS-2559.
Authors' addresses: Y. Zhang, W.-S. Luk, Y. Yang, the State Key Laboratory of ASIC and System, School of Microelectronics, Fudan University, Shanghai 200433, China; email: yezhang12@fudan.edu.cn; H. Zhou, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL; C. Yan, the State Key Laboratory of ASIC and System, School of Microelectronics, Fudan University, Shanghai 200433, China; D. Z. Pan, Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX; X. Zeng (corresponding author), the State Key Laboratory of ASIC and System, School of Microelectronics, Fudan University, Shanghai 200433, China.

DOI: http://dx.doi.org/10.1145/2764904

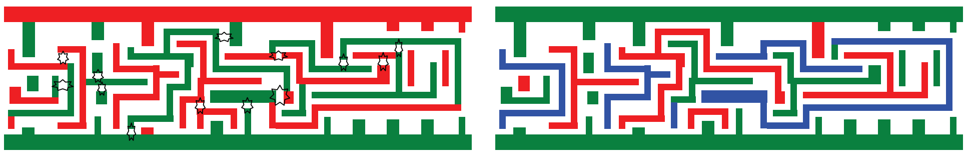

(a) double patterning (b) triple patterning

Fig. 1. Multiple patterning: features in different colors are printed in different lithograph steps. Conflict regions are indicated by star shapes.

They include *multiple patterning lithography* (MPL), *extreme ultraviolet* (EUV) lithography, and *electron beam* (E-beam) lithography. Currently, the implementation of MPL remains challenging. However, due to the delay of availability of EUV and E-beam for mass-volume chip production, MPL is still regarded as the most promising solution for the next technology node.

One of the biggest challenges of MPL is how to split a layout into k nonconflicting masks, where double and triple patterning correspond to $k = 2$ and 3, respectively. An example of layout decomposition for *double patterning lithography* (DPL) in a *litho-etch-litho-etch* (LELE) process is shown in Figure 1(a). Two different colors in the figure correspond to two different masks which are printed in two separate lithography steps. The main principle of the color assignment is that, given a minimum coloring spacing d, there exists a conflict between two nontouching features if the distance between them is less than d. Therefore they should be assigned different colors. The color assignment problem can be formulated as the MIN k-PARTITION problem [Ausiello 1999, GT32] or its dual, the MAX k-CUT problem [Ausiello 1999, ND17].

For DPL, many innovative bi-coloring methods have been proposed to tackle this problem, such as the *integer linear program* (ILP) approach [Kahng et al. 2008; Yuan et al. 2010; Sun et al. 2011], *graph-theoretic approach* [Yang et al. 2010; Xu and Chu 2010; Luk and Huang 2010]. The ILP approach is known to be time consuming, whereas the quality of the results achieved with the graph-theoretic approach cannot be guaranteed. Parallel computing was also incorporated to reduce runtime and memory consumption [Zhao et al. 2014]. Nevertheless, conflicts cannot be avoided whenever odd cycles exist in the conflict graph. To further resolve the conflicts, the stitch insertion [Kahng et al. 2008] and post-coloring legalization techniques [Ghaida et al. 2013] have also been proposed.

As the technology node shrinks further down to 13nm and beyond, DPL will not be sufficient. Thus, *triple patterning lithography* (TPL) is proposed in which one more mask is introduced. Figure 1(b) shows an example of layout decomposition for TPL in a *litho-etch-litho-etch-litho-etch* (LELELE) process where all conflicts can be resolved. From the problem complexity point of view, the color assignment problem for TPL is more “difficult” to solve than that for DPL, in the sense that the MAX k-CUT problem for $k = 3$ has no known polynomial-time solution even for planar graphs. Furthermore, the MAX k-CUT problem for bipartite graphs and $k = 2$ can be solved in linear time, whereas the problem for tripartite graphs and $k = 3$ is NP-hard. As a result, not all DPL techniques can be directly extended for TPL. In Yu et al. [2011], an approximation algorithm is proposed which is based on the *semidefinite programming* (SDP) relaxation, which is extended for $k > 3$ in Yu and Pan [2014]. The effectiveness of the algorithm relies on the underlying numerical SDP solver. In Fang et al. [2012, 2014], the authors proposed a heuristic algorithm which is based on the *recursive-largest-first* (RLF) algorithm. Another previous work [Tian et al. 2012] proposed an algorithm which is specialized for cell-based row structure layouts. In Kuang and Young [2013], a layout decomposition approach was implemented by building a coloring library. In Yu et al.

[2013], Tian et al. [2013], and Chen et al. [2013], the color balancing was considered as well.

This article presents a *pairwise coloring* (PWC) approach for solving the TPL layout decomposition problem. The main idea is to reduce the problem to a set of bi-coloring problems. The overall solution is refined by applying a bi-coloring method for pairs of colors per pass. This idea has been exploited, for example, in circuit partitioning [Cong and Lim 1998]. One major advantage of the PWC approach is that existing bi-coloring techniques can be reusable. Any improvement of them can directly benefit the TPL counterpart. Nevertheless, two questions about this approach have to be addressed. The first is that whether its runtime performance is competitive with other methods. To answer this, we observe that each pass can be fulfilled in a reasonable time by integrating an SPQR-tree-graph-division-based bi-coloring method into this framework. The second question is that whether the solution will get stuck in the local minima. To deal with this, an *adaptive multi-start* (AMS) approach is adopted, where adaptive starting points are generated according to the vote of previous solutions. The effectiveness of our method will further be demonstrated with the experimental results.

In this article, we make the following contributions.

(1) We solve the layout decomposition problem for TPL with a PWC approach where the existing bi-coloring techniques can be reused.
(2) In order to make the PWC approach applicable in practice, we integrate an SPQR-tree-graph-division-based bi-coloring method to improve the runtime performance.
(3) We incorporate an AMS approach in this framework. Consequently, the solutions can largely avoid getting stuck in the local minima.
(4) Experimental results show that our proposed method is effective.

The remainder of this article is organized as follows. First, the layout decomposition problem formulation for MPL is given in Section 2. An overview of our layout decomposition tool is presented in Section 3. After that, we present the layout decomposition methods for TPL in Section 4. Experimental results are provided in Section 5. Our conclusions are given in Section 6.

2. PROBLEM FORMULATION

Given a layout that consists of features represented by polygonal shapes, features are first fractured into rectangles [Kahng et al. 2008]. Although this process will increase the problem size, the reasons it is preferable are three-folds:

(1) since a rectangle is easier to handle than a polygon, this process can simplify the later algorithms such as plane sweeping and stitch insertion;
(2) the process introduces stitch candidates naturally;
(3) self-conflict features can be more easily detected.

In Section 4.1, a simple layout fracturing method that reduces the number of rectangles will be given. Note that stitch insertion techniques could further be applied if necessary.

A provision that a set of rectangles that represents a fractured layout is given. Given a minimum coloring spacing d, there exists a conflict between two nontouching rectangles if the distance between them is less than d (more precise description of conflict detection will be given in Section 4.1). If two rectangles are touching each other, we assume they belong to the same feature and that there exists a *stitch candidate*. A *conflict graph* $G = (V, E)$ is then constructed by the following. Each vertex $v \in V$ represents a rectangle in a layout. Each edge $e \in E$ represents a conflict or a stitch candidate between two rectangles. A weight function w is associated with each edge. If two rectangles are in conflict, then the corresponding edge is assigned a positive

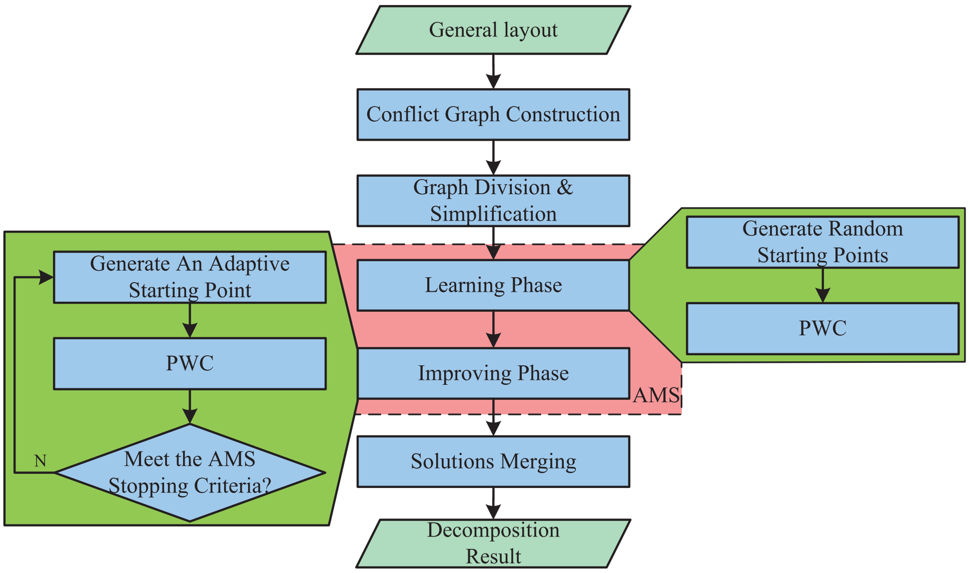

General layout
Conflict Graph Construction
Graph Division & Simplification
Learning Phase
Improving Phase
AMS
Solutions Merging
Decomposition Result
Generate An Adaptive Starting Point
PWC
Meet the AMS Stopping Criteria?
N
Generate Random Starting Points
PWC

Fig. 2. The overall flow of our layout decomposition method for TPL.

weight. In addition, a negative weight is assigned to each pair of touching rectangles so that the same color is preferred, hence minimizing the number of stitches. The layout color assignment problem for MPL can now be formulated as follows.

—INSTANCE: Graph $G = (V, E)$ and a weight $w : w \rightarrow \mathbb{R}$.
—SOLUTION: A color assignment $c : V \rightarrow [1..k]$.
—MINIMIZE: the weight of the monochromatic edges,

$$\sum_{(v_1,v_2)\in E:c(v_1)=c(v_2)} w(v_1, v_2).$$

DPL and TPL correspond to the cases of $k = 2$ and 3, respectively. The problem is the same as the MIN k-PARTITION problem described in Ausiello [1999, GT32] which is known to be NP-hard in general. If G is a planar graph, it is polynomial-time solvable for $k = 2$, but still NP-hard for $k > 2$. Furthermore, if G is a bipartite graph, the problem can be solved in linear time for $k = 2$. Thus, the problem for $k = 2$ can be recast as finding a bipartite subgraph $G' = (V, E - E_c)$ where E_c is a set of edges to be deleted. In DPL, it is more convenient to keep track of E_c instead of the whole color solution, since the size of E_c is practically small. More importantly, *color flipping* can be avoided when a graph division method is applied. Nevertheless, the technique cannot be applied to TPL directly, due to the fact that the problem is still NP-hard when G is a tri-colorable (tripartite) graph and $k = 3$ (c.f. Yu et al. [2011]).

3. OVERVIEW OF OUR LAYOUT DECOMPOSITION FLOW

As certain techniques can only be applied for $k = 2$, the PWC approach for TPL is presented, which is essentially a local search metaheuristic. In order to avoid getting stuck in the local minima, the AMS approach is also incorporated. The overall flow of our method is shown in Figure 2. The polygonal-shaped features are fractured into rectangles and a conflict graph is constructed based on d. More details are presented in Section 4.1. The layout decomposition problem for TPL is then transformed into a tri-coloring problem. By coloring all vertices with three colors, rectangle features are assigned to three masks.

First the conflict graph is divided into biconnected components, each of which is then simplified (presented in Section 4.4). Consequently, the problem size can be reduced.

Afterwards, the tri-coloring problem for each biconnected component is independently solved using the AMS approach. There are two phases in the AMS approach: the *Learning Phase* and the *Improving Phase*. In the Learning Phase, a set of random starting points are constructed such that each vertex is randomly assigned to one of the three colors with equal probability. Then, from each starting point, the PWC approach is conducted to achieve the corresponding solution. The solutions build up the "experience". In the Improving Phase, the starting points are adaptively generated according to the vote of previous solutions. The generated starting points are more likely to lead to better solutions [Boese et al. 1994]. The new solutions achieved with the PWC approach from the adaptive starting points update the "experience" as well. The Improving Phase will exit if the AMS stopping criteria are met. The achieved best solution is selected as output. Section 4.3 provides more details about the AMS approach.

In both of the phases, given a starting point, the corresponding solution is achieved with the PWC approach. In this approach, the tri-coloring problem is reduced to a set of bi-coloring problems, each conquered with a bi-coloring algorithm. The adopted bi-coloring algorithm is based on the SPQR-tree technique (described in Section 4.5.1) and Hadlock's algorithm (described in Section 4.5.2). If the bi-coloring problem can be solved well, the quality of overall tri-coloring solution might be improved after a PWC pass (defined in Section 4.2). The PWC passes continue until the stopping criteria are met. More details about the PWC approach are presented in Section 4.2.

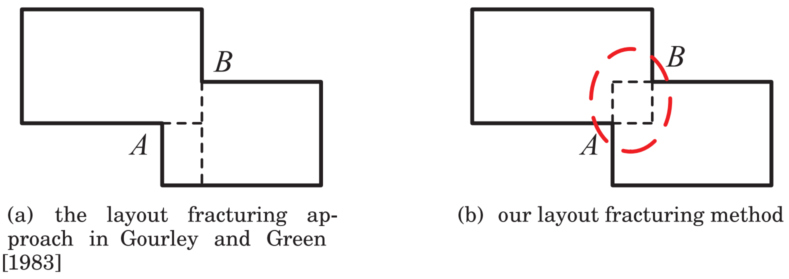
Finally the outputs of all biconnected components are merged in linear time with no solution-quality degradation.

4. LAYOUT DECOMPOSITION FOR TPL

In this section, the TPL layout decomposition techniques are presented. We first perform layout fracturing and conflict graph construction. The TPL layout decomposition is formulated as a tri-coloring problem. Then the problem is solved with the PWC and AMS approaches. Besides, graph division and simplification techniques are also incorporated for TPL.

4.1. Layout Fracturing and Conflict Graph Construction

We employ a layout fracturing algorithm based on Gourley and Green [1983] that fractures the polygonal features into rectangle pieces. The algorithm is simple and works as follows. Let us say a vertex with internal angle equals 270° a fracture vertex. The algorithm basically looks for any fracture vertex, and then makes a cut from it to its nearest line segment. Because each cut eliminates one fracture vertex, the algorithm terminates when no more fracture vertex is found. The algorithm runs in $O(N^2)$ time, where N is the number of vertices of a given polygon. However, the algorithm may generate slivers which are undesired in our application. One example is shown in Figure 3(a). The polygonal feature is fractured into three rectangles, the smallest one a sliver. Based on Gourley and Green [1983], we present a slightly modified version which can eliminate the sliver by allowing overlap between rectangle pieces. First for each fracture vertex A, we check whether there is another nearest fracture vertex B in its "opposite" direction. If not, then we simply follow the procedure of the original algorithm. Otherwise, two cuts with overlapped area are generated as indicated by the dashed circle in Figure 3(b). Note that the overlapping area also partly alleviates manufacturing overlay error. As shown in Figure 3(a), with manufacturing overlay



(a) the layout fracturing approach in Gourley and Green [1983]

(b) our layout fracturing method

Fig. 3. Layout fracturing methods comparison.

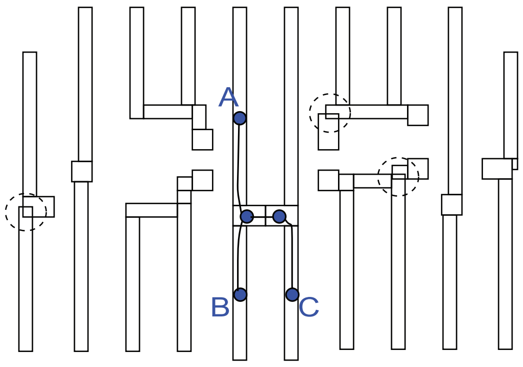


Fig. 4. Layout fracturing and conflict detection. Overlapped regions are indicated by dashed circles.

error, the circuit could be easily disconnected. By introducing an overlapping area, it will maintain connected if the overlay error is not severe.

The conflict graph construction is presented as follows. Each layout rectangle is first enlarged by moving each side outward by $d/2$. Since two rectangles are in conflict only if their enlarged counterparts are overlapped, we can then apply a plane sweeping method that screens out most of the unnecessary conflict detection. With the help of a data structure named dynamic priority search tree [McCrelght 1985], the sweeping process can be performed in $O(n \log n)$ time, where n is the number of rectangles. After shrinking back each rectangle, further examinations of conflict are performed as follows. If two rectangles are not touching each other and their distance is less than d, then we examine whether two rectangles belong to the same polygon. If they do, then we examine whether this is the case that they form a rectilinearly concave shape along their path in the polygon. If it is the case, then an edge with a positive weight is created for the two rectangles. For example, as shown in Figure 4, rectangles B and C are in conflict, whereas rectangles A and C are not even though their minimum distance is less than d. Note that a rectilinear polygon is called rectilinearly convex if it is both x-monotone and y-monotone. Otherwise, it is called rectilinearly concave. Two examples are presented in Figure 5. If two rectangles are touching each other, we create an edge with negative weight so that two rectangles having the same color are preferred and hence minimize the number of stitches. Note that further checking such as *design rule checking* (DRC) could also be performed if necessary.

4.2. Pairwise Coloring Method

After the conflict graph construction, the PWC approach is executed to perform local search. To describe the PWC approach, we first define some terms.

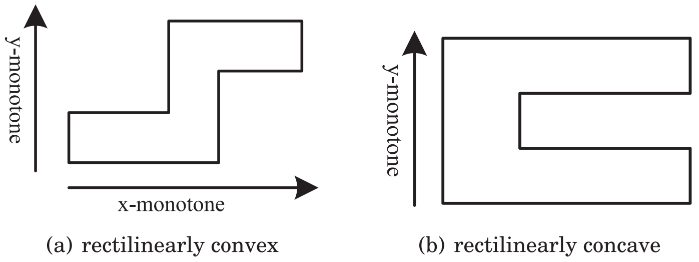


Fig. 5. Examples for rectilinearly convex and rectilinearly concave.

Definition 4.1 (*Color Set*). A color set is a set of vertices which are assigned a designated color.

Definition 4.2 (*Pairing Colors*). Pairing colors are a pair of designated color sets to which a bi-coloring process is executed to assign vertices.

Assume there are three color sets: R (red), G (green), and B (blue). All pairing colors are: $\{R, G\}$, $\{G, B\}$ and $\{B, R\}$.

Definition 4.3 (*Bi-Coloring Conflict Subgraph*). Given a conflict graph, $G = (V, E)$, and a tri-coloring solution. The bi-coloring conflict subgraph $G_c = (V_c, E_c)$ of certain pairing colors (e.g., $\{R,G\}$) is that:

—$V_c = R \cup G$,
—$\forall\, u, v \in V_c, e = uv \in E_c$ if $e \in E$.

Definition 4.4 (*PWC Pass*). During a PWC pass, three bi-coloring processes are executed sequentially with different pairing colors on the corresponding bi-coloring conflict subgraphs.

An example is presented in Figure 6 to illustrate the definitions. Figure 6(a) shows a conflict graph after tri-coloring. The conflicts and stitch candidates are represented by the solid and dashed lines, respectively. As shown in Figure 6(b), the bi-coloring conflict subgraph of $\{R,G\}$ is induced by the vertices colored red and green.

The PWC approach is essentially a local search metaheuristic. Its main idea is to reduce the tri-coloring problem to a set of bi-coloring problems for the corresponding bi-coloring conflict subgraphs. A bi-coloring is limited between the pairing colors, but the overall solution can be improved during each PWC pass. The order of bi-coloring processes can be arranged in many ways. In this article, we select the simplest order as shown in Figure 7. A PWC pass consists of three bi-coloring processes executed with $\{G,B\}$, $\{B,R\}$ and $\{R,G\}$ sequentially. After each pass, the cost of the tri-coloring problem, defined in Section 2, is evaluated. If the cost is the smallest in this run, the corresponding solution is updated. The PWC passes continue until the stopping criteria are met.

An example of the PWC approach is presented in Figure 8 to illustrate the solution improvement process. Figure 8(a) shows a conflict graph. The solid and dashed edges indicate the conflicts and stitch candidates, respectively. A random starting point is presented in Figure 8(b), where there exist three conflicts and two stitches. As the flow shown in Figure 7, Figure 8(c) to 8(f) present the solutions after each bi-coloring process. In this simple example, the solution is quickly achieved as shown in Figure 8(f).

A similar method was used in Ghaida et al. [2012]. However, in order to make it applicable in practice, there are three challenges that must be considered in this approach:

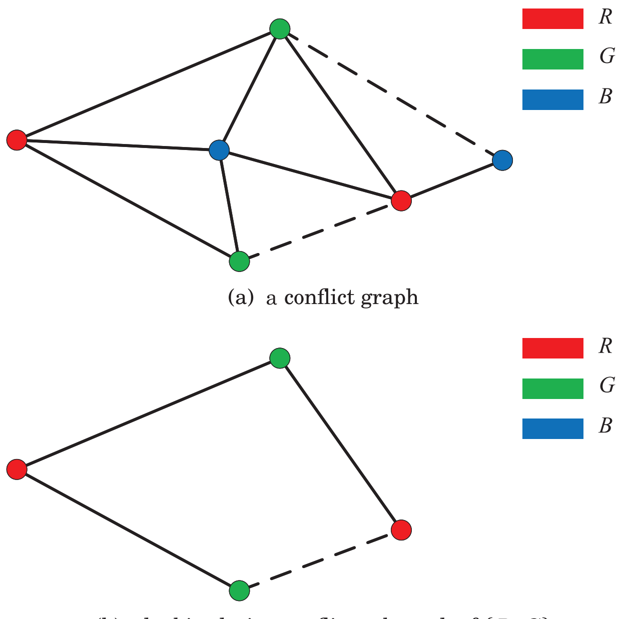


Fig. 6. A conflict graph showing the concepts of pairing colors and bi-coloring conflict subgraph.

(1) whether its runtime performance is competitive with other methods;
(2) when to terminate the current run; and
(3) whether the solution will get stuck in the local minima.

In this section, we endeavor to solve those challenges.

4.2.1. Bi-Coloring Algorithm Design Criteria. In the PWC approach, the overall solution is refined by applying a bi-coloring method whose design criteria are listed as follows.

(1) *Small time complexity*. In order to compete with other methods on the runtime performance, the bi-coloring process should be completed efficiently.
(2) *Compatibility with unresolved conflicts*. For a practical layout, there is no guarantee that all bi-coloring conflict subgraphs are bipartite graphs.

The bi-coloring method integrated into our tool is presented in Section 4.5.

4.2.2. Stopping Criteria. Another issue worth further consideration is the stopping criteria of a run. Assume there exists a "good" bi-coloring approach that can solve the bi-coloring problem optimally. As a result, the cost would not increase after each pass. In that case we can adopt the immediate stopping, where a run stops right after the cost stops decreasing. This criterion was adopted in Ghaida et al. [2012]. However, as mentioned in Section 2, MIN 2-PARTITION is known to be NP-hard for a general graph. Consequently, as shown in Figure 9, the cost after each PWC pass might slightly rise before further drop. Therefore we set the stopping criteria so that: (1) the cost achieves zero; or (2) the solution of this run fails to be updated after t passes successively. With the stopping criteria, in Figure 9, the run stops at C and achieves the solution at B with $t = 3$. For comparison, with the immediate stopping it stops at A.

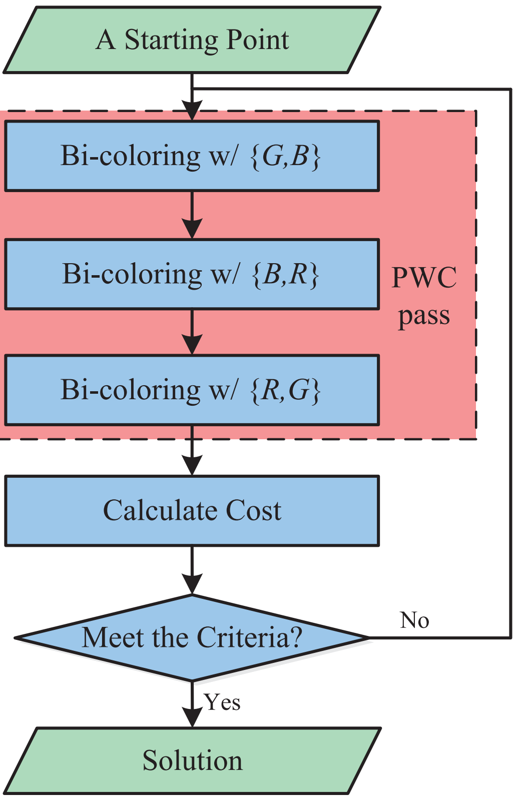


Fig. 7. The flow of the PWC method of a run.

4.3. Adaptive Multi-Start Approach

A tri-coloring problem with one more color to spare possesses a larger solution space than a bi-coloring problem does. Consequently, it is more difficult to find the global optima for TPL. For a local search method such as the PWC approach, it can become localized in a small area of the solution space. Therefore some type of diversification is required to overcome the local optimality. One way to achieve diversification is the multi-start strategy: restart the search from a new solution once a region has been extensively explored [Glover and Kochenberger 2003]. A multi-start procedure usually consists of two phases: (1) the starting points are generated; (2) the solution gets improved. The diversification is achieved in the first step.

The starting points can be constructed with various strategies. Randomization is the simplest way, which was adopted in our previous work [Zhang et al. 2013]. The starting point of each run is purely randomized. It introduces the natural parallelism because all runs are independent from each other. However, it is a memoryless approach where the current decisions derive no benefit from the knowledge accumulated during prior search. Besides, with no control over the diversity, very similar solutions can be reached [Glover and Kochenberger 2003].

In order to overcome the disadvantages of the randomization approach, an AMS approach is adopted in this article. It has been previously exploited and analyzed in detail for combinatorial optimization in Boese et al. [1994]. The authors investigated the global structure of optimization cost surface of the combinatorial optimization problems. They claimed that in the solution space there exists a "big valley" [Boese et al.

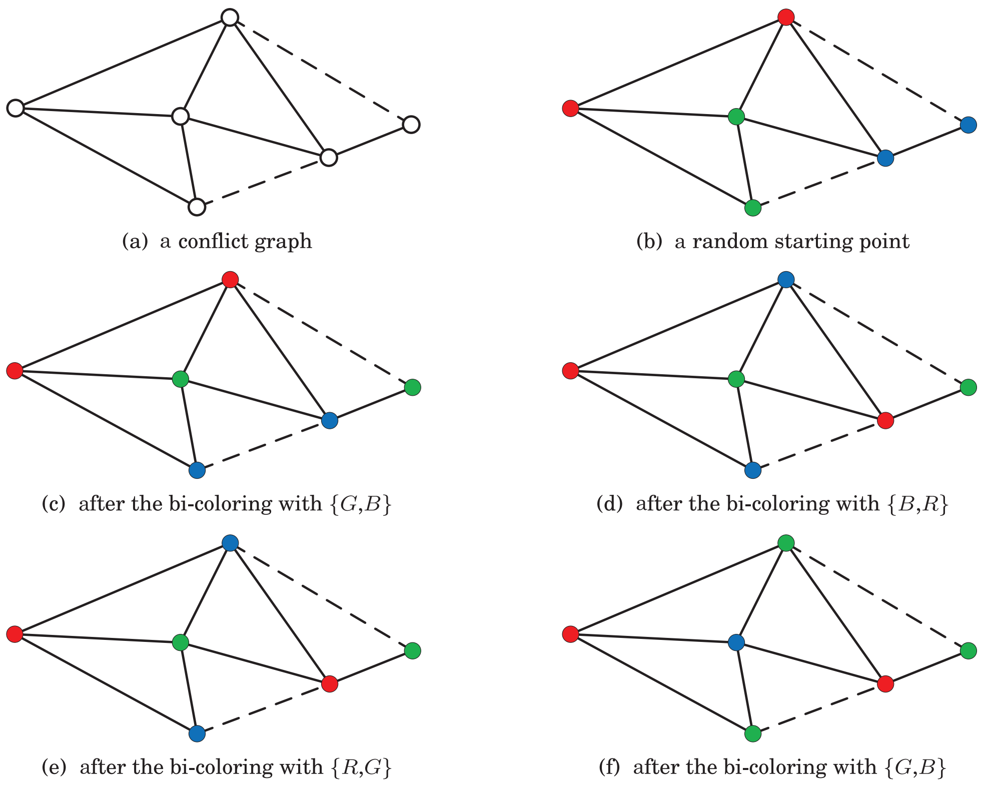

(a) a conflict graph
(b) a random starting point
(c) after the bi-coloring with $\{G,B\}$
(d) after the bi-coloring with $\{B,R\}$
(e) after the bi-coloring with $\{R,G\}$
(f) after the bi-coloring with $\{G,B\}$

Fig. 8. An example of PWC process.

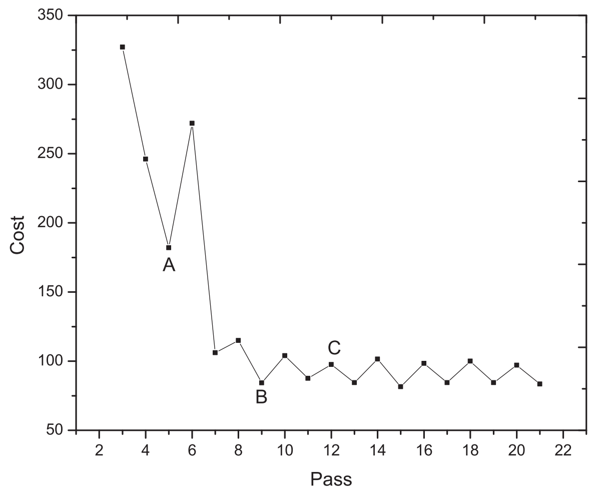


Fig. 9. Cost variation trend of the PWC method on C432 for TPL.

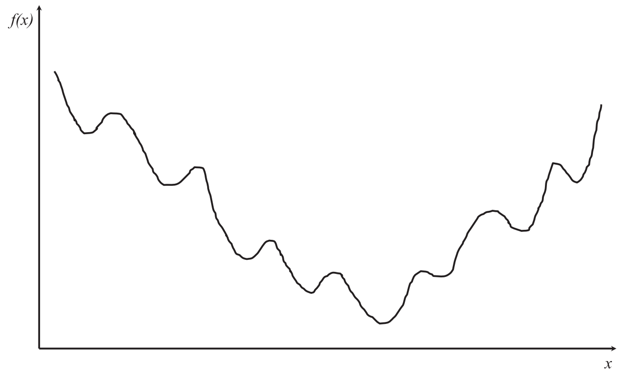


Fig. 10. An intuitive picture of "big valley" [Boese et al. 1993] in the solution space.

1993], an intuitive example of which is shown in Figure 10. The "big valley" implies a correlation: the best local minimum locates central to the others'. It means that the best local minimum is likely to include what other local minima share. This correlation was demonstrated in Boese et al. [1994] by taking *traveling salesman problem* (TSP) and *graph bisection problem* as the examples. The correlation suggests an improved multi-start approach, AMS, which derives starting points from the previous local minima. The derived starting points are more easily led to better local minima. Unlike the randomization approach, the AMS approach constructs the starting points systematically with memory by exploiting the history. The approach was applied and proven both efficient and effective in many applications, such as circuit partition [Hagen and Kahng 1997]. For detailed analysis of the AMS approach, one can refer to Boese et al. [1994].

The AMS approach essentially is a voting approach, which consists of two phases.

(1) *Learning Phase*. Generate a set of random starting points, and the corresponding local minima are achieved by applying the PWC approach on each starting point.
(2) *Improving Phase*. Construct adaptive starting points from the obtained local minima, and run the PWC approach on these to yield corresponding new local minima.

Assume that there are three color sets, R, G, and B. The AMS approach is described in Algorithm 1. A reference vertex v_r is selected randomly in the conflict graph. The "experience", M, stores the latest n_s solutions [Hagen and Kahng 1997]. In the Improving Phase (line 5 to line 18), a single process of improving corresponds to an improving loop (line 6 to line 17). In each loop, from an adaptive starting point, the PWC approach is applied to achieve a new solution S' which then updates M by replacing the earliest solution in M with S'. If S' has lower cost compared with S, the best solution will also be updated. In practice, the solution-quality improvement cannot be guaranteed after an improving loop. Thus, as described in Section 4.2.2, we use the AMS stopping criteria that the improving loop will be exited if: (1) the cost achieves zero; or (2) the best solution fails to be updated after t runs successively. It can save the risk of missing better solutions.

The subroutine *Generate_Adaptive_Starting_Point* constructs an adaptive starting point as shown in Algorithm 2. The n_s solutions in M vote to determine a set of vertices which are more likely to be colored the same as v_r in a better solution. Here we adopt the unanimity rule, where a vertex will be added to the set only if it shares a same color with v_r for all solutions in M. The rule was also applied in Hagen and Kahng

ALGORITHM 1: Adaptive_Multi-Start

```
   Input: The conflict graph $G = (V, E)$, and the number of solutions used to construct each
          adaptive starting point $n_s$.
   Output: The best solution $S$.
 1 $M \leftarrow n_s$ solutions from random starting points;
 2 $S \leftarrow$ best solution in $M$;
 3 $v_r \leftarrow$ a random vertex in $V$;
 4 $u \leftarrow 0$;
 5 while $u \leq t$ do
 6     $S' \leftarrow$ Generate_Adaptive_Starting_Point($V$, $M$, $v_r$);
 7     $S' \leftarrow$ PWC($G$) from $S'$;
 8     if $cost(S') = 0$ then
 9         $S \leftarrow S'$;
10         break;
11     end
12     $M \leftarrow$ update $M$ with $S'$;
13     $u \leftarrow u + 1$;
14     if $cost(S') < cost(S)$ then
15         $S \leftarrow S'$;
16         $u \leftarrow 0$;
17     end
18 end
```

ALGORITHM 2: Generate_Adaptive_Starting_Point

```
   Input: The conflict graph $G = (V, E)$, the previous solutions $M$, the reference vertex $v_r$.
   Output: A new starting point $S'$.
 1 $R \leftarrow \emptyset$;
 2 $G \leftarrow \emptyset$;
 3 $B \leftarrow \emptyset$;
 4 for each vertex $v$ in $V$ do
 5     $flag(v) \leftarrow true$;
 6     for each solution $S$ in $M$ do
 7         if $c(v) \neq c(v_r)$ then
 8             $flag(v) \leftarrow false$;
 9             break;
10         end
11     end
12     if $flag(v) = true$ then
13         $R \leftarrow R \cup \{v\}$;
14     end
15 end
16 $G_c \leftarrow$ the subgraph of $G$ induced by $V - R$;
17 $G, B \leftarrow$ bi-coloring($G_c$);
18 $S' \leftarrow R, G, B$;
```

[1997]. Then the set of vertices are colored red first. Finally the adaptive starting point is constructed after a bi-coloring process with $\{G,B\}$.

4.4. Graph Division and Simplification

As mentioned in Section 2, the tri-coloring problem is NP-hard. The runtime increases dramatically as the problem size grows. In order to reduce the problem size, the graph division and simplification techniques are incorporated.

A connected conflict graph is divided into the biconnected components based on *Depth-First Search* (DFS) in linear time [Tarjan 1972]. On each biconnected component, a graph simplification is performed in a way that the vertices whose degree is less than three are iteratively removed and stored in a stack [Yu et al. 2011, 2013, Fang et al. 2012, 2014, Kuang and Young 2013]. Then the tri-coloring problem on each biconnected component is conquered with the PWC and AMS approaches independently. After that, the vertices in the stack are sequentially added back to the component and assigned a legal color. Finally the coloring solutions of all biconnected components are merged in linear time.

4.5. The Bi-Coloring Method

As mentioned in Section 4.2, the PWC approach reduces the tri-coloring problem to a set of bi-coloring problems on the corresponding bi-coloring conflict subgraphs. The bi-coloring method integrated into our tool is presented in this section. First, the bi-coloring problem is reduced by an SPQR-tree-graph division method. Then a bi-coloring method based on Hadlock's algorithm is applied.

4.5.1. SPQR-Tree-Graph Division Method. The SPQR-tree-graph division method was first introduced for DPL in Luk and Huang [2010]. In order to illustrate the idea, some definitions are as follows.

Definition 4.5 (*Separation Pair*). If $G = (V, E)$ is a biconnected graph, a separation pair of G is a pair of vertices if their removal increases the number of connected components.

Definition 4.6 (*Triconnected Graph*). A graph $G = (V, E)$ is a triconnected graph if there is no separation pair in V.

An SPQR-tree is a data structure in which each node is associated with an undirected triconnected graph G_s with a set of virtual edges. A node, and the graph associated with it, may have one of the following four types.

(1) *S-node*. G_s is a cycle graph.
(2) *P-node*. G_s contains exactly two vertices and at least three edges.
(3) *Q-node*. G_s contains two vertices and two edges. One of the two edges is a virtual edge; the other one is a real edge.
(4) *R-node*. G_s is a triconnected graph other than the three preceding types.

A division of any biconnected graph into its triconnected components can be performed by identifying the separation pairs in linear time with the help of the SPQR-tree [Hopcroft and Tarjan 1973]. According to the prior definition, each real edge corresponds to one Q-node of an SPQR-tree. In actual implementation, the Q-node can simply be replaced with a flag distinguishing a real edge from a virtual one [Gutwenger and Mutzel 2001]. Figure 11 shows an example of a biconnected graph, its triconnected components, and the corresponding SPQR-tree. Note that the SPQR-tree is an unrooted tree. On the other hand, an arbitrary node can practically be chosen as a root.

With the help of SPQR-tree, a bi-coloring problem can be solved with a divide-and-conquer method which consists of three steps.

(1) Divide a biconnected graph into its triconnected components.
(2) Conquer each component in a bottom-up manner.
(3) Merge the solutions into a complete one in a top-down manner.

In step 1, the algorithm in Gutwenger and Mutzel [2001] is implemented to divide the biconnected graph and generate an SPQR-tree. We can choose an arbitrary node

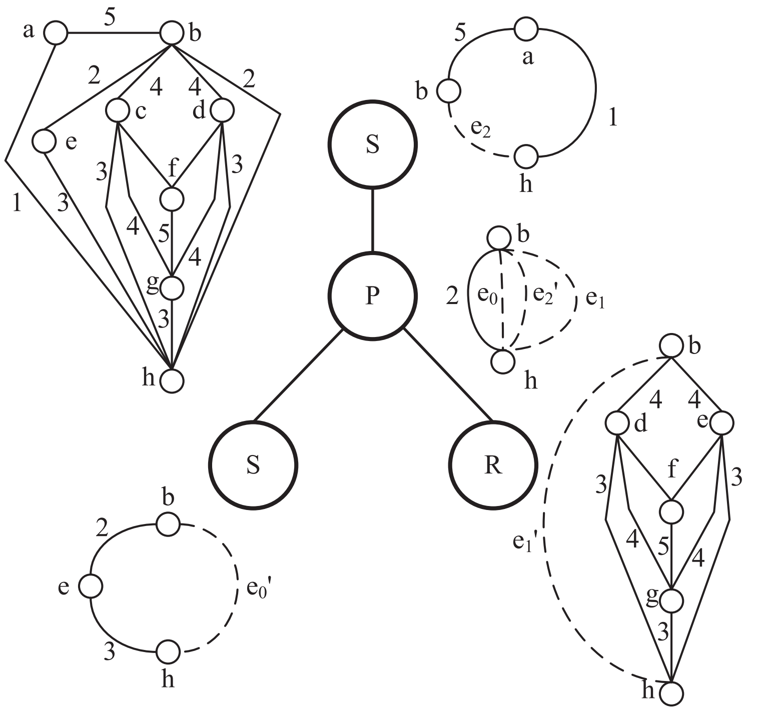


Fig. 11. An example of an SPQR-tree. Virtual edges are indicated by dashed lines.

as a root for convenience. In step 2, the triconnected components are then solved in a bottom-up manner according to the SPQR-tree. First we define the term.

Definition 4.7 (*Reference Edge*). In an SPQR-tree, the reference edge of a node x is a virtual edge of its parent tree node, connecting the separation pair between x and its parent tree node.

Take an example of the R-node shown in Figure 11. The reference edge of the R-node is e_1 and the corresponding separation pair is $\{b, h\}$. There are two possible solutions for the R-node component, namely b and h having a same color, or b and h having different colors. For R-node components, we rely on a bi-coloring method for solving the two solutions. To ensure that b and h are assigned a same color, we assign $-\infty$ to $w(e_1')$. Similarly, to ensure that b and h are assigned different colors, we assign $+\infty$ to $w(e_1')$. The two solutions associate with their cost, C_{00} and C_{01}. The cost difference between the two solutions (e.g., $C_{00} - C_{01}$) is assigned to the weight of its reference edge.

In step 3, all solutions are merged into a complete one in a top-down manner. We simply select and collect the solutions of the tree nodes based on the coloring of the corresponding separation pair. Note that the root component is need only be solved once. Also we can keep track of the solutions by a set of edges E_c for which $G' = (V, E - E_c)$ is a bipartite graph. One advantage is that color flipping can be avoided when merging the solutions. It can be performed in $O(|V| + |E|)$ time.

4.5.2. Bi-Coloring Based on Hadlock's Algorithm. In step 2, in order to achieve E_c for the R-node components, we have implemented a graph-theoretic method described in Chiang et al. [2005] in our tool, which is based on Hadlock's algorithm [Hadlock 1975] for planar graphs. Hadlock's algorithm can find a maximum cut of a planar graph in

ALGORITHM 3: The Bi-coloring Method

```
  Input: An R-node component $G_s = (V_s, E_s)$.
  Output: An edge set $E_c$.
1 Find the maximum planar subgraph $G_p = (V_s, E_s - E_p)$ of $G_s$;
2 Constructs the dual graph $G_0$ of $G_p$;
3 Construct a complete graph $G_m = (V_m, E_m)$;
4 Solve the minimum weighted perfect matching on $G_m$ for $E_{c1}$;
5 Find the edge set $E_{c2} \subseteq E_p$ whose connected vertices are with a same color;
6 $E_c = E_{c1} \cup E_{c2}$;
```

polynomial time. Its main idea is to transform the maximum cut problem on a planar graph into finding a minimum odd-vertex pairing of its dual [Hadlock 1975]. The bi-coloring method based on Hadlock's algorithm is described in Algorithm 3. Other types of nodes can be solved easily and the description is omitted in this article.

For a given R-node component G_s, we first find a planar subgraph of G_s by removing a minimum number of edges (line 1). Note that the maximum planar subgraph problem is NP-hard in general. Here we assume that G_s is a nearly planar graph in practice. The assumption will be justified in Section 5. Finding the maximum planar subgraph is performed as follows: The edges are all deleted first. Then the planar subgraph is obtained by putting back the edges one by one in order of their weight, which can facilitate the post-coloring legalization [Ghaida et al. 2013]. The planarity of the subgraph is checked after adding every edge. Any edge damaging the planarity will be removed. Note that we only need to solve the maximum planar subgraph problem once, even though the bi-coloring problem for each R-type component is needed to be solved twice. Let E_p be the set of removed edges. The resulting planar subgraph $G_p = (V_s, E_s - E_p)$ is then solved optimally by Hadlock's algorithm [Hadlock 1975]. The algorithm eliminates the odd cycles of a planar graph by pairing them in a clever way and removing the edges between every pair. To do that, the algorithm first constructs the dual graph G_0 of G_p in linear time (line 2). Then a complete graph $G_m = (V_m, E_m)$ is formed where V_m is the set of odd-degree vertices (named T-join) of G_0 (line 3). The edge weight in E_m is assigned to be the total length of the shortest path between two vertices in G_0. The *minimum weighted perfect matching* (MWPM) is then applied to G_m, which is polynomial-time solvable using a network flow algorithm (line 4).

In our implementation of Hadlock's algorithm, Dijkstra's method is employed for finding shortest paths when determining the edge weight in E_m. The method is fast yet can only handle nonnegative weights. Luckily, the bi-coloring problem can be transformed into one that contains edges with only positive weights. This can be done by a simple modification of the edges described as follows. An example is shown in Figure 12. For each edge $e = (A, B)$ having a negative weight $-|w|$, we create a dummy node D and replace the edge with (A, D) and (D, B), each having the weight $|w|$. There are only two possible outcomes of the coloring solution, namely A, B having the same color and A, B having different colors. In the original graph, say G, the cost difference is $-|w| - 0 = -|w|$. In the modified graph, say G', if A, B have a same color, D is assigned the alternative color; if A, B have different colors, D can be assigned either color. Hence the cost difference is $0 - |w| = -|w|$, which is the same as that in G. Consequently, the modification does not change any decision of choosing the two possible outcomes and hence the optimality of the result.

Finally, edges in E_p whose connected vertices have different colors are reinserted (line 5).

Note that Hadlock's algorithm cannot be extended to the cases of $k > 2$.

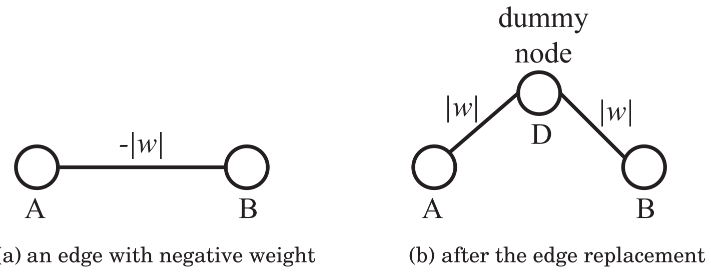

-|w|
A
B
(a) an edge with negative weight
dummy node
|w|
|w|
D
A
B
(b) after the edge replacement

Fig. 12. An edge with negative weight is replaced with two adjacent edges with positive weight and a dummy node.

5. EXPERIMENTAL RESULTS

We have implemented our method in C++, using the LEDA package [Mehlhorn et al. 1997] for the basic graph structure and algorithms. The program was run on a 64-bit Linux machine with a 3.10 GHz CPU and 6GB RAM. ISCAS-85 and 89 benchmarks provided by the authors of Fang et al. [2012, 2014] were tested. The metal-1 layer was used for evaluation, in which the minimum metal-metal spacing and the minimum metal line width was 50nm and 40nm, respectively. Our method was mainly compared against two state-of-the-art works [Yu et al. 2013; Fang et al. 2014]. For the approach in Kuang and Young [2013], a graph library has to be built manually first before decomposition. However, with $d = 160$nm, a conflict graph can be so dense that building the graph library is almost impossible. Therefore it was not used for comparison.

The absolute weight of each edge in a conflict graph can be assigned based on the corresponding overlapping area after rectangle enlargement to represent the importance of the conflict or stitch. However, for the sake of comparison, the weights of conflict and stitch were set to 1 and 0.1, respectively, according to Yu et al. [2013] and Fang et al. [2014]. Note that the values of weights do not affect the effectiveness of the proposed coloring method.

With respect to the setting in Yu et al. [2013] and Fang et al. [2014], the experiments were conducted with $d = 120$nm. First, we evaluate the effectiveness of the AMS approach. Our previous work [Zhang et al. 2013] was used for comparison. Graph division and simplification techniques mentioned in Section 4.4 were not incorporated in either of them. For the method in Zhang et al. [2013], we set $n_L = 30$, $t_L = +\infty$, where n_L is the number of random starting points, and t_L is the time limit. For the AMS approach, we set $n_s = 3$, where n_s is the number of random starting points in the Learning Phase. The results are shown in Table I. "$|V|$" and "$|E|$" denote the number of vertices and number of edges in the conflict graph, respectively. "COST" is the cost of the output. The CPU time is denoted by "CPU (s)". "Imp (%)" and "spdup" denote the cost percentage reduction and speedup times compared with the method in Zhang et al. [2013]. According to Table I, with the help of the AMS approach, COST has been reduced significantly. On the other hand, the runtime is also seriously reduced. The speedup can be up to 8.1x.

Afterwards, we tested our layout decomposition method combining the PWC approach, the AMS approach, and the graph division and simplification techniques. Our method is compared against the RLF-based algorithm in Fang et al. [2014] and the SDP-based method in Yu et al. [2013]. Note that the work in Yu et al. [2013] considered the color balancing as well. We set $n_s = 5$. The rest of the parameters' settings are maintained for our method.

The results are shown in Table II. In the table, "#C", "#S", "COST", and "CPU(s)" denote the number of conflicts, the number of stitches, the cost of the output, and

Table I. Comparison between the PWC Method With and Without AMS, $d = 120\text{nm}$

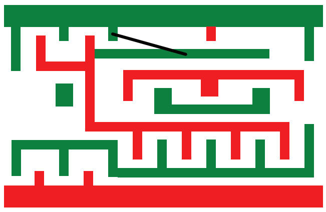
			PWC		PWC+AMS			
Design	$\|V\|$	$\|E\|$	COST	CPU (s)	COST	CPU (s)	Imp (%)	spdup
C432	2883	4787	0.4	14	1.4	2.4	-250	5.8X
C499	5536	9896	0.1	34	0	4.2	100	8.1X
C880	5817	9006	1.3	28	0.7	4.9	46.2	5.5X
C1355	8915	14342	0.9	57	0.5	8.1	44.4	7X
C1908	14127	22371	0.5	84	0.3	14	40	6X
C2670	21160	33988	3.2	131	1.4	26	56.3	5X
C3540	26274	38523	4.4	166	2.4	36	45.5	4.6X
C5315	38523	61472	6.1	294	3.1	47	49.2	6.3X
C6288	37724	57678	38.4	314	33.6	77	12.5	4.1X
C7552	55677	88034	10.8	410	4.2	77	61.1	5.3X
S1488	11129	17369	3	63	1	13	66.7	4.8X
S38417	144501	220465	74.1	1277	47.8	227	35.5	5.6X
S35932	342529	546956	179.8	3270	95.6	963	46.8	3.4X
S38584	355001	538932	192.9	3278	116.9	920	39.4	3.6X
S15850	349210	540952	173.2	3494	99.7	638	42.4	5.5X

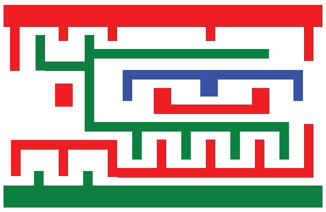
Table II. Experimental Results Comparison with $d = 120\text{nm}$

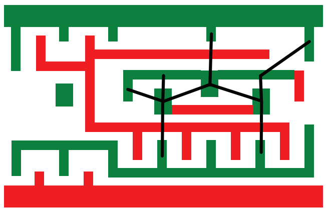
	RLF Based [Fang et al. 2014]				SDP Based [Yu et al. 2013]				Ours			
Design	#C	#S	COST	CPU(s)	#C	#S	COST	CPU(s)	#C	#S	COST	CPU(s)
C432	0	4	0.4	0.01	0	4	**0.4**	0.37	0	4	**0.4**	0.29
C499	0	0	**0**	0.01	0	0	**0**	0.11	0	0	**0**	0.34
C880	1	8	1.8	0.01	0	7	**0.7**	0.10	0	8	0.8	0.31
C1355	1	4	1.4	0.02	0	3	**0.3**	0.29	0	3	**0.3**	0.35
C1908	1	0	1	0.03	0	1	**0.1**	0.13	0	1	**0.1**	0.54
C2670	2	11	3.1	0.05	0	6	**0.6**	0.24	0	6	**0.6**	1.14
C3540	2	12	3.2	0.06	1	9	1.9	0.38	1	8	**1.8**	1.13
C5315	3	11	4.1	0.1	0	9	**0.9**	0.47	0	13	1.3	1.88
C6288	19	248	43.8	0.11	1	212	22.2	4.46	1	205	**21.5**	7.44
C7552	3	37	6.7	0.18	0	26	**2.6**	0.87	1	21	3.1	2.52
S1488	0	4	0.4	0.03	0	2	**0.2**	0.12	0	2	**0.2**	0.42
S38417	43	105	53.5	0.65	30	75	37.5	3.20	27	73	**34.3**	6.24
S35932	108	114	119.4	1.93	77	87	85.7	12.13	77	88	**85.8**	16.33
S38584	120	202	140.2	1.83	83	165	99.5	10.68	83	152	**98.2**	15.43
S15850	85	214	106.4	1.72	57	165	**73.5**	8.64	61	158	76.8	15.94
avg.			32.36	0.45			21.74	2.81			**21.68**	4.69
ratio			1.49	0.095			1.003	0.60			**1.000**	1.000

the CPU time, respectively. Compared between Table I and Table II, the cost and the runtime of our method are reduced further using the graph division and simplification techniques. Compared with the RLF-based algorithm in Fang et al. [2014], our method achieved lower or equal cost for all of the benchmarks. On average the cost has been reduced by 32.9%. Against the SDP-based method in Yu et al. [2013], the solution quality has been improved by 0.3% on average, while the runtime of our method is in a reasonable range. Note that the stitch insertion technique has been incorporated in Yu et al. [2013] and Fang et al. [2014].

However, DPL is preferred for its lower cost. For $d = 120\text{nm}$, DPL can still produce reasonable results. One example is shown in Figure 13(a). The conflict is indicated by the solid line for DPL. The only unresolved conflict can be easily eliminated with the layout legalization technique [Ghaida et al. 2013]. For TPL, all conflicts are resolved


(a) DPL for metal-1 layer with $d = 120$nm


(b) TPL for metal-1 layer with $d = 120$nm


(c) DPL for metal-1 layer with $d = 160$nm

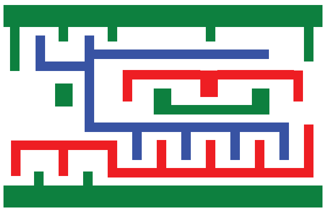

(d) TPL for metal-1 layer with $d = 160$nm

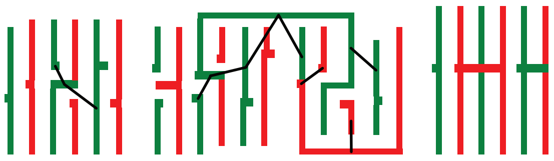
Fig. 13. Multiple patterning for metal-1 layer with different values of d.

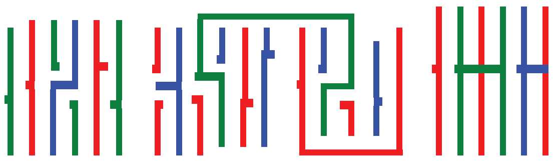
under the same d as shown in Figure 13(b). However, with one additional mask, the cost of TPL is much higher than that of DPL. Therefore TPL will only be adopted at the technology node where DPL fails. With $d = 160$nm, two rectangles might conflict with each other even with another one between them, where there exists an odd cycle which is unsolvable for DPL as shown in Figure 13(c). However, TPL can produce a pleasant solution as shown in Figure 13(d). Figure 14 illustrates a similar example for a poly-layer with different values of d.

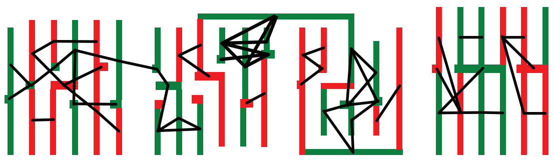
Based on the preceding analysis, we again ran the experiments. The AMS approach was evaluated first with $d = 160$nm. The results are shown in Table III. As d increases, "$|V|$" increases too. The reason is that with a small d there exist isolated vertices whose color can be given randomly. Therefore they can be removed from the conflict graph. With a larger d, "$|V|$" increases for the dropping number of the isolated vertices. In Table III, with the AMS approach, at cost of less random starting points, a relatively equal solution quality is achieved. Especially for the large benchmarks (e.g., S38417, S35932, S38584, S15850), the cost is reduced slightly. On the runtime performance, the speedup is more significant with a larger d, especially for the large benchmarks (e.g., S38417, S35932, S38584, S15850). The effectiveness of the AMS approach is demonstrated with different d.

Next, our method was compared against the RLF-based algorithm in Fang et al. [2014], again with $d = 160$nm. According to Table IV, the RLF-based algorithm achieves the smallest runtime in this experiment. However, its solutions have the lowest quality here. Using our method, COST can be reduced by around 43.7% on average. Then the SDP-based method in Yu et al. [2013] was also compared against our method. Table IV shows that, with our method, COST and the runtime can be reduced by around 15.1% and 42.1% on average.

In Section 4.5.2, we assume that the bi-coloring conflict graph $G_s = (V_s, E_s)$ is a nearly planar graph. Under this assumption, the maximum planar subgraph is found by removing a set of edges D_s in which case Hadlock's algorithm can be then utilized. In order to justify the assumption, the following experiments were conducted. With $d = 120$nm, the results are shown in Table V. In the table, R, G, and B are three


(a) DPL for poly layer with $d = 240$nm


(b) TPL for poly layer with $d = 240$nm


(c) DPL for poly layer with $d = 360$nm

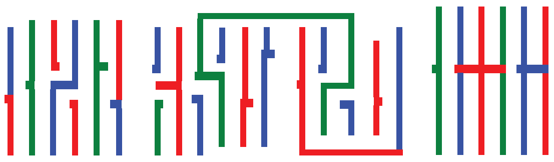

(d) TPL for poly layer with $d = 360$nm

Fig. 14. Multiple patterning for poly-layer with different values of d.

Table III. Comparison between the PWC Method With and Without AMS, $d = 160$nm

			PWC		PWC+AMS			
Design	$\|V\|$	$\|E\|$	COST	CPU (s)	COST	CPU (s)	Imp (%)	spdup
C432	2959	6790	81	58	82.3	8.2	−1.6	7.1X
C499	5615	14040	293.9	176	295.8	25	−0.6	7X
C880	5946	12738	145	142	141.3	17	2.6	8.4X
C1355	9210	19586	143.7	187	145.4	29	−1.2	6.4X
C1908	14393	30095	185.3	307	190.2	44	−2.6	7X
C2670	21341	46199	496.8	628	503.8	105	−1.4	6X
C3540	26549	55573	523.9	763	520.6	122	0.6	6.3X
C5315	38885	84010	973.4	1279	950.6	224	2.3	5.7X
C6288	39427	88219	760.3	1164	762.5	201	−0.3	5.8X
C7552	56438	119346	1164	1708	1147.4	371	1.4	4.6X
S1488	11136	23845	464.5	5919	457.3	77	1.6	4X
S38417	146006	304680	4714	19721	4631.4	828	1.8	7.1X
S35932	349839	771252	13813	19721	13581.7	2978	1.7	6.6X
S38584	360520	753103	11797	18732	11282.8	2765	4.4	6.8X
S15850	353646	757725	14105	14007.1	13878.7	2972	0.9	6.7X

Table IV. Experimental Results Comparison with $d = 160$nm

Design	RLF Based [Fang et al. 2014]				SDP Based [Yu et al. 2013]				Ours			
	#C	#S	COST	CPU(s)	#C	#S	COST	CPU(s)	#C	#S	COST	CPU(s)
C432	94	11	95.1	0.02	86	19	87.9	4.26	78	17	**79.7**	6.81
C499	350	17	351.7	0.06	296	62	302.2	24.83	281	49	**285.9**	14.96
C880	230	36	233.6	0.04	145	124	157.4	5.61	120	94	**129.4**	8.67
C1355	227	50	232	0.05	144	120	156	13.36	124	85	**132.5**	21.77
C1908	287	51	292.1	0.08	184	105	194.5	13.65	158	83	**166.3**	20.97
C2670	809	119	820.9	0.14	471	316	502.6	45.17	432	233	**455.3**	38.26
C3540	810	151	825.1	0.19	525	511	576.1	24.93	410	393	**449.3**	34.63
C5315	1306	197	1325.7	0.26	951	485	999.5	36.52	856	402	**896.2**	49.2
C6288	879	219	900.9	0.24	704	447	748.7	42.27	674	303	**704.3**	69.36
C7552	1585	371	1622.1	0.36	1166	827	1248.7	45.23	974	624	**1036.4**	72.95
S1488	615	107	625.7	0.08	450	274	477.4	11.3	412	224	**434.4**	13.44
S38417	7749	522	7801.2	1.41	4890	2999	5189.9	223.7	4116	2456	**4361.6**	201.9
S35932	23767	1377	23904.7	5.24	14380	8878	15267.8	1543	12306	7321	**13038.1**	755.47
S38584	20106	1377	20243.7	5.4	12287	7907	13077.7	974	10162	6536	**10815.6**	502.4
S15850	22559	1904	22749.4	5.32	14541	8958	15436.8	1065	12504	7200	**13224**	548.7
avg.			5468.3	1.26			3628.2	271.5			**3080.6**	157.3
ratio			1.775	0.008			1.178	1.726			**1.000**	1.000

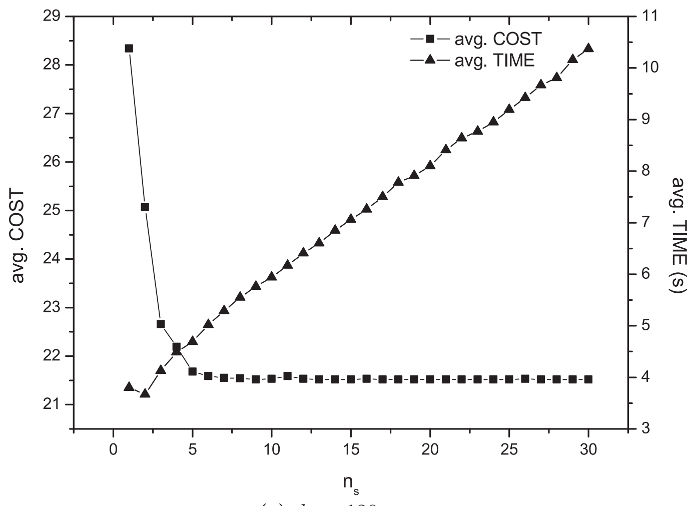
Table V. Justifying that the Bi-Coloring Conflict Subgraph is Nearly Planar with $d = 120$nm

	G_s of $\{R, G\}$			G_s of $\{G, B\}$			G_s of $\{B, R\}$		
Design	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)
C432	2199	8	0.36	2090	11	0.53	2348	12	0.51
C499	4212	15	0.36	4398	7	0.16	4688	6	0.13
C880	3718	11	0.3	4362	41	0.94	4465	36	0.81
C1355	6284	57	0.91	6848	85	1.24	7162	57	0.8
C1908	9985	107	1.07	10476	112	1.07	11185	94	0.84
C2670	15322	160	1.04	15833	120	0.76	16245	150	0.92
C3540	18323	141	0.77	19218	152	0.79	20214	169	0.84
C5315	27830	279	1	28444	332	1.17	29468	290	0.98
C6288	25413	108	0.42	27955	69	0.25	29178	80	0.27
C7552	39274	323	0.82	40928	367	0.9	43023	309	0.72
S1488	7558	81	1.07	8288	78	0.94	8063	61	0.76
S38417	93912	293	0.31	99063	319	0.32	105858	348	0.33
S35932	254796	976	0.38	247197	899	0.36	237433	912	0.38
S38584	231773	515	0.22	241418	445	0.18	258044	471	0.18
S15850	234015	241	0.1	245222	925	0.38	255484	944	0.37
avg.	64974	221	0.34	66783	264	0.39	68858	263	0.38

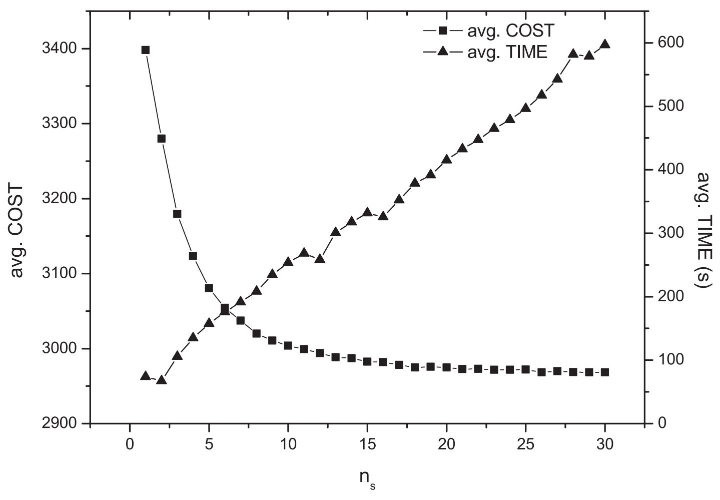
Table VI. Justifying that the Bi-Coloring Conflict Subgraph is Nearly Planar with $d = 160$nm

	G_s of $\{R, G\}$			G_s of $\{G, B\}$			G_s of $\{B, R\}$		
Design	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)	$\lvert E_s \rvert$	$\lvert D_s \rvert$	per (%)
C432	2799	99	3.54	2919	80	2.74	3009	72	2.39
C499	5787	194	3.35	5890	172	2.92	6057	206	3.4
C880	5440	94	1.73	5368	82	1.53	5603	88	1.57
C1355	8415	182	2.14	8696	197	2.27	8464	175	2.07
C1908	12669	214	1.69	13419	277	2.06	13458	276	2.05
C2670	19717	485	2.46	20157	504	2.5	20206	469	2.32
C3540	23697	473	2	23784	449	1.89	24949	481	1.93
C5315	35848	853	2.38	36693	887	2.42	36669	763	2.08
C6288	36411	1047	2.88	38840	987	2.54	38682	1092	2.82
C7552	50641	964	1.9	52131	1027	1.97	52855	969	1.83
S1488	10217	200	1.96	10339	166	1.61	10267	136	1.32
S38417	126486	1891	1.5	129343	2028	1.57	131684	1773	1.35
S35932	320671	5462	1.7	325986	5897	1.81	330223	5486	1.66
S38584	310558	4780	1.54	319364	5117	1.60	326537	4870	1.49
S15850	314970	4820	1.53	323085	5333	1.65	326956	4866	1.49
avg.	85628	1451	1.69	87734	1547	1.76	89041	1448	1.63

available color sets. After a tri-coloring process, G_s is the bi-coloring conflict subgraph of the corresponding pairing colors. There are three bi-coloring conflict subgraphs for TPL. "$\lvert E_s \rvert$" indicates the number of edges of G_s. In order to find the maximum planar subgraph, the number of edges to be removed is denoted as "$\lvert D_s \rvert$". The percentage of "$\lvert D_s \rvert$" over "$\lvert E_s \rvert$" is denoted by "per (%)". According to Table V, the average of "per" is smaller than 0.4%. Therefore the assumption is valid with $d = 120$nm. With $d = 160$nm, the assumption is justified again. The results are shown in Table VI. With a larger d, "$\lvert E_s \rvert$", "$\lvert D_s \rvert$", and "per" all increase. However, the average of "per" is still smaller than 2%. Therefore, the assumption that the bi-coloring conflict subgraph is nearly planar is reasonable with $d = 120$nm or 160nm. Note that a removed edge does not necessarily introduce a conflict or a stitch.



(a) $d = 120\text{nm}$



(b) $d = 160\text{nm}$

Fig. 15. The average cost and the average running time versus n_s with different values of d.

Next, we conducted an experiment to show the trade-off between the solution quality and the runtime. With $d = 120\text{nm}$ and 160nm, Figure 15 shows the graph of "avg. COST" and "avg. TIME" versus n_s, where "avg. COST" and "avg. TIME" indicate the average cost and the average runtime on all ISCAS-85 and 89 benchmarks, respectively. It is obvious that as n_s increases, the average cost decreases while the average runtime increases. Therefore our method possesses the potential to achieve better solutions if more CPU resources are given.

All the previous benchmarks are cell-based structures without *power/ground* (P/G) tracks. To further evaluate our method, we decomposed the metal-1 layer of a more complicated layout of 45nm with P/G tracks, 400k features, and 90% utilization with $d = 160\text{nm}$. Figure 16 shows part of the layout decomposition result. In practice, the P/G tracks are preferred with no stitch, which can be implemented by assigning a large weight to the stitch candidates on the P/G tracks.

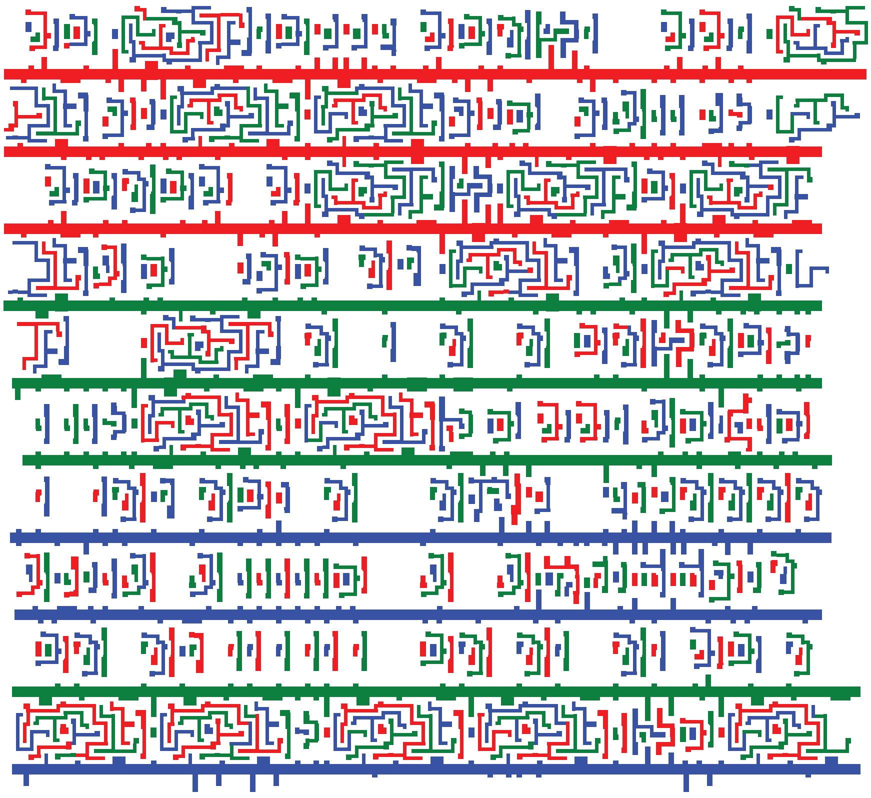


Fig. 16. Layout decomposition result for a portion of a layout with P/G tracks.

6. CONCLUSION

We have presented a layout decomposition method using the pairwise coloring for TPL. It reduces the tri-coloring problem to a set of bi-coloring problems which can be solved with the existing bi-coloring methods. In order to alleviate the intrinsic problems of the pairwise coloring approach, we incorporated an SPQR-tree-based bi-coloring method and an adaptive multi-start approach. By comparing against two state-of-the-art works with different coloring spacing, our method is demonstrated to be competitive both on the solution quality and the runtime performance. Our method is of practical significance because the existing DPL techniques can be reused effortlessly. Consequently, the tool development cost would be remarkably reduced.

REFERENCES

Giorgio Ausiello. 1999. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*. Springer.

Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. 1993. On the big valley and adaptive multi-start for discrete global optimizations. Tech. rep. TR-930, 15. http://ftp.cs.ucla.edu/tech-report/1993-reports/930015.pdf.

Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. 1994. A new adaptive multi-start technique for combinatorial global optimizations. *Oper. Res. Lett.* 16, 2, 101–113.

Zihao Chen, Hailong Yao, and Yici Cai. 2013. SUALD: Spacing uniformity-aware layout decomposition in triple patterning lithography. In *Proceedings of the 14th IEEE International Symposium on Quality Electronic Design (ISQED'13)*. 566–571.

Charles Chiang, Andrew B. Kahng, Subarna Sinha, and Xinnu Xu. 2005. Fast and efficient phase conflict detection and correction in standard-cell layouts. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'05).* 149–156.

Jason Cong and Sung Kyu Lim. 1998. Multiway partitioning with pairwise movement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'98).* 512–516.

Shao-Yun Fang, Yao-Wen Chang, and Wei-Yu Chen. 2012. A novel layout decomposition algorithm for triple patterning lithography. In *Proceedings of the 49th Annual IEEE/ACM Design Automation Conference (DAC'12).* 1185–1190.

Shao-Yun Fang, Yao-Wen Chang, and Wei-Yu Chen. 2014. A novel layout decomposition algorithm for triple patterning lithography. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 33, 3, 397–408.

Rani S. Ghaida, Kanak B. Agarwal, Lars W. Liebmann, Sani R. Nassif, and Puneet Gupta. 2012. A novel methodology for triple/multiple-patterning layout decomposition. *Proc. SPIE 8327*.

Rani S. Ghaida, Kanak B. Agarwal, Sani R. Nassif, Xin Yuan, Lars W. Liebmann, and Puneet Gupta. 2013. Layout decomposition and legalization for double-patterning technology. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 32, 2, 202–215.

Fred Glover and Gary A. Kochenberger. 2003. *Handbook of Metaheuristics*. Springer.

Kevin D. Gourley and Douglas M. Green. 1983. Polygon-to-rectangle conversion algorithm. *IEEE Comput. Graph. Appl.* 3, 1, 31–32.

Carsten Gutwenger and Petra Mutzel. 2001. A linear time implementation of SPQR-trees. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'01).* 77–90.

F. Hadlock. 1975. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.* 4, 3, 221–225.

Lars W. Hagen and Andrew B. Kahng. 1997. Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 16, 7, 709–717.

John E. Hopcroft and Robert E. Tarjan. 1973. Dividing a graph into triconnected components. *SIAM J. Comput.* 2, 3, 135–158.

Andrew B. Kahng, Chul-Hong Park, Xu Xu, and Hailong Yao. 2008. Layout decomposition for double patterning lithography. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08).* 465–472.

Jian Kuang and Evangeline F. Y. Young. 2013. An efficient layout decomposition approach for triple patterning lithography. In *Proceedings of the 50th Annual IEEE/ACM Design Automation Conference (ICCAD'13).* 69–74.

Wai-Shing Luk and Huiping Huang. 2010. Fast and lossless graph division method for layout decomposition using SPQR-tree. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'10).* 112–115.

Edward M. Mccreight. 1985. Priority search trees. *SIAM J. Comput.* 14, 2, 257–276.

Kurt Mehlhorn, Stefan Naher, and Christian Uhrig. 1997. The LEDA platform for combinatorial and geometric computing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICLAP'97).* 7–16.

Jian Sun, Yinghai Lu, Hai Zhou, and Xuan Zeng. 2011. Post-routing layer assignment for double patterning. In *Proceedings of the 16th IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'11).* 793–798.

Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1, 2, 146–160.

Haitong Tian, Yuelin Du, Hongbo Zhang, Zigang Xiao, and Martin D. F. Wong. 2013. Constrained pattern assignment for standard cell based triple patterning lithography. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13).* 178–185.

Haitong Tian, Hongbo Zhang, Qiang Ma, Zigang Xiao, and Martin D. F. Wong. 2012. A polynomial time triple patterning algorithm for cell based row-structure layout. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'12).* 57–64.

Yue Xu and Chris Chu. 2010. A matching based decomposer for double patterning lithography. In *Proceedings of the 19th International Symposium on Physical Design (ISPD'10).* 121–126.

Jae-Seok Yang, Katrina Lu, Minsik Cho, Kun Yuan, and David Z. Pan. 2010. A new graph-theoretic, multiobjective layout decomposition framework for double patterning lithography. In *Proceedings of the 15th IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC'10).* 637–644.

Bei Yu, Yen-Hung Lin, Luk-Pat Gerard, Duo Ding, Kevin Lucas, and David Z. Pan. 2013. A high-performance triple patterning layout decomposer with balanced density. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13).* 163–169.

Bei Yu and David Z. Pan. 2014. Layout decomposition for quadruple patterning lithography and beyond. In *Proceedings of the 51st Annual IEEE/ACM Design Automation Conference (DAC'14)*. 1–6.

Bei Yu, Kun Yuan, Boyang Zhang, Duo Ding, and David Z. Pan. 2011. Layout decomposition for triple patterning lithography. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'11)*. 1–8.

Kun Yuan, Jae-Seok Yang, and David Z. Pan. 2010. Double patterning layout decomposition for simultaneous conflict and stitch minimization. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 29, 2, 185–196.

Ye Zhang, Wai-Shing Luk, Hai Zhou, Changhao Yan, and Xuan Zeng. 2013. Layout decomposition with pairwise coloring for multiple patterning lithography. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 170–177.

Wei Zhao, Hailong Yao, Yici Cai, Subarna Sinha, and Charles Chiang. 2014. Fast and scalable parallel layout decomposition in double patterning lithography. *Integr. VLSI J.* 47, 2, 175–183.

Received December 2014; revised March 2015; accepted April 2015