This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

OpenMPL: An Open Source Layout Decomposer

Wei Li, Yuzhe Ma, Qi Sun, Lu Zhang, Yibo Lin, *Member, IEEE*, Iris Hui-Ru Jiang, *Senior Member, IEEE*, Bei Yu, *Member, IEEE*, David Z. Pan, *Fellow, IEEE*

***Abstract*—Multiple patterning lithography has been widely adopted in advanced technology nodes of VLSI manufacturing. As a key step in the design flow, multiple patterning layout decomposition (MPLD) is critical to design closure. Due to the $\mathcal{NP}$-hardness of the general decomposition problem, various efficient algorithms have been proposed with high-quality solutions. However, with increasingly complicated design flow and peripheral processing steps, developing a high-quality layout decomposer becomes more and more difficult, slowing down further advancement in this field. This paper presents OpenMPL [1], an open-source layout decomposition framework, with well-separated peripheral processing and core solving steps. Besides, previous algorithms or techniques are inspected and several issues are discovered. We then propose corresponding new algorithms to resolve these issues. The experiments demonstrate the effectiveness of our proposed algorithms and the efficiency of OpenMPL.**

I. Introduction

MULTIPLE patterning layout decomposition (MPLD) has been adopted to enhance the lithography resolution. The key idea of MPLD is to assign features that are close to each other to different masks, such that these features are far away enough to be printed with existing lithography techniques. MPLD can be divided into double patterning layout decomposition (DPLD), triple patterning layout decomposition (TPLD) and quadruple patterning layout decomposition (QPLD), according to the number of masks. This problem is difficult since it is a variation of the graph coloring problem, which is $\mathcal{NP}$-hard for $k \geq 3$, where k is the number of colors (masks).

Fig. 1 is an example of TPLD, where different colors represent different masks. Unlike the classical graph coloring problem, the MPLD problem has several unique characteristics. 1) Stitch: a polygon feature is allowed to be split into multiple overlapping segments to resolve coloring conflicts, as shown by the dashed edge in Fig. 1(c). 2) Special patterns: there are different kinds of special features in a circuit layout, e.g., alternative power and ground lines, which may help to simplify the graph. 3) Complex rules: besides the widely adopted spacing constraint for the same color, there are also other rules. The different color spacing constraints [2] are related to the ordering of masks. That is, these constraints predetermine the colors of some features before decomposition. All above characteristics impose different challenges to the MPLD problem, thus specialized algorithms are in demand to solve the MPLD problem effectively and efficiently.

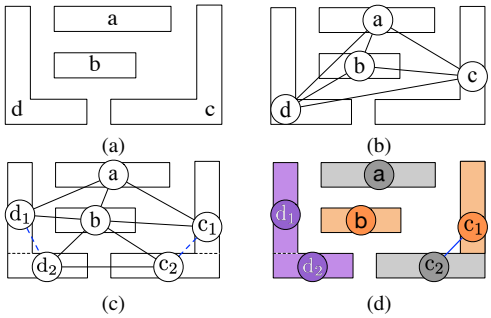


Fig. 1 An example of TPLD with stitches. (a) The input features. (b) The constructed layout graph without stitch candidate generation, which is a 4-clique and therefore not 3-colorable; (c) The constructed layout graph with stitch candidate generation. Two stitch candidates are introduced and the original 4-clique is dismissed; (d) Coloring on the layout graph with stitch candidate generation. The final decomposed layout with three masks (each color corresponds to one mask). The stitch candidates are highlighted in blue.

To achieve high efficiency and to maintain high solution quality, a variety of decomposition algorithms have been proposed. These algorithms can be roughly categorized into three types [3], [4]: mathematical programming and relaxation, graph-theoretical approaches, and search-based approaches. Mathematical programming solves the MPLD problem by formulating it into a standard optimization model, such as integer linear programming (ILP) for DPLD [5]–[7] and TPLD [8]–[10]. Due to the $\mathcal{NP}$-hardness of TPLD and QPLD, a set of relaxation techniques such as semidefinite programming (SDP) [8], linear programming (LP) [11], and discrete relaxation method [12] are proposed based on ILP. Another category is to directly perform color assignment based on a set of graph-theoretical algorithms, e.g., the maximal independent set (MIS) [13], the shortest-path [14], [15], and fixed-parameter tractable (FPT) algorithms [16]. Search-based algorithms follow a divide-and-conquer principle with each sub-graph containing a small number of nodes, e.g., less than 20. Then a search procedure is applied to find the optimal solutions for small sub-graphs [8], [13], [17]–[20]. Besides the researches on the single layout decomposition stage, recent

The preliminary version was invited to be presented at the IEEE International Conference on ASIC (ASICON) in 2019. This work is supported in part by Cadence Design Systems, The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017), Taiwan MOST under Grant No. 106-2628-E-002-019-MY3, and US NSF under award No. 1718570. (*Corresponding author: Bei Yu.*)

W. Li, Y. Ma, Q. Sun, L. Zhang and B. Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong.

Y. Lin is with the Center for Energy-Efficient Computing and Applications, School of Electronics Engineering and Computer Science, Peking University.

I. H. Jiang is with the Department of Electrical Engineering, National Taiwan University.

D. Z. Pan is with the Department of Electrical and Computer Engineering, University of Texas at Austin.

work [21], [22] pioneers a new direction that integrates layout decomposition and mask optimization seamlessly, achieving compelling results from a global view of the solution space.

No matter how efficient the decomposition algorithm is, the $\mathcal{NP}$-hardness of TPLD and QPLD still makes the problem suffer from the runtime issue, especially when the graph size is large. Therefore, many graph simplification techniques have been developed to reduce problem size. The representative techniques include independent component computation (ICC) [8], iterative vertex removal (IVR) [8], [17], biconnected component extraction (BCE) [6], [7] and sub-K4 structure merging for TPLD [11].

To reduce the repeated effort in the reimplementation of the whole decomposition framework and lower the bar of research on MPLD, we present OpenMPL as an open platform for developing MPLD algorithms. OpenMPL contains efficient implementations of widely adopted graph simplification techniques and state-of-the-art layout decomposition algorithms. We carefully design the software architectures and APIs to decouple the innovations on the core optimization steps. For example, one can focus on developing novel graph simplification or decomposition techniques without worrying about the peripheral processing issues as the platform provides clean and well-defined APIs for the kernel optimization engines.

Moreover, considering that the framework is well decoupled, which makes each step separated clearly, we can inspect individual algorithm or technique easily. Through the inspections, a set of issues are discovered and corresponding solutions to these issues are proposed in OpenMPL. Specifically, there are three possible issues which can be further improved: (1) There exist some redundant stitches which can be removed without decomposition quality loss; (2) The original problem formulation and corresponding ILP method cannot quantify the cost accurately, which makes the previous ILP-based algorithm sub-optimal; (3) The original exact cover (EC)-based algorithm fails to obtain the optimal solution in some cases. All these issues are well described and solved in this paper. Our contributions are highlighted as follows:

- We present OpenMPL [1], an open-source layout decomposition framework, with efficient implementations of various state-of-the-art simplification and decomposition algorithms.
- We prove the stitch candidate redundancy in the state-of-the-art stitch generation algorithm and propose a corresponding solution.
- We find the sub-optimality in the widely-adopted ILP formulation and propose an optimized ILP-based algorithm with improved performance.
- We improve the exact cover (EC)-based algorithm by some techniques which were not revealed and studied in the previous work.
- We conduct experiments on widely-recognized benchmarks and new large-scale designs derived from the latest ISPD'19 benchmark suites. The results demonstrate the effectiveness of our proposed algorithms and techniques.

The rest of this paper is organized as follows. Section II gives the problem formulation and discusses the design principles, the workflow, and some other properties of OpenMPL. Section III discusses the redundancy of the stitch candidate and gives the corresponding stitch redundancy removal algorithm. Section IV provides the non-optimal cases generated by the previous ILP-based algorithm and the updated optimized ILP-based algorithm is proposed. Section V introduces the drawbacks of the previous EC-based algorithm in some cases and proposes the optimized EC-based algorithm. Section VI lists comprehensive experimental results, followed by conclusion and future work in Section VII.

II. The OpenMPL Framework

In this section, we first formulate the MPLD problem, which is the target of OpenMPL. Then, we introduce OpenMPL by covering the design principles, workflows, and functionalities. Finally, some additional features of OpenMPL are discussed.

A. Problem Formulation

The general MPLD problem can be formulated as follows:

Problem 1 (MPLD)**.** *Given 1) a routed layout which is a set of polygonal features; 2) the number of masks k; 3) the minimal conflict space d; 4) other constraints like pre-coloring constraints, the goal is to assign one or more masks (if the stitch is enabled) to each feature so that the weighted sum of conflict cost and stitch cost is minimized.*

B. Design Principles

OpenMPL is designed for end-users, developers, and researchers as a general platform for the MPLD algorithms. Therefore, we emphasize usability, efficiency, and extensibility during development. The core design principles are highlighted as follows. (1) **Decoupled design stages**. The implementation clearly separates different optimization stages, as shown in Fig. 2. Therefore, the interdependence between them is minimized. In this way, developers can focus on verifying individual stages without worrying about cross-stage impacts. (2) **Graph representations throughout the core stages**. After layout graph construction, the graph simplification, decomposition solver, and the simplified graph recovery stages use pure graphs as input/output, without involving mask data. This design leads to well-defined and highly separable core algorithms, making the framework highly extensible. (3) **Efficiency and generality for different mask data**. As a mask layer can be a contact layer or a metal layer, the processing efficiency varies significantly for different types of layers. We design a general mask database with separate processing routines for contact layers and metal polygon layers for efficiency enabled by C++ polymorphism since contact layers can be processed in a much simpler way.

C. Workflow and Functionalities

The workflow of OpenMPL is illustrated in Fig. 2. Firstly, one chip layout information (in GDS format) file is loaded and transformed into a layout graph (LG), which is represented by a vector of rectangle pointers, where the rectangles are

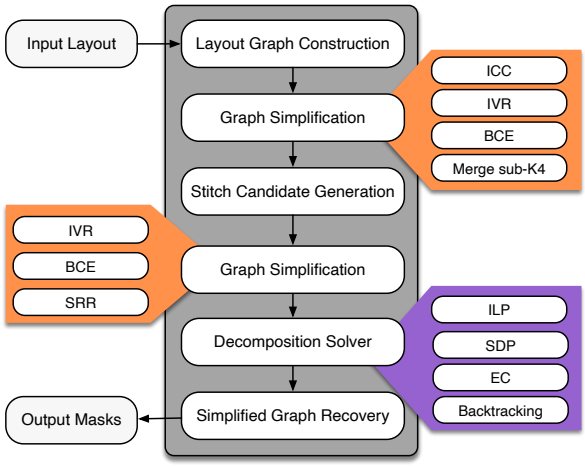


Fig. 2 The workflow of OpenMPL.

defined in Boost. Secondly, LG is simplified by some optional graph simplification techniques, where some of them are implemented in a third-party library Limbo [23]. Then, if stitch is enabled, the stitch insertion process [8] is executed to generate a decomposed graph (DG) with stitches. DG is further simplified by several simplification techniques. After the simplification, a coloring solver is called for each component in DG to solve the component coloring problem. Finally, our framework recovers nodes removed in the simplification step and assigns legal color for each removed node. In the following sub-sections, we are going to introduce all of the functionalities in two crucial procedures of OpenMPL: graph simplification and decomposition.

Graph simplification techniques can be used to reduce the graph size and therefore reduce the computational complexity. Through layout graph simplification, we only need to deal with the smaller graph without affecting the final result. All of the simplification techniques mentioned in Section I are supported in our framework, including independent component computation (`ICC`), iterative vertex removal (`IVR`), biconnected component extraction (`BCE`), and sub-K4 structure merging for TPLD (`Merge sub-K4`). `ICC` is proposed based on the fact that there are many isolated clusters in a real layout, which enables `ICC` to break down the layout graph into several independent components. `IVR` temporarily removes the nodes whose degree is less than the number of colors in an iterative manner. `BCE` simplifies the graph by duplicating the bridge vertices and then removing the bridge edges. `Merge sub-K4` detects and merges specific structures whose number of edges is exactly one less than four-clique structures and thus is only applicable for TPLD. Except `Merge sub-K4`, other implemented simplification techniques support any number of masks. Besides these simplification methods, we develop a simplification method which focuses on the removal of redundant stitches. The details are shown in Section III. Different simplification techniques require different recovery methods. However, those nodes which are shared among different components may be assigned different colors after recovery. To tackle this, color rotation [6] is implemented in our framework. Specifically, color rotation is to rotate the color assignments of the sub-graphs to avoid unnecessary conflict when coloring the whole layout graph from the sub-graphs.

Graph color assignment is the most crucial step in the flow, which impacts the final coloring results directly. In the graph color assignment, a simplified graph is provided and each vertex in the graph should be assigned one color by the specified algorithm. OpenMPL has supported all of the commonly-used algorithms in the layout decomposition and some updated algorithms are also implemented. The algorithms are briefly introduced in the following context:

- **Original Integer Linear Programming**: The details are covered in Section IV-A. We use Gurobi [24], Lemon [25], and CBC [26] as the ILP solvers.
- **Optimized Integer Linear Programming**: The details are covered in Section IV-B.
- **Semidefinite Programming**: The discrete integer programming solving process of Equation (9) is $\mathcal{NP}$-hard, thus it may suffer from run-time overhead for practical designs. As shown in [8], [27], [28], the color assignment can be formulated as a vector programming and then relaxed and solved by semidefinite programming in polynomial time. Given the solutions of SDP, a mapping process is used to map the solutions to coloring results. CSDP [29] is used as the SDP solver.
- **Backtracking**: Backtracking [8] is a DFS fashion algorithm used to find solutions in the whole solution space. Especially, we use a simple but effective heuristic technique to speed up the backtracking process. We set the upper bound of the cost as 0 at the beginning to cut branches more frequently and thus speed up the process. If no feasible solution is found under such an upper bound constraint, we relax the constraint by adding the bound to 1 and repeat the procedure until finding the optimal solution.
- **Original Exact cover-based algorithm**: The details are covered in Section V-A. We implement the dancing links data structure and EC solver, instead of calling the third-party solver like ILP and SDP. Therefore, the runtime of the EC-based algorithm can be optimized further compared to other algorithms.
- **Flexible Exact cover-based algorithm**: The details are covered in Section V-B.

OpenMPL also supports decomposition algorithms like maximal independent set (MIS) [13], linear programming (LP) [11], etc., which cannot decompose the graph containing stitch edges while working well on stitch-free graphs. Due to the page limit, we leave the details on the tool release page [1].

D. Additional Features

Some additional features are supported for better usability, efficiency and extensibility. 1) OpenMPL supports **multi-threading** operations by OpenMP [30] and users can specify the number of threads. Graph components are solved in parallel and layout decomposition algorithms also support multi-threading computations; 2) We can identify all the possible positions of stitches through pattern projections [8]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

in **stitch insertion**, which is one of the most critical steps to parse a layout. One example of the stitch is shown in Fig. 1. There are lots of candidate positions to insert a stitch, and only some are chosen as the final stitches. 3) In practice, a pattern in the layout may be a polygon or rectangle. Consequently, the storage may vary from case to case. OpenMPL provides a **shape-friendly** system considering this case and users can specify the shape, `POLYGON` or `RECTANGLE`, to guarantee the performance to avoid unnecessary calculations. For polygonal inputs, to simplify the storage structure design and save space, OpenMPL first decomposes the polygons to rectangles. After reading the whole input file, DFS is utilized to find connected components and re-union rectangles into polygons. For rectangle circuits, we directly store these patterns without further operations.

III. Stitch Redundancy Removal

In this section, we briefly introduce the widely used stitch candidate generation method and then propose an algorithm for stitch redundancy removal (SRR) with mathematical proof.

A. Stitch Candidate Generation

The original layout does not contain stitch information, thus the framework for MPLD problem should determine the positions to insert stitches. One example of stitch can be found in Fig. 1(c), where c_1-c_2 and d_1-d_2 are two generated stitch candidates. Previous works proposed solutions to generate candidate stitches for DPL [5], [6] and TPL [9], [17]. The key idea of stitch candidate generation is to project each feature into its neighbor features, where the projection results are then used to determine stitch candidates. For example, [17] proposed a heuristic algorithm to find all legal stitch positions in TPL using the projection results. Kahng *et al.* [6] used the projection sequence to directly carry out stitch candidate generation by some simple rules. One example of the stitch candidate insertion by projection sequence is shown in Fig. 3, where the middle feature a has three conflict features, b, c,and d. Based on the projection indicated by the black dash line in Fig. 3, the feature a is divided into 7 segments. Each segment is labeled by the number of projected conflict features, then we can get its projection sequence: 01212101010. The rules of the projection sequence are different when the number of masks varies. The general rules of the projection sequence for TPLD can be summarized as follows [9]: If 1) the projection sequence contains sub-sequences whose value xyz satisfies $x > y, z > y$; 2) the sub-sequence is not at the beginning or end of the projection sequence with form 01010, then the middle positions of y should insert one stitch candidate. As shown in Fig. 3, the middle feature a has three conflict features, b, c, d. According to the rules stated above, one stitch candidate is inserted into a as shown in the figure. In our implementation, such a stitch candidate generation approach supports any number of masks and therefore can be used for general MPL. However, when the mask number is larger than three, the stitch candidates may be redundant or missed since we haven't considered special properties for larger mask numbers.

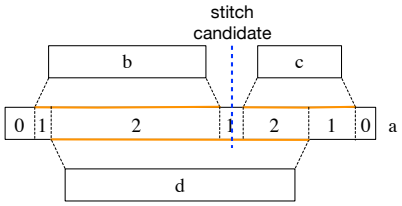


Fig. 3 Projection results, where the projection sequence is 0121210 and the middle segment whose label is "1" should be inserted a stitch for TPLD, which is highlighted by blue.

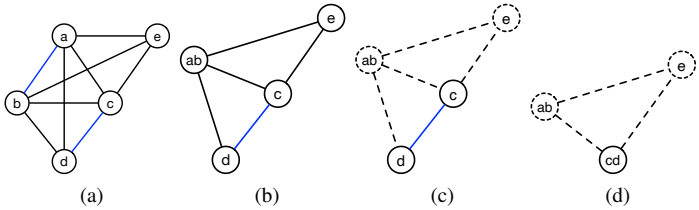


Fig. 4 An TPLD example of stitch redundancy removal (SRR). The conflict edge is marked with black and the stitch edge is blue. Dotted edges/nodes are removed. (a) The decomposed graph before SRR. (b) Nodes a and b are merged into node ab. (c) Node ab and node e are further removed by IVR. (d) Node c and node d are merged into node cd.

B. Stitch Redundancy Removal

Although the current stitch candidate generation algorithm is able to find all possible stitches [8], [17], there are a few stitch candidates that are redundant after further graph simplification. One example is shown in Fig. 4(a), where the edge a-b is redundant, i.e., a, b can be assigned with the same color without additional cost. To clarify the phenomenon, we define $C(u)$ as the cost of node u and compute as:

$$C(u) = \sum_{i \in N^c(u)} c(i,u) + \alpha \sum_{i \in N^s(u)} s(i,u), \tag{1a}$$

$$\text{s.t.} \quad c(i,u) = \min\{\sum_{r_j \in p_i} (x_j == x_u), 1\}, \forall p_i \in N^c(u), \tag{1b}$$

$$s(i,u) = (x_i \neq x_u), \forall r_i \in N^s(u), \tag{1c}$$

$$x_i, x_u \in \{1, \ldots, k\}, \tag{1d}$$

where $N^c(u)$ is the neighbor feature set of u connected by conflict edges, $N^s(u)$ is the neighbor node set of u connected by stitch edges, the node/feature is defined by r/p respectively and x_i is a variable for the k available colors of the node r_i. $x_j == x_u$ represents 1 if x_i equals to x_u and 0 if they are inequivalent. $x_i \neq x_u$ is defined in an opposite way. Take Fig. 4(a) as an example, for the node a, $N^c(a) = \{cd, e\}, N^s(a) = \{b\}$, where cd represents the original feature divided by the stitch edge c-d.

Given a coloring solution $f : V \rightarrow \{1, ..., k\}$, where $\{1, ..., k\}$ is the index set of the k colors. $C_f(u)$ is the cost of node u when u is colored by f and computed by:

$$C_f(u) = \sum_{i \in N^c(u)} c_f(i,u) + \alpha \sum_{i \in N^s(u)} s_f(i,u), \tag{2}$$

where $c_f(i,u)$ and $s_f(i,u)$ are defined similarly to $c(i,u)$ and $s(i,u)$ in Equation (1). The color x_u for node u when calculating $C_f(u)$ is given by f, i.e., $x_u = f(u)$. We have the following theorem about stitch redundancy:

Theorem 1. *Given a decomposed graph G, if there exists a stitch edge $e_s = \{u,v\}$ and the node pair $\{u,v\}$ satisfies three constraints:*

1) $N^c(u) = N^c(v)$;
2) $|N^s(u) \backslash v| \leq 1$;
3) $|N^s(v) \backslash u| \leq 1$,

then at least one optimal coloring solution will assign the two nodes with the same color.

Proof. The proof can be finished by contradiction. Assume that all of the optimal coloring solutions assign u, v into two different colors. Let f^* be one of the optimal coloring solutions and we have $f^*(u) \neq f^*(v)$. We will show that there is another coloring solution f', which assigns u, v into the same color and has at least the same cost with f^* and thus makes f' be the optimal coloring solution. Without loss of generality, we assume:

$$\sum_{i \in N^c(u)} c_{f^*}(i,u) \leq \sum_{i \in N^c(v)} c_{f^*}(i,v), \tag{3}$$

Then f' is defined as follows:

$$f'(i) = \begin{cases} f^*(u), & \text{if } i = v; \\ f^*(i), & \text{otherwise.} \end{cases} \tag{4}$$

Since the only difference between f^* and f' is the color of v, the cost difference $\triangle$ between f^* and f' on G is given by:

$$\triangle = C_{f^*}(v) - C_{f'}(v). \tag{5}$$

By Equation (2) and Equation (5), $\triangle$ can be further interpreted as:

$$\begin{aligned} \triangle = & \Big(\sum_{i \in N^c(v)} c_{f^*}(i,v) - \sum_{i \in N^c(v)} c_{f'}(i,v) \Big) \\ & + \alpha \Big(\sum_{i \in N^s(v)} s_{f^*}(i,v) - \sum_{i \in N^s(v)} s_{f'}(i,v) \Big). \end{aligned} \tag{6}$$

For the first conflict term, combining the first constraint, Equation (3) and Equation (4), we have:

$$\sum_{i \in N^c(v)} c_{f^*}(i,v) \geq \sum_{i \in N^c(v)} c_{f'}(i,v). \tag{7}$$

For the second stitch term, the third constraint $|N^s(v) \backslash u| \leq 1$ indicates that: $\sum_{i \in N^s(v) \backslash u} s_{f^*}(i,v) \leq 1$ and $\sum_{i \in N^s(v) \backslash u} s_{f'}(i,v) \leq 1$. Moreover, we have $s_{f^*}(u,v) = 1 > s_{f'}(u,v) = 0$ since the colors of u, v by f^* are different. Therefore, we have:

$$\sum_{i \in N^c(v)} s_{f^*}(i,v) \geq 1 \geq \sum_{i \in N^c(v)} s_{f'}(i,v). \tag{8}$$

Combining Equation (6), Equation (7) and Equation (8), it is clear to see that $\triangle \geq 0$ always holds, which means that we can color G by f' without additional cost compared with the optimal solution f^* and thus complete the proof. □

Algorithm 1 StitchRedundancyRemoval

```
Input: S → Decomposed graph set.
 1: for DG ∈ S do
 2:     NeedSimplification ← False;
 3:     for s_{i,j} ∈ DG do
 4:         if {i, j} satisfies constraints in theorem 1 then
 5:             DG' ← Merge i, j in DG;
 6:             NeedSimplification ← True;
 7:         end if
 8:     end for
 9:     if NeedSimplification then
10:         S' ← Simplified sub-graph set by simplifying DG';
11:         StitchRedundancyRemoval(S');
12:     end if
13: end for
```

According to the theorem, we can conclude that all stitch edges satisfying constraints specified in Theorem 1 are redundant and corresponding node pairs can be merged to further simplify the graph. Motivated by this conclusion, we propose Algorithm 1 to remove redundant stitch candidates. The algorithm is simply described as follows: after the stitch insertion and the graph simplification, the layout is divided and simplified into a decomposed graph set S. For each decomposed graph (DG) in S, the algorithm detects all stitch edges which satisfy the constraints specified in the theorem 1 (line 4) and merges all valid stitch edges (line 5). If DG can be further simplified (line 10) after the removal of redundant stitch edges, the simplified graph set (S') can be processed again (line 11) by Algorithm 1. One simple TPLD example is given in Fig. 4. As shown in the example, the stitch edge a-b is redundant and thus the node pair $\{a,b\}$ is merged (Fig. 4(b)). After the removal of a-b, the graph can be further simplified by IVR (Fig. 4(c)). Then, the stitch edge c-d in the simplified graph is also redundant, and thus the node pair $\{c,d\}$ is merged (Fig. 4(d)).

IV. Optimized ILP-based Algorithm

In this section, we first introduce the previous cost formulation and corresponding ILP-based algorithm proposed in [8], then the non-optimal case of such formulation is provided and discussed, followed by a new cost formulation and the corresponding optimized ILP-based algorithm proposed by us.

A. Original ILP-based Algorithm

Given an input layout specified by features in polygonal shapes, the layout can be translated into an undirected layout graph $G = (V, E)$, where every node $v_i \in V$ corresponds to one feature/sub-feature in the layout and each edge $e_{ij} \in E$ is used to characterize relationships between features. E is composed of both conflict and stitch relationship, denoted by $E = \{CE \cup SE\}$, where SE is the set of stitch edges and CE is the set of conflict edges. One example is shown in Fig. 1(c), where the stitch edges are orange and the conflict edges are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

black. Previous work [8] formulates the MPLD problem as below:

$$\min_{\boldsymbol{x}} \quad \sum c_{ij} + \alpha \sum s_{ij}, \tag{9a}$$

$$\text{s.t.} \quad c_{ij} = (x_i == x_j), \qquad \forall e_{ij} \in CE, \tag{9b}$$

$$s_{ij} = (x_i \neq x_j), \qquad \forall e_{ij} \in SE, \tag{9c}$$

$$x_i \in \{0, 1, \ldots, k\}, \qquad \forall x_i \in \boldsymbol{x}, \tag{9d}$$

where x_i is defined as in Equation (1), c_{ij} is a binary variable representing the conflict edge $e_{ij} \in CE$, s_{ij} stands for the stitch edge $e_{ij} \in SE$, α, which is a user-defined parameter indicating the relative importance between the conflict cost and the stitch cost and set as 0.1 by default, If two nodes, v_i and v_j, within the minimal coloring distance are assigned the same color, i.e., $x_i = x_j$, then $c_{ij} = 1$. On the contrary, $s_{ij} = 1$ when two nodes connected by the stitch edge are assigned different colors, i.e., $x_i \neq x_j$. The objective function is to minimize the weighted sum of the conflict number and the stitch number.

Based on the objective function shown in Equation (9), the problem can be solved by ILP [6], [8], where x_i is represented by 1-bit 0-1 variable(s). The ILP model for TPLD can be formulated as in Formula (10), where the objective function of MPLD in Equation (9) can be directly applied in ILP-based formula, as shown in Equation (10a), constraints Equation (10c)–Equation (10g) play the same role as Equation (9b), where 0–1 variable c_{ij} is true only if two nodes connected by the conflict edge e_{ij} are assigned the same color.

$$\min \sum_{e_{ij} \in \text{CE}} c_{ij} + \alpha \sum_{e_{ij} \in \text{SE}} s_{ij} \tag{10a}$$

$$\text{s.t.} \quad x_{i1} + x_{i2} \leq 1, \tag{10b}$$

$$x_{i1} + x_{j1} \leq 1 + c_{ij1}, \qquad \forall e_{ij} \in \text{CE}, \tag{10c}$$

$$(1 - x_{i1}) + (1 - x_{j1}) \leq 1 + c_{ij1}, \qquad \forall e_{ij} \in \text{CE}, \tag{10d}$$

$$x_{i2} + x_{j2} \leq 1 + c_{ij2}, \qquad \forall e_{ij} \in \text{CE}, \tag{10e}$$

$$(1 - x_{i2}) + (1 - x_{j2}) \leq 1 + c_{ij2}, \qquad \forall e_{ij} \in \text{CE}, \tag{10f}$$

$$c_{ij1} + c_{ij2} \leq 1 + c_{ij}, \qquad \forall e_{ij} \in \text{CE}, \tag{10g}$$

$$|x_{j1} - x_{i1}| \leq s_{ij1}, \qquad \forall e_{ij} \in \text{SE}, \tag{10h}$$

$$|x_{j2} - x_{i2}| \leq s_{ij2}, \qquad \forall e_{ij} \in \text{SE}, \tag{10i}$$

$$s_{ij} \geq s_{ij1}, s_{ij} \geq s_{ij2}, \qquad \forall e_{ij} \in \text{SE}, \tag{10j}$$

$$x_{ij} \in \{0, 1\}. \tag{10k}$$

In Equation (10), c_{ij} is true when both c_{ij1} and c_{ij2} are true by the constraint Equation (10g). 0–1 variable $c_{ij1}(c_{ij2})$ demonstrates whether $x_{i1}(x_{i2})$ equals to $x_{j1}(x_{j2})$. Therefore, c_{ij} is true only when $x_i = x_j$, i.e., v_i and v_j are assigned the same color. Similarly, constraints Equation (10h) - Equation (10j) correspond to Equation (9c), where 0–1 variable s_{ij} is true only if v_i and v_j are assigned different colors.

B. New ILP-based Algorithm

It is no doubt that the cost of the MPLD problem is the weighted sum of the conflict cost and the stitch cost. However,

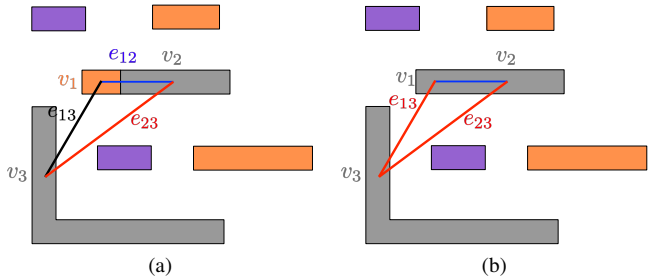


Fig. 5 An example of the non-optimality of the original ILP-based algorithm. (a) The solution of the original ILP-based algorithm, where one stitch (blue line) happens at the top of the conflict (red line). (b) The solution of our ILP-based algorithm, where no stitch is introduced and can obtain the optimal solution.

the previous ILP-based algorithm [8] measures the conflict cost by a summation of the binary variables representing conflict edges $e_{ij} \in CE$, i.e, $\sum c_{ij}$. Such a measurement method is not accurate and ignores a simple but important fact: conflict happens between features instead of nodes. In other words, If the stitch candidate divides one feature into two sub-features, which are represented by two nodes v_1, v_2 in the graph, and both nodes have a conflict edge with the third node v_3, i.e., $e_{12}, e_{13} \in CE$, then the previous conflict cost shown in Equation (10) will count both e_{13} and e_{23} while they represent the same conflict between features. Fig. 5 illustrates one example, where the result of the original ILP, as shown in Fig. 5(a), introduces one more stitch. The reason is: if the stitch edge e_{12} is ignored as shown in Fig. 5(b), i.e, the two connected nodes, v_1 and v_2, are assigned the same color and thus $x_1 = x_2$, the original cost function shown in Equation (10) will calculate the cost as 2 since both e_{13} and e_{23} are true. Therefore, ILP with the original problem formulation prefers to assign v_1 and v_2 with different colors, which results in a 1.1 cost value for the original cost function. However, it is easy to see that when this stitch is ignored, the conflict should be 1 instead of 2 since only one conflict between features happens.

Based on this observation, we present a new formulation shown in Equation (11). The objective function of the new formulation is the weighted sum of conflict cost ($\sum C_{mn}$) and stitch cost ($\sum s_{ij}$), which exactly matches the objective of the color assignment problem. The modified part is highlighted in blue. P indicates the feature set before stitch insertion, r_i and r_j are the sub-features after stitch insertion and belong to p_m and p_n respectively. For example, d_1 and d_2 are the sub-features of the original feature d in Fig. 1.

Given the new formula for MPLD, the problem can also be solved by ILP. The ILP model for TPLD is formulated in Equation (12): here the conflict cost is calculated by $\sum C_{mn}$ between the feature m and n instead of $\sum c_{ij}$ between node i and j. In Equation (12), C_{mn} is true when both C_{mn1} and C_{mn2} are true by the constraint formulated in Equation (12g). 0–1 variable $C_{mn1}(C_{mn2})$ demonstrates whether there exists $r_i \in p_m, r_j \in p_n, s.t.,\ x_{i1}(x_{i2}) = x_{j1}(x_{j2})$. By considering C_{mn}, the conflict cost between features instead of nodes, our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

$$\min_{\boldsymbol{x}} \quad \sum C_{mn} + \alpha \sum s_{ij}, \tag{11a}$$

$$\text{s.t.} \quad C_{mn} = \min\{ \sum_{\substack{r_i \in p_m, \\ r_j \in p_n, \\ c_{ij} \in \text{CE}}} (x_i == x_j), 1\}, \forall p_m, p_n \in P, \tag{11b}$$

$$s_{ij} = (x_i \neq x_j), \quad \forall e_{ij} \in SE, \tag{11c}$$

$$x_i \in \{0, 1, 2\}, \quad \forall x_i \in \boldsymbol{x}. \tag{11d}$$

$$\min \sum_{c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n} C_{mn} + \alpha \sum_{e_{ij} \in \text{SE}} s_{ij}, \tag{12a}$$

$$\text{s.t.} \quad x_{i1} + x_{i2} \leq 1, \tag{12b}$$

$$x_{i1} + x_{j1} \leq 1 + C_{mn1}, \quad \forall c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n, \tag{12c}$$

$$(1 - x_{i1}) + (1 - x_{j1}) \leq 1 + C_{mn1}, \quad \forall c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n, \tag{12d}$$

$$x_{i2} + x_{j2} \leq 1 + C_{mn2}, \quad \forall c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n, \tag{12e}$$

$$(1 - x_{i2}) + (1 - x_{j2}) \leq 1 + C_{mn2}, \quad \forall c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n, \tag{12f}$$

$$C_{mn1} + C_{mn2} \leq 1 + C_{mn}, \quad \forall c_{ij} \in \text{CE}, r_i \in p_m, r_j \in p_n. \tag{12g}$$

new ILP-based algorithm is able to capture the conflict cost accurately.

V. Flexible Exact cover-based algorithm

In this section, we first introduce the exact cover (EC)-based algorithm proposed by [19], and then some non-optimal examples are discussed. Finally, we propose a flexible EC-based algorithm, which achieves a trade-off between quality and runtime and therefore outperforms the previous algorithm on the quality with a sacrifice in the runtime.

A. Exact Cover (EC)-based Algorithm

Though our ILP is able to obtain the optimal solution of the objective function, it suffers from runtime for large graphs. EC-based algorithm [19] models the MPLD problem as an exact cover problem, which can be efficiently solved by a customized and augmented combination of dancing links data structure and Algorithm X* (DLX). Generally speaking, the layout is represented by a homogeneous graph. The graph is further translated into a 0-1 matrix and then can be solved as an exact cover problem of the obtained matrix. Each column index in the matrix can be viewed as the element of a universe U to be covered, and each row can be viewed as a subset of the universe. The final solution (a set of rows) of the exact cover problem is then translated back to the solution (coloring results of each node) of the graph coloring problem.

The details of the EC-based algorithm are shown in Algorithm 2. The input of the algorithm is a no-stitch graph

Algorithm 2 ExactCoverSolver

Input: $G_p \leftarrow$ No-stitch graph;
Ourput: Coloring solution;
1: Convert G_p into exact cover matrix M;
2: Call X* with G_p and M;
3: **if** X* exits with a solution **then**
4: **return** the found solution;
5: **else**
6: Construct the stitch-inserted graph G'_p based on G_p;
7: Construct the new exact cover matrix M' based on M;
8: **while** no solution is found **do**
9: Call X* with G'_p and M';
10: **if** X* exits without a solution **then**
11: Remove the exact conflict edge in G'_p and M';
12: **end if**
13: **end while**
14: **return** the found solution;
15: **end if**

$G_p = \{V_p, E_p\}$, which is obtained from the layout features and each feature represents exactly one node in G_p. The algorithm first tries to solve the exact cover problem induced by the graph coloring problem on G_p, in which no stitch is introduced (lines 1–4). To be more specific, the target graph G_p is translated into a corresponding exact cover matrix M (line 1). Then, algorithm X* is called to solve M (line 2). The details of algorithm X* are illustrated in [19]. If one feasible solution is found by X*, then the solution is returned (lines 3–4). Otherwise, the algorithm is going to solve the exact cover problem induced by the graph coloring problem on the stitch-inserted graph G'_p (lines 5–15). Here, $G'_p = \{V'_p, E'_p\}$ is obtained by splitting the nodes in G_p whose corresponding features have stitch candidates and new edges are added following the distance constraints (line 6). The algorithm then builds up the new matrix M' based on M (line 7) and calls algorithm X* to solve M' (line 9). Furthermore, if graph G'_p is still un-colorable, the detected exact conflict edge will be remarked as the reason for un-colorability and removed in G'_p and M' (line 11). Such a procedure is repeated until G'_p is colorable and the final coloring solution is found (lines 8–13).

In the exact cover matrix M translated from G_p, all nodes in V_p are inserted into the universe U. In addition, for each edge $e \in E_p$, k elements $e_c, s.t.\ c \in \{1, ..., k\}$ are inserted into U for the k-coloring problem. Therefore, the total size of U (also the column size of M) is $\mathcal{O}(|V_p| + k|E_p|)$. For each node $v \in V_p$, k subsets $S_c^v, s.t.\ c \in \{1, ..., k\}$ corresponding to k available colors are created, where each subset contains the node element $v \in U$ and e_c for each edge $e = \{u, v\} \in E_p$. e_c is inserted into both S_c^v and S_c^u and thus prevents u, v from being assigned to the same color, which represents the conflict constraint between u and v. Therefore, the total size of the subsets (also the row size of M) is $\mathcal{O}(k|V_p| + k|E_p|)$. One DPLD example of the translation from G_p to M is shown in Fig. 6, where row 1,4,6 are selected as the final solution of the exact cover problem so that the corresponding coloring solution is given and shown in Fig. 6.

In the converted matrix with stitch insertion, M', besides the original rows (subsets) in M, additional rows are added below the original rows. Specifically, for each stitch can-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

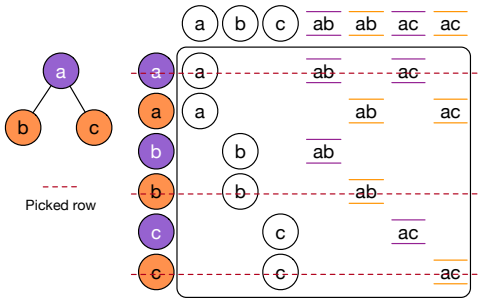


Fig. 6 Double patterning instance with its exact cover matrix.

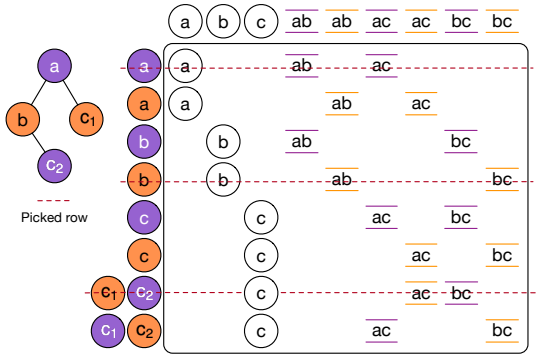


Fig. 7 Double patterning instance containing stitch edge with its exact cover matrix.

didate e_c, which splits the parent node $v \in V_p$ into two nodes $v'_1, v'_2 \in V'_p$, $k(k-1)$ subsets $S^v_{c_1c_2}, s.t., \ c_1, c_2 \in \{1, ..., k\}, c_1 \neq c_2$ corresponding to $k(k-1)$ available coloring solutions of v'_1 and v'_2 are created, where each subset contains the node element $v \in U$ and e_{c_1}, e_{c_2} for each edge $e_{c_1} = \{v'_1, v'\} \in E'_p, e_{c_2} = \{v'_2, v'\} \in E'_p$. $e_{c_1}(e_{c_2})$ inserted in $S^v_{c_1c_2}$ is to prevent $v'_1(v'_2)$ and v' from being assigned to the same color. Therefore, the total number of newly-added rows is $\mathcal{O}(k^2|E_s|)$, where $|E_s|$ is the number of stitch candidates in all features of G_p. Fig. 7 gives an example of G'_p and M', where the 7_{th} row is selected as the part of the final solution so that one stitch candidate is used to avoid the conflict. When graph G'_p is still un-colorable, the exact conflict edge detected by algorithm X* is marked and removed. Such procedure is repeated until G'_p is colorable and the final coloring solution is found.

B. Flexible Exact Cover-based algorithm

The exact cover-based algorithm shows impressive performance improvement due to the efficient augmenting DLX. However, the algorithm cannot always guarantee the optimality of the results. Here, we propose two techniques to improve the exact cover-based algorithm.

1) Flexible Stitch Handling: The first possible reason for the non-optimality is the handling rules for stitch cases. Although the exact cover-based algorithm considers all stitch candidates on features concurrently, the example demonstrated in [19] uses at most one stitch to resolve conflict in a single feature since the rows in M' are added in the unit of stitch candidate. However, there are some features in which multiple stitch candidates are able to resolve multiple conflicts. One example is shown in Fig. 8, where our algorithm uses two stitches in the feature d and generates a result with cost 0.2, while the original EC-based algorithm only uses one stitch and generates a coloring result with cost 1.1. Although some commercial decomposition tools based on [19] have considered multiple stitch cases to improve the solution quality, related techniques are not detailed in [19]. In OpenMPL, we formalize the flexible stitch handling method by introducing a maximally usable stitch candidate number n and quantifying the complexity of the EC-based algorithm with different n.

The direct reason for the non-optimality in the stitch handling is, the rows are added in the unit of the stitch candidate, which constrains at most one stitch candidate to be selected for each node. To overcome such constraint, i.e., to use an arbitrary number of stitch candidates in one feature, we handle the stitch in the unit of node element. We present our optimal stitch handling method as follows: denote the maximal number of usable stitch candidates as n, which is a controllable parameter. For each node element $v \in V_p$, if the corresponding feature of v contains t_v stitch candidates and thus the feature is divided into $t_v + 1$ sub-features, the original stitch handling approach is going to insert $t_v(k^2 - k)$ rows while we insert $C^m_{t_v}(k^{m+1} - k)$ rows, where m is the number of used stitch candidates, which split the feature of v into $m + 1$ sub-features and is calculated by the minimum value between n and t_v, i.e., $m = \min\{n, t_v\}$. In our flexible algorithm, each row indicates one possible coloring solution for the divided $m+1$ sub-features. Clearly, when n equals one, the algorithm is the same as the previous one, i.e., only one stitch candidate is used in each feature and the space complexity is also $\mathcal{O}(t_v k^2)$. When n becomes large enough, i.e., $m = t_v$, the algorithm will use all stitch candidates at the same time, which is more possible to be optimal. However, the large n increases the space complexity to $\mathcal{O}(k^{t_v+1})$ and thus exponentially worsens the runtime. One DPLD example is shown in Fig. 8, where Fig. 8(a) is the matrix and corresponding coloring solution following the original stitch handling approach. $(d_1, d_2), (d_3, d_4)$ are the sub-features divided by two stitch candidates respectively and one conflict is introduced. Fig. 8(b) shows the results for our flexible stitch handling, where d_1, d_2, and d_3 are the sub-features divided by two stitch candidates at one time, and all conflicts are resolved by stitches. Although the proposed stitch handling approach can obtain optimal results, it suffers from efficiency due to the explosion of the number of newly-added rows, especially when n and t_v are large. To speed up our algorithm without additional quality loss, we further use a heuristic technique. Firstly, the graph follows the original stitch handling approach, i.e., $n = 1$. If all conflicts are resolved by stitches or the graph contains no features whose number of stitch candidates is more than one, then the coloring procedure completes and the optimal stitch handling is not used. Otherwise, the graph is further handled by our flexible algorithm with n as the maximal number of stitch candidates in the features, which is closer to the optimal solution.

2) Optimized Traversal Order: Another possible reason for the non-optimality is the traversal order of nodes for the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

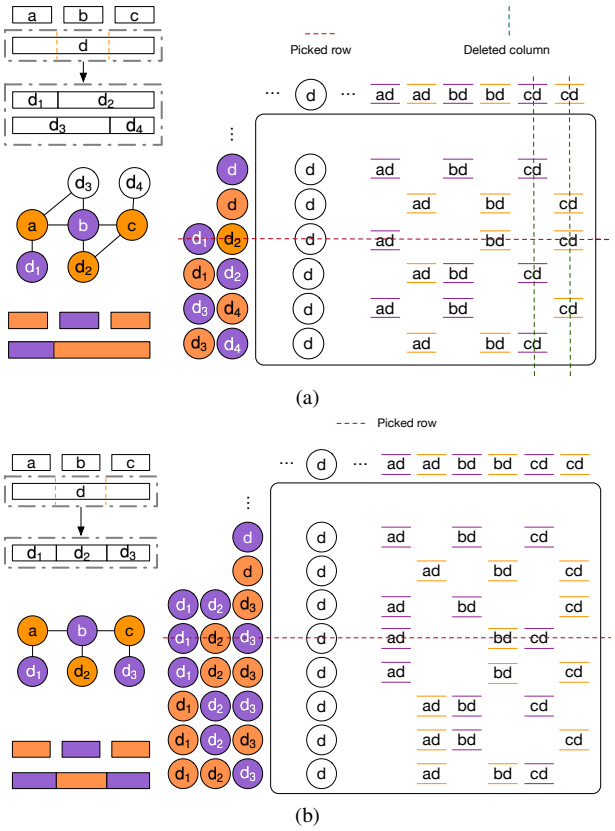


Fig. 8 (a) The non-optimal case of original EC-based algorithm. (b) The same case by our flexible EC-based algorithm, in which the result is optimal.

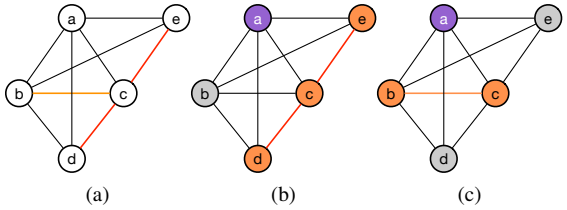


Fig. 9 The non-optimal case of the original EC-based algorithm due to the traversal order. (a) The exact conflict(s) selected by the original rule (red) and ours (orange). (b) The coloring results by the original rule. (c) The coloring results by our optimized rule.

conflict-overlapping cases. Here, we formallly define such a case as the **overlapping k-clique**:

Definition 1. *For a homogeneous graph $G = \{V, E\}$, where $V = \{V_s, v_1, v_2\}$, G is called an **overlapping k-clique** if $(v_1, v_2) \notin E$ and the subgraphs $G_1 = G \backslash v_1$ and $G_2 = G \backslash v_2$ are both k-cliques where $k > 2$.*

One example of the overlapping 4-clique is given in Fig. 9, where $V_s = \{a, b, c\}$, $v_1 = d$ and $v_2 = e$. With the definition, the following theorem shows the cost of the optimal solution for an overlapping k-clique in the $|V_s|$-coloring problem.

Theorem 2. *The optimal solution for an overlapping k-clique in the $|V_s|$-coloring problem has exactly one conflict.*

Proof. It is obvious that the optimal solution for a k-clique in the $|V_s|$-coloring problem has exactly one conflict since $|V_s| = k - 1$. Therefore, the optimal solution for both G_1 and G_2 has exactly one conflict, which results in a lower bound of the conflict number for graph G as one. We then prove that there exists a feasible solution $f : V \to \{1, ..., k-1\}$ which colors G within one conflict. Let's define f as follows: Given any two different nodes in V_s, v_1^s and v_2^s , f first assigns color 1 to v_1^s and v_2^s and color 2 to v_1 and v_2, which generates one conflict. Then f assigns colors $\{3, ..., |V_s|\}$ to the left nodes in V_s, i.e., $V_s \backslash \{v_1^s, v_2^s\}$. Since the size of $V_s \backslash \{v_1^s, v_2^s\}$ is $|V_s| - 2$, which is the same as the size of available colors, this coloring procedure is conflict-free. Totally, the number of conflicts is one under this coloring scheme and such completes the proof. □

Although the minimum conflict of an overlapping k-clique is 1 as stated in Theorem 2, the quality of the results by algorithm X* is highly dependent on the traversal order. Original algorithm X* in [19] traverses the node in the BFS order unless one uncovered node has only one possible color. The root of BFS is the node whose corresponding feature has the largest area. If there are multiple available nodes, nodes will be selected following a numerical order in the implementation. However, such BFS-based traversal order may fail to obtain the optimal solution in some overlapping k-cliques as mentioned in [19]. Such a situation can be formally described as:

Claim 1. *The solution for an overlapping k-clique in the $|V_s|$-coloring problem by algorithm X* with the BFS-based traversal order proposed in [19] cannot guarantee optimality.*

Proof. The proof can be finished by a simple non-optimal case. Assume that $k > 2$, the corresponding feature of node v_1 has the largest area, which makes v_1 the root of BFS, and v_2 is the node at the end of the numerical order, then the detected exact conflict edge, i.e., the last reported conflict edge, must be the edge between v_2 and v_i^s, where v_i^s is the node in V_s. Therefore, edge $\{v_2, v_i^s\}$ is removed and one conflict happens. However, the sub-graph G_2 is still a k-clique and contributes to one conflict in the $|V_s|$-coloring problem besides the edge $\{v_2, v_i^s\}$. Therefore, such a traversal order finally results in at least two conflicts totally, which is not optimal. □

One example of non-optimality is shown in Fig. 9(b). The edge c-e is first marked as an exact conflict and then one more conflict c-d is introduced because the left sub-graph a-b-c-d still forms a 4-clique. Considering the non-optimal case of original traversal order, we propose a heuristic traversal order which is nearer to the optimal solution. The differences of our optimized traversal order are organized as follows: (1) The root of BFS is the node with the largest degree; (2) If nodes are in the same depth in the BFS, the node with the smallest degree is selected; (3) If there are multiple uncovered nodes that have only one possible color, the node with the maximal degree is selected. Through these special treatments, the new traversal order is optimal for the k-clique and can be formally described as:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I Effective of stitch redundancy removal (SRR)

Circuit	Our EC w/o. SRR		Our EC w. SRR	
	time (s)	cost	time (s)	cost
test1_100	2.163	385.9	2.866	385.9
test5_101	0.013	625.3	0.008	625.3
test6_102	1.441	352.8	1.331	352.8
test8_100	2.889	6238.1	2.254	6238.1
test9_100	5.064	9651.2	3.877	9651.2
test10_100	11.716	11129.3	9.783	11129.3
average	3.881	4730.433	3.353	4730.433
ratio	1.000	1.000	0.864	1.000

TABLE II Our ILP vs. Original ILP [8] on ISCAS benchmarks

Circuit	Original ILP [8]				Our ILP			
	time (s)	st#	cn#	cost	time (s)	st#	cn#	cost
C432	0.050	4	0	0.4	0.054	4	0	0.4
C499	0.029	0	0	0	0.017	0	0	0
C880	0.045	7	0	0.7	0.039	7	0	0.7
C1355	0.045	3	0	0.3	0.038	3	0	0.3
C1908	0.078	1	0	0.1	0.063	1	0	0.1
C2670	0.049	6	0	0.6	0.055	6	0	0.6
C3540	0.055	8	1	1.8	0.059	8	1	1.8
C5315	0.065	9	0	0.9	0.060	9	0	0.9
C6288	0.961	205	1	21.5	1.075	204	1	21.4
C7552	0.105	23	0	2.3	0.146	23	0	2.3
S1488	0.030	2	0	0.2	0.031	2	0	0.2
S38417	0.581	54	19	24.4	0.635	54	19	24.4
S35932	1.641	40	44	48	1.478	40	44	48
S38584	1.540	116	36	47.6	1.541	116	36	47.6
S15850	1.604	97	34	43.7	1.479	97	34	43.7
average	0.459	38.333	9.000	12.833	0.451	38.267	9.000	12.827
ratio	1.000	1.000	1.000	1.000	0.984	0.998	1.000	0.999

Theorem 3. *The solution for an overlapping k-clique in the $|V_s|$-coloring problem by algorithm X* with the new traversal order guarantees optimality.*

Proof. Let v_i^s be the root, $v_i^s \in V_s$ since the root has the largest degree. Because both v_1 and and v_2 have the smallest degree, which will be selected firstly, the last detected conflict is $\{v_i^s, v_j^s\}$, where $v_j^s \in V_s$. After the edge $\{v_i^s, v_j^s\}$ is removed, both G_1 and G_2 are not k-cliques and can be colored by algorithm X^* without additional conflict. Therefore, the total number of conflicts by algorithm X* with the new traversal order is one, which is optimal according to the Theorem 2 and completes the proof. □

One example of the new traversal order is shown in Fig. 9(c), where our new traversal order marks b-c as the exact conflict and thus achieves optimality.

VI. Experimental Results

We implement OpenMPL in C++ and use Boost [31] as the basic graphics library. All of the experiments are tested on an Intel Core 2.9 GHz Linux machine. We conduct experiments under two series of benchmarks. The first smaller benchmarks are the scaled-down and modified versions of ISCAS benchmarks, which are widely used in previous works. The minimum coloring spacing is set to 120 nm for the first ten cases and 100 nm for the last five cases, as in [8], [13], [19]. The second larger benchmarks are the ISPD'19 benchmarks for detailed routing. We use the metal layers in the benchmark obtained by Dr.CU 2.0 [32] and set the minimum coloring spacing as $k \cdot s + (k-1) \cdot w$, where k is the number of colors and set as 3 in our experiments, s is the minimum spacing between two features and w is the standard width of one feature. Here, we only show the results of metal layers which can be decomposed by our ILP within three hours (6 cases in total). Each selected layer with id n on benchmark m is represented by m_n. For example, test1_100 represents the layer with id 100 on the test1 benchmark of ISPD2019. We only focus on the results of different decomposition algorithms on the TPLD problem due to page limit, which is more difficult to obtain optimal results compared with DPLD. More detailed results and discussions can be found in [1]. The stitch weight α is set to 0.1, the thread number is 8 and the graph simplification level is 3 which represents that the framework enables three simplification techniques:ICC, IVR, and BCE. Especially, SDP is set to one thread due to no maintenance of CSDP now. Fig. 10 shows the decomposition results for the case C432 of ISCAS benchmarks and the case test1_100 of ISPD benchmarks.

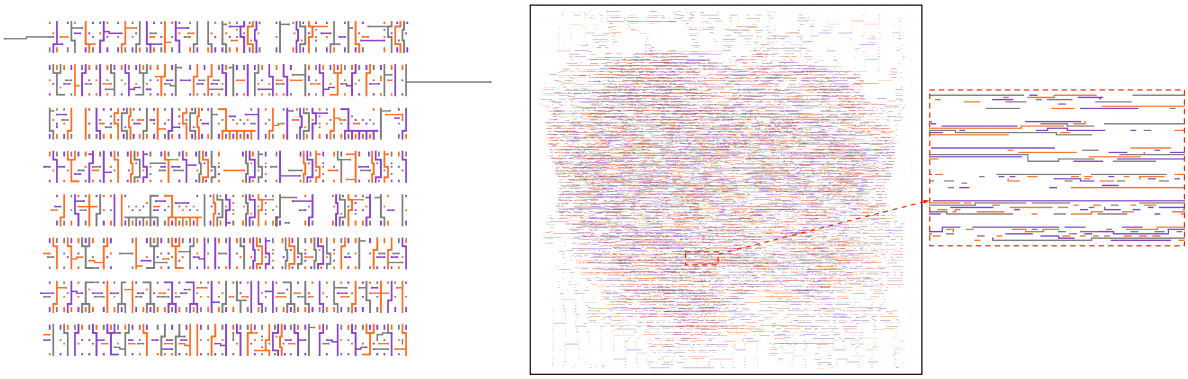
A. Effectiveness of Stitch Redundancy Removal

Firstly, we demonstrate the effectiveness of the proposed stitch redundancy removal (SRR) technique. Through stitch redundancy removal, some redundant stitch candidates can be removed and the two connected nodes are merged into one node. Therefore, the graph size for the decomposition is reduced and thus the decomposition runtime is decreased without decomposition quality loss theoretically. We only conducted SRR on the graphs whose sizes are larger than 8. TABLE I compares the performance and runtime on the target graphs. Column "time (s)" is the total simplification and decomposition runtime of graphs which have redundant stitches to be removed. The "cost" column is the total decomposition cost of our EC. When the case is sparse, i.e., the case can be easily simplified such that the total number of simplified graphs is huge while the size of each graph is usually small, our SRR may harm the runtime since the number of redundant stitches is not very much while the runtime for scanning all stitches in SRR cannot be avoided. For example, the runtime for graphs on test1_100 is increased from 2.163 seconds to 2.866 seconds when SRR is used. Despite such a sparse case, which may not be the major bottleneck due to its low complexity, our SRR shows a considerable runtime improvement in most cases. We can see that compared with decomposing the graph by our EC directly, further applying SRR can reduce the average runtime by 13.6% without any performance loss.

B. Original ILP Versus Our ILP

Secondly, we compare our ILP with the original ILP proposed by [8] on both small ISCAS benchmarks and large ISPD benchmarks. The results are shown in TABLE II for ISCAS benchmarks and TABLE III for ISPD benchmarks. The column "time (s)" is the real time of decomposition in seconds instead of CPU time. Columns "st#" and "cn#" are the stitch number and the conflict number, "cost" is the decomposition cost calculated by Equation (12). On the small benchmarks, our ILP shows a slight improvement in both

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3042175, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) (b)

Fig. 10 An example of the decomposition results. (a) Decomposition result of the circuit `C432` in the ISCAS benchmarks. (b) Decomposition result of the 100_{th} layer in the circuit `test1` in the ISPD benchmarks.

TABLE III Our ILP vs. Original ILP [8] on ISPD benchmarks

Circuit	Original ILP [8]				Our ILP			
	time (s)	st#	cn#	cost	time (s)	st#	cn#	cost
test1_100	122.249	299	243	272.9	115.075	239	219	242.9
test5_101	118.029	232	466	489.2	149.040	190	433	452
test6_102	235.359	482	115	163.2	398.106	454	108	153.4
test8_100	19.025	4616	5683	6144.6	17.277	4389	5567	6005.9
test9_100	28.365	6969	8739	9435.9	25.534	6643	8559	9223.3
test10_100	110.479	9594	9775	10734.4	99.529	8945	9555	10449.5
average	105.584	3698.667	4170.167	4540.033	134.094	3476.667	4073.500	4421.167
ratio	1.000	1.000	1.000	1.000	1.270	0.940	0.977	0.974

TABLE IV Our EC vs. Original EC [19] on ISCAS benchmarks

Circuit	Original EC [19]				Our EC			
	time (s)	st#	cn#	cost	time (s)	st#	cn#	cost
C432	0.005	4	0	0.4	0.008	4	0	0.4
C499	0.004	0	0	0	0.006	0	0	0
C880	0.005	7	0	0.7	0.007	7	0	0.7
C1355	0.007	3	0	0.3	0.018	3	0	0.3
C1908	0.008	1	0	0.1	0.022	1	0	0.1
C2670	0.014	6	0	0.6	0.021	6	0	0.6
C3540	0.029	8	1	1.8	0.035	8	1	1.8
C5315	0.019	9	0	0.9	0.033	9	0	0.9
C6288	0.114	203	8	28.3	0.142	204	1	21.4
C7552	0.028	21	1	3.1	0.055	21	1	3.1
S1488	0.008	2	0	0.2	0.007	2	0	0.2
S38417	0.127	54	19	24.4	0.175	54	19	24.4
S35932	0.286	48	44	48.8	0.299	40	44	48
S38584	0.291	117	36	47.7	0.323	117	36	47.7
S15850	0.285	100	34	44	0.342	100	34	44
average	0.082	38.867	9.533	13.42	0.1	38.4	9.067	12.907
ratio	1.000	1.000	1.000	1.000	1.220	0.988	0.951	0.962

the runtime and the quality. The time is reduced by 1.6% and the stitch number is reduced by 1 on circuit `C6288` while the costs on other circuits are not changed, which indicates that such re-count case in the small benchmarks is not frequent. On the large benchmarks, Our ILP reduces 222 stitches and 96.67 conflicts averagely, i.e., from 3698.667 to 3476.667 and from 4170.167 to 4073.5. Therefore, the average cost is significantly reduced by 118.867 while the runtime is increased by 27%. However, such runtime loss is acceptable considering the unignorable quality improvement.

C. Original EC Versus Our EC

Thirdly, we compare our EC with the original EC proposed by [19] on both small and large benchmarks. The results are listed in TABLE IV for ISCAS benchmarks and TABLE V for ISPD benchmarks. As discussed in Section V-B, the original EC assumes exactly one stitch candidate to be activated for each feature, which reduces the matrix size and thus reduces the time complexity with a potential quality loss. Our EC assumes that at most n stitch candidates are activated for each feature, where n is a dynamic parameter and therefore we can achieve a flexible balance between runtime and quality by changing n. The results in both the small and large benchmarks demonstrate our analysis, where n is set to 2 in our EC, i.e., at most two stitch candidates are activated for each feature. Our EC reduces the average cost by 3.8% on the small benchmarks and 2.9% on the large benchmarks. As a tradeoff, the average runtime is increased from 0.082s to 0.1s on the small benchmarks and from 6.996s to 30.776s on the large benchmarks, which is not trivial and demonstrates one of the drawbacks for the EC-based algorithm: the increase of n results in an exponentially increasing on the runtime.

D. Comparison of Different Decomposers

Fourthly, we compare different decomposers in both stitch-enabled cases and no-stitch cases. The quality results without stitch for ISCAS benchmarks are listed in TABLE VI. Column "Back." is the result of the backtracking algorithm introduced in [33]. As shown in TABLE VI, our ILP, MIS [13], and backtracking [33] in our implementation obtain the optimal

TABLE V Our EC vs. Original EC [19] on ISPD benchmarks

Circuit	Original EC [19]				Our EC			
	time (s)	st#	cn#	cost	time (s)	st#	cn#	cost
test1_100	1.772	236	383	406.6	11.521	279	358	385.9
test5_101	3.229	282	615	643.2	21.052	303	595	625.3
test6_102	7.209	560	327	383	57.525	558	297	352.8
test8_100	5.585	4236	5994	6417.6	10.269	4561	5782	6238.1
test9_100	9.042	6329	9270	9902.9	17.139	6852	8966	9651.2
test10_100	15.14	8697	10621	11490.7	67.149	9433	10186	11129.3
average	6.996	3390	4535	4874	30.776	3664.333	4364	4730.433
ratio	1.000	1.000	1.000	1.000	4.399	1.081	0.962	0.971

TABLE VI Non-stitch Decomposition Cost Comparison on ISCAS benchmarks

Circuit	Our ILP	LP [11]	MIS [13]	SDP [8]	EC [19]	Back. [33]
C432	4	4	4	4	4	4
C499	0	0	0	0	0	0
C880	7	7	7	7	7	7
C1355	3	3	3	3	3	3
C1908	1	1	1	1	1	1
C2670	6	6	6	6	6	6
C3540	9	9	9	9	9	9
C5315	9	9	9	9	9	9
C6288	205	205	205	205	205	205
C7552	22	22	22	22	22	22
S1488	2	2	2	2	2	2
S38417	95	97	95	95	97	95
S35932	157	166	157	159	163	157
S38584	230	233	230	231	231	230
S15850	212	215	212	212	215	212
average	64.133	65.267	64.133	64.333	64.933	64.133
ratio	1.000	1.018	1.000	1.003	1.012	1.000

solution while other relaxation-based or heuristic methods degrade the result quality.

For the stitch-enabled cases, the quality comparison is shown in TABLE VII and TABLE VIII; The runtime comparison is shown in TABLE IX and TABLE VIII. Especially, backtracking is not shown in TABLE VIII, since it cannot be processed within three hours for any layout in ISPD benchmarks. The results of MIS [13] and LUT [17] in TABLE VII are directly quoted from their papers. The ratio is calculated based on the results of our ILP. For the decomposition cost, our optimized ILP outperforms other algorithms and achieves the best cost performance as expected. On the small benchmarks, SDP is the worst and increases the cost by 28.4% while the cost of our EC and backtracking are close to the ILP. On the large benchmarks, SDP only increases the cost by 4.4% and is better than our EC, which increases the cost by 7%. For the runtime, the original EC is the best due to the efficient augmenting DLX technique. Backtracking shows a good runtime performance on the small benchmarks due to our heuristic algorithm but fails to obtain the results on the large benchmarks within three hours. The runtime of SDP is much worse than our ILP on the small benchmarks, i.e., $2.572\times$ runtime, while much better on the large benchmarks, whose ratio is close to our EC, i.e., 0.27 vs. 0.23.

VII. Conclusion and Future Work

In this paper, we proposed OpenMPL, a general framework for the multiple patterning layout decomposition problem, with efficient implementations of various state-of-the-art simplification and decomposition algorithms. Besides the re-implementation of previous algorithms, we optimized several algorithms based on some typical non-optimal cases. We presented a new simplification algorithm to remove the redundant stitches. Then, we proposed the correct problem formulation followed by a corresponding new ILP-based algorithm, which captures the objective of the color assignment problem accurately. Furthermore, we proposed a flexible EC-based algorithm, which achieves a trade-off between quality and runtime and therefore outperforms the previous algorithm on quality with a sacrifice in the runtime. The experiments demonstrate the effectiveness of our proposed algorithms and the efficiency of OpenMPL. In the future, we plan to integrate post-refinement into the workflow and further accelerate the EC-based algorithm. We believe that this open platform paves the road for the development of MPLD engines and will stimulate more research in the near future, eventually contributing to better manufacturability in advanced technology nodes.

References

[1] "OpenMPL," https://github.com/limbo018/OpenMPL.

[2] H.-Y. Chen, Y.-T. Hou, K. Yu-Hsiang, K.-H. Hsieh, R.-G. Liu, and L.-C. Lu, "Layout optimization for integrated circuit design," 2017, US Patent.

[3] D. Z. Pan, L. Liebmann, B. Yu, X. Xu, and Y. Lin, "Pushing multiple patterning in sub-10nm: Are we ready?" in *ACM/IEEE Design Automation Conference (DAC)*, 2015, pp. 197:1–197:6.

[4] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: A systematic review," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, pp. 1–6.

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

[6] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, pp. 939–952, June 2010.

[7] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 2, pp. 185–196, Feb. 2010.

[8] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 3, pp. 433–446, March 2015.

[9] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 163–169.

TABLE VII Decomposition Cost Comparison on ISCAS benchmarks

Circuit	Our ILP			SDP [8]			Our EC			Back. [33]			MIS [13]			LUT [17]		
	st#	cn#	cost	st#	cn#	cost	st#	cn#	cost	st#	cn#	cost	st#	cn#	cost	st#	cn#	cost
C432	4	0	0.4	4	0	0.4	4	0	0.4	4	0	0.4	6	0	0.6	4	0	0.4
C499	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C880	7	0	0.7	8	0	0.8	7	0	0.7	7	0	0.7	8	1	1.8	7	0	0.7
C1355	3	0	0.3	3	0	0.3	3	0	0.3	3	0	0.3	4	1	1.4	3	0	0.3
C1908	1	0	0.1	1	0	0.1	1	0	0.1	1	0	0.1	1	0	0.1	1	0	0.1
C2670	6	0	0.6	6	0	0.6	6	0	0.6	6	0	0.6	11	2	3.1	6	0	0.6
C3540	8	1	1.8	8	1	1.8	8	1	1.8	8	1	1.8	11	3	4.1	8	1	1.8
C5315	9	0	0.9	9	0	0.9	9	0	0.9	9	0	0.9	11	3	4.1	9	0	0.9
C6288	204	1	21.4	203	7	27.3	204	1	21.4	205	1	21.5	243	20	44.3	191	14	33.1
C7552	23	0	2.3	23	0	2.3	21	1	3.1	23	0	2.3	37	3	6.7	22	0	2.2
S1488	2	0	0.2	2	0	0.2	2	0	0.2	2	0	0.2	4	0	0.4	2	0	0.2
S38417	54	19	24.4	46	27	31.6	54	19	24.4	54	19	24.4	82	20	28.2	55	19	24.5
S35932	40	44	48	20	64	66	40	44	48	40	44	48	63	46	52.3	41	44	48.1
S38584	116	36	47.6	105	48	58.5	117	36	47.7	116	36	47.6	176	36	53.6	116	36	47.6
S15850	97	34	43.7	83	48	56.3	100	34	44	97	34	43.7	146	36	50.6	100	34	44
average	38.27	9.00	12.83	34.73	13	16.47	38.4	9.07	12.91	38.33	9.00	12.83	53.48	11.47	16.75	37.67	9.87	13.63
ratio	1.00	1.00	1.00	0.91	1.44	1.28	1.00	1.01	1.01	1.00	1.00	1.00	1.39	1.27	1.31	0.98	1.10	1.06

* The results of MIS [13] and LUT [17] are directly quoted from their papers.

TABLE VIII Decomposition comparison on ISPD benchmarks

Circuit	Our ILP				SDP [8]				Our EC			
	st#	cn#	cost	time	st#	cn#	cost	time	st#	cn#	cost	time
test1_100	239	219	242.9	115.075	287	269	297.7	4.929	279	358	385.9	11.521
test5_101	190	433	452	149.04	228	527	549.8	10.52	303	595	625.3	21.052
test6_102	454	108	153.4	398.106	477	144	191.7	67.856	558	297	352.8	57.525
test8_100	4389	5567	6005.9	17.277	4547	5750	6204.7	22.526	4561	5782	6238.1	10.269
test9_100	6643	8559	9223.3	25.534	6880	8842	9530	35.01	6852	8966	9651.2	17.139
test10_100	8945	9555	10449.5	99.529	9457	9963	10908.7	76.583	9433	10186	11129.3	67.149
average	3476.667	4073.5	4421.167	134.094	3646	4249.167	4613.767	26.237	3664.333	4364	4730.433	30.776
ratio	1.000	1.000	1.000	1.000	1.049	1.043	1.044	0.270	1.054	1.071	1.070	0.230

TABLE IX Decomposition Runtime (s) Comparison on ISCAS benchmarks

Circuit	Our ILP	SDP [8]	Our EC	Back. [33]
C432	0.054	0.027	0.008	0.003
C499	0.017	0.016	0.006	0.004
C880	0.039	0.046	0.007	0.004
C1355	0.038	0.024	0.018	0.005
C1908	0.063	0.028	0.022	0.01
C2670	0.055	0.05	0.021	0.015
C3540	0.059	0.076	0.035	0.163
C5315	0.06	0.085	0.033	0.022
C6288	1.075	0.615	0.142	1.308
C7552	0.146	0.209	0.055	0.02
S1488	0.031	0.031	0.007	0.007
S38417	0.635	1.673	0.175	0.262
S35932	1.478	5.118	0.299	0.299
S38584	1.541	4.99	0.323	0.281
S15850	1.479	4.416	0.342	0.63
average	0.451	1.16	0.1	0.202
ratio	1.000	2.572	0.222	0.448

[10] B. Yu, S. Roy, J.-R. Gao, and D. Z. Pan, "Triple patterning lithography layout decomposition using end-cutting," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, vol. 14, no. 1, pp. 011 002–011 002, 2015.

[11] Y. Lin, X. Xu, B. Yu, R. Baldick, and D. Z. Pan, "Triple/quadruple patterning layout decomposition via linear programming and iterative rounding," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, vol. 16, no. 2, 2017.

[12] X. Li, Z. Zhu, and W. Zhu, "Discrete relaxation method for triple patterning lithography layout decomposition," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 285–298, 2017.

[13] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 3, pp. 397–408, March 2014.

[14] 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 *ACM International Symposium on Physical Design (ISPD)*, 2015, pp. 67–74.

[15] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. D. F. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 57–64.

[16] J. Kuang and E. F. Y. Young, "Fixed-parameter tractable algorithms for optimal layout decomposition and beyond," in *ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 61:1–61:6.

[17] ——, "An efficient layout decomposition approach for triple patterning lithography," in *ACM/IEEE Design Automation Conference (DAC)*, 2013, pp. 69:1–69:6.

[18] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 170–177.

[19] H.-Y. Chang and I. H.-R. Jiang, "Multiple patterning layout decomposition considering complex coloring rules," in *ACM/IEEE Design Automation Conference (DAC)*, 2016, pp. 40:1–40:6.

[20] W. Fang, S. Arikati, E. Cilingir, M. A. Hug, P. De Bisschop, J. Mailfert, K. Lucas, and W. Gao, "A fast triple-patterning solution with fix guidance," in *Proceedings of SPIE*, vol. 9053, 2014.

[21] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 81–88.

[22] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[23] "Limbo," http://yibolin.com/Limbo/docs/html/index.html.

[24] Gurobi Optimization Inc., "Gurobi optimizer reference manual," http://www.gurobi.com, 2016.

[25] "LEMON," http://lemon.cs.elte.hu/trac/lemon.
[26] "CBC," http://www.coin-or.org/projects/Cbc.xml.
[27] T. Matsui, Y. Kohira, C. Kodama, and A. Takahashi, "Positive semidefinite relaxation and approximation algorithm for triple patterning lithography," in *International Symposium on Algorithms and Computation (ISAAC)*, 2014, pp. 365–375.
[28] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *ACM/IEEE Design Automation Conference (DAC)*, 2014, pp. 53:1–53:6.
[29] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613–623, 1999.
[30] "OpenMP," http://www.openmp.org/.
[31] "Boost C++ Library," http://www.boost.org.
[32] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Young, "Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.
[33] W. Klotz, *Graph coloring algorithms*. Verlag nicht ermittelbar, 2002.

Wei Li received his B.Sc. degree in computer science from Chung Chi College, the Chinese University of Hong Kong (CUHK) in 2018. He is currently pursuing his MPhil degree at CUHK. He received Best Student Paper Award from ICTAI'2019 and Distinguished Paper Award from ISSTA'19. He was also the Richard Newton Young Student Fellow in DAC 2020. His research interests include explorations on Graph Neural Networks and its applications on VLSI design.

Yuzhe Ma received his B.E. degree from the Department of Microelectronics, Sun Yat-sen University in 2016. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include VLSI design for manufacturing, physical design and machine learning on chips. He received Best Student Paper Award from ICTAI'2019, Best Paper Award Nomination from ASPDAC'2019, and Best Poster Research Award from Student Research Forum of ASPDAC'2020.

Qi Sun received his B.Sc. degree in computer science from Xidian University in 2018. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His current research interests include Deep Neural Network hardware acceleration, high-level synthesis, and design space exploration.

Lu Zhang received her B.Sc. degree in microelectronics from Fudan University in 2018. She is currently pursuing her Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. Her current research interest is machine learning for EDA.

Yibo Lin (S'16–M'19) received the B.S. degree in microelectronics from Shanghai Jiaotong University in 2013, and his Ph.D. degree from the Electrical and Computer Engineering Department of the University of Texas at Austin in 2018. He is current an assistant professor in the Computer Science Department associated with the Center for Energy-Efficient Computing and Applications at Peking University, China. His research interests include physical design, machine learning applications, GPU acceleration, and hardware security. He has received 4 Best Paper Awards at premier venues (ISPD 2020, DAC 2019, VLSI Integration 2018, and SPIE 2016). He has also served in the Technical Program Committees of many major conferences, including ICCAD, ICCD, ISPD, and DAC.

Iris Hui-Ru Jiang (M'07–SM'17) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1995 and 2002, respectively. She is currently a Professor with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan. She is currently an associate editor of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems and has served on technical program committees of major EDA conferences, including DAC, ICCAD, ISPD, ASP-DAC, SLIP, and IWLS. She received the Best Paper Award from ASP-DAC 2020, the Best Paper Award nomination from DAC 2016 and ISPD 2013, the Best in-track Paper from ICCAD 2014, and the First Place Award from TAU Timing Analysis Contest (four times)..

Bei Yu (S'11–M'14) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Assistant Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is Editor of IEEE TCCPS Newsletter. He received six Best Paper Awards from ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, ICCAD 2013, ASPDAC 2012, and six ICCAD/ISPD contest awards.

David Z. Pan (S'97–M'00—SM'06—F'14) received his BS degree from Peking University and MS/PhD degrees from UCLA. He is currently Silicon Laboratories Endowed Chair Professor at the Department of Electrical and Computer Engineering, The University of Texas at Austin. His research interests include cross-layer IC design for manufacturing, reliability, security, machine learning in EDA, design/CAD for analog/mixed signal designs and emerging technologies. He has published over 390 refereed journal/conference papers and 8 US patents. He has served in many journal editorial boards and conference committees, including various leadership roles. He has received many awards, including SRC Technical Excellence Award, 19 Best Paper Awards, DAC Top 10 Author Award in Fifth Decade, ASP-DAC Frequently Cited Author Award, Communications of ACM Research Highlights, ACM/SIGDA Outstanding New Faculty Award, NSF CAREER Award, IBM Faculty Award (4 times), UCLA Engineering Distinguished Young Alumnus Award, UT Austin RAISE Faculty Excellence Award, etc. He is a Fellow of IEEE and SPIE.