The Dissertation Committee for Bei Yu
certifies that this is the approved version of the following dissertation:

Design for Manufacturing with Advanced Lithography

Committee:

David Z. Pan, Supervisor

Ross Baldick

Ray T. Chen

Kevin D. Lucas

Michael Orshansky

Nur A. Touba

Design for Manufacturing with Advanced Lithography

by

Bei Yu, B.S.; M.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2014

To Le and Mandy

Acknowledgments

I am heartily thankful to my adviser, Professor David Z. Pan, for his guidance and support during the years. He is a wise leader to encourage me to pursue important research problems, and a great mentor to solve these challenging problems together. In addition, he is a dear friend who teaches me to pick up my courage to pursue my dreams. His warm advice and continuous support have helped me balance my research and life.

My sincere thanks go to other committee members. In particular, I would like to thank Professor Ross Baldick for his technical suggestions and comments on the optimization formulations. I would like to thank Professor Ray Chen for his comments on future applications of my research. I would like to thank Professor Michael Orshansky for his technical suggestions during the development of this dissertation. I would like to thank Professor Nur A. Touba for his kindness and support to this dissertation. I would like to thank Dr. Kevin Lucas for the great insights and helpful comments to patterning techniques during the years.

Besides, I really appreciate Dr. Salim Chowdhury, Dr. Akshay Sharma and Dr. Rajendran Panda for their active and valuable guidance in my internship at Oracle. My sincere thanks also go to other co-authors or industry liaison: Dr. Charles Alpert at Cadence, Dr. Gilda Garreton at Oracle Lab, Dr. Lars Liebmann at IBM, Dr. Zhuo Li at Cadence, and Dr. Gerard Luk-Pat at Synopsys. This work would not exist without their helpful suggestions.

I am lucky to have been working with the colleagues at UTDA group: Yongchan (James) Ban, Ashutosh Chakraborty, Duo Ding, Che-Lun Hsu, Jerrica Gao, Tsung-Wei Huang, Wooyoung Jang, Yang Li, Yen-Hung Lin, Yibo Lin, Tetsuaki Matsunawa, Joydeep Mitra, Ou He, Jiaojiao Ou, Jiwoo Pak, Subhendu Roy, Samuel Ward, Xiaoqing Xu, Jae-Seok Yang, Kun Yuan, Wen Zhang, Boyang Zhang, and Yilin Zhang. Only through those inspiring discussions and productive collaborations that this dissertation could be developed and polished.

Last but not least, I am deeply thankful to my wife, Le Qin, for her love and encouragement to pursue the dreams in my academic career. Another thank you should go to my dearest daughter, Mandy Yu, who brings a smile to my face and joy to my heart each and every day. Without the support and sacrifice of my family, this dissertation would not have been possible.

Portions of this work were supported by: SRC, NSF, NSFC, Oracle, Toshiba, and IBM Scholarship.

Design for Manufacturing with Advanced Lithography

Publication No. ____________________

Bei Yu, Ph.D.
The University of Texas at Austin, 2014

Supervisor: David Z. Pan

Shrinking the feature size of very large scale integrated circuits (VLSI) with advanced lithography has been a holy grail for the semiconductor industry. However, the gap between manufacturing capability and the expectation of design performance becomes critically challenged in sub-16nm technology nodes. To bridge this gap, design for manufacturing (DFM) is a must to co-optimize both design and lithography process at the same time. DFM for advanced lithography could be defined very differently under different circumstances. In general, progress in advanced lithography happens along three different directions: (1) **new patterning technique** (e.g., layout decomposition for different patterning techniques); (2) **new design methodology** (e.g., lithography aware standard cell design and physical design); (3) **new illumination system** (e.g., layout fracturing for EBL system, stencil planning for EBL system).

In this dissertation, we present our research results on design for manufacturing (DFM) with multiple patterning lithography (MPL) and electron beam lithography (EBL) addressing these three DFM research directions in advanced lithography.

For the research direction of new patterning technique, we study the layout decomposition problems for different patterning technique and explore four important topics: (1) layout decomposition for triple patterning; (2) density balanced layout decomposition for triple patterning; (3) layout decomposition for triple patterning with end-cutting; (4) layout decomposition for quadruple patterning and beyond. We present the proof that triple patterning layout decomposition is NP-hard. Besides, we propose a number of CAD optimization and integration techniques to solve different problems.

For the research direction of new design methodology, we will show the limitation of traditional design flow. That is, ignoring triple patterning lithography (TPL) in early stages may limit the potential to resolve all the TPL conflicts. We propose a coherent framework, including standard cell compliance and detailed placement, to enable TPL friendly design. Considering TPL constraints during early design stages, such as standard cell compliance, improves the layout decomposability. With the pre-coloring solutions of standard cells, we present a TPL aware detailed placement where the layout decomposition and placement can be resolved simultaneously. In addition, we propose a linear dynamic programming to solve TPL aware detailed placement with maximum displacement, which can achieve good trade-off in terms of runtime and performance.

For the EBL illumination system, we focus on two topics to improve the throughput of the whole EBL system: (1) overlapping aware stencil planning under MCC system; (2) L-shape based layout fracturing for mask preparation.

With simulations and experiments, we demonstrate the critical role and effectiveness of DFM techniques for the advanced lithography, as the semiconductor industry marches forward in the deeper sub-micron domain.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Shrinking the feature size of very large scale integrated circuits (VLSI) with advanced lithography has been a holy grail for the semiconductor industry. However, the gap between manufacturing capability and the expectation of design performance becomes critical challengs for sub-32nm technology nodes [78, 102]. Before addressing these challenges, we introduce some preliminaries of the current main-stream lithography system.

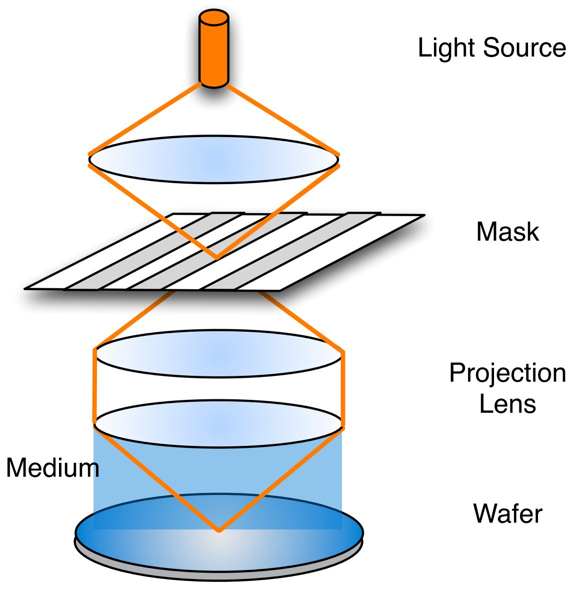


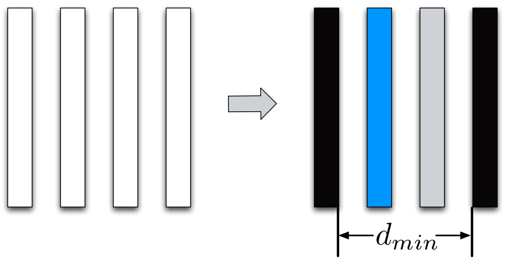
Figure 1.1: Schematic diagram of conventional lithography system.

As illustrated in Fig. 1.1, a conventional lithography system consists of four basic components: light source, mask, projection lens, and wafer. The high energy laser source sheds light on the mask and exposes the wafer through an extremely complex combination of projection lens. In the conventional lithography system, the resolution (R) is represented as follows [67]:

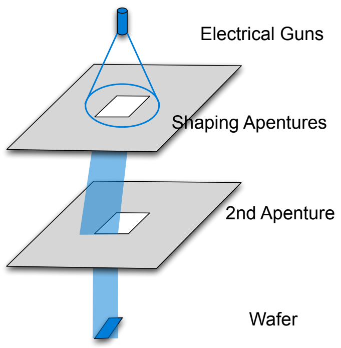
$$R = k_1 \times \frac{\lambda}{NA} \tag{1.1}$$

where λ is the wavelength of the light source (currently 193nm), k_1 is process-related parameter, and NA is the numerical aperture. For smaller feature sizes (smaller R), we need smaller k_1 and larger NA. The theoretical limitation of k_1 is 0.25 with intensive optical proximity correction (OPC) [97]. The NA can be enhanced from 0.93 to 1.35 using a technique called *immersion lithography*, where water is used as the medium between the lens to wafer. But it is hard to find new liquid material to get more than 1.35 NA value in the near future [60]. Therefore, the current optical lithography system is reaching its fundamental limit and severe variations are observed on the wafer at sub-32nm technology nodes. Due to these severe variations, the conventional lithography is no longer capable for emerging technology nodes and a set of advanced lithography techniques are called for help.

In emerging technology node and the near future, multiple patterning lithography (MPL) has become the most viable lithography technique. As shown in Fig. 1.2 (a), in MPL the original layout design is divided into several masks. Then each mask is implemented through one exposure-etch step,



(a)



(b)

Figure 1.2: Advanced lithography technique candidates: (a) multiple patterning lithography (MPL); (b) electron beam lithography (EBL).

through which the layout can be produced. Generally speaking, MPL consists of double patterning lithography (DPL), triple patterning lithography (TPL), or even quadruple patterning lithography (QPL) [48, 106, 109]. There are two main types of DPL with different manufacturing processes: litho-etch-litho-etch (LELE) [48] and self-aligned double patterning (SADP) [114]. The advantage of MPL is that the effective pitch improves which can further enhance lithography resolution [64]. Thus DPL has been heavily developed by industry for 22nm technology node, while triple patterning or quadruple patterning has been explored in industry test-chip designs [16].

In the longer future (for the logic node beyond 14nm), electron beam lithography (EBL) is a promising advanced lithography technique, along with other candidates, e.g., extreme ultra violet (EUV), directed self-assembly (DSA), and nanoimprint lithography (NIL) [78]. As shown in Fig. 1.2 (b), EBL is a maskless technology that shoots desired patterns directly into the silicon wafer using charged particle beam [77]. EBL has been widely deployed in the mask manufacturing, which is a significant step affecting the fidelity of the printed image on the wafer and critical dimension (CD) control. In addition, due to the capability of accurate pattern generation, EBL system has been developed for several decades [79]. Compared with the traditional lithographic system, EBL has several advantages. (1) Electron beam can be easily focused into nanometer diameter with charged particle beam, which can avoid the diffraction limitation of light. (2) The price of a photomask set is getting unaffordable, especially through the emerging MPL techniques. As a maskless technology, EBL can reduce the manufacturing cost. (3) EBL allows flexibility for fast turnaround times and even late design modifications to correct or adapt a given chip layout. Because of all these advantages, EBL is being used in mask making, small volume LSI production, and R&D to develop the technological nodes ahead of mass production.

1.1 Advanced Lithography Challenges

Challenges for new patterning technique: The key challenge of MPL is the new design problem, called *layout decomposition*, where input layout is divided into three masks. When the distance between two input features is less than minimum coloring distance min_s, they need to be assigned to different masks to avoid a coloring conflict. Sometimes coloring conflict can be also resolved by inserting stitch to split a pattern into two touching parts. However, this introduces stitches, which lead to yield loss because of overlay

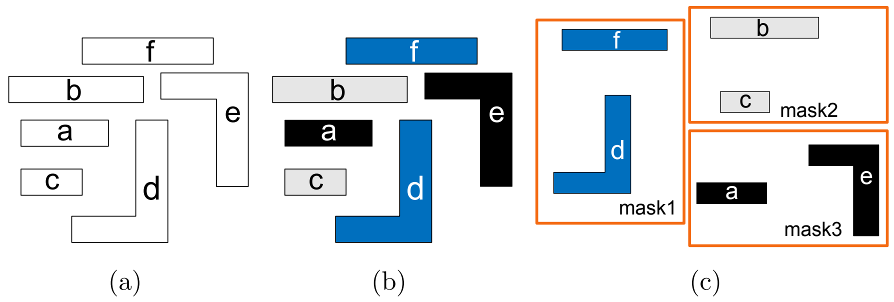


Figure 1.3: An example of triple patterning layout decomposition. (a) Input layout; (b) Features with different colors mean that they are assigned into different masks; (c) Features on each mask.

error. Therefore, two of the main objectives in layout decomposition are conflict minimization and stitch minimization. An example of triple patterning layout decomposition is shown in Fig. 1.3, where all features in input layout are divided into three masks (colors).

Chanllenges for new design methodology: With widening manufacturing gap, even the most advanced resolution enhancement techniques still cannot guarantee lithography-friendly design. Therefore, increasing cooperation of physical design is a must.

Challenges for EBL illumination system: The conventional type of EBL system is *variable shaped beam* (VSB). As illustrated in Fig. 1.2 (b), in VSB mode the layout is decomposed into a set of rectangles, and each rectangle would be shot into resist by dose of electron sequentially. The whole processing time of EBL system increases with number of beam shots. Even with decades of development, the key limitation of the EBL system has been and still is the low throughput [113]. Therefore, how to improve the throughput of EBL system is an open question.

Note that although other advanced lithography techniques are not discussed in this dissertation, all of them suffer from different technical barriers. For instance, extreme ultra violet (EUV) is challenged by issues like lack of power sources, resists, and defect-free masks [11, 95]. Directed self-assembly (DSA) technique is to phase block copolymers to construct nanostructure, but currently it has only the potential to generate contact or via layer [18].

1.2 Overview of this Dissertation

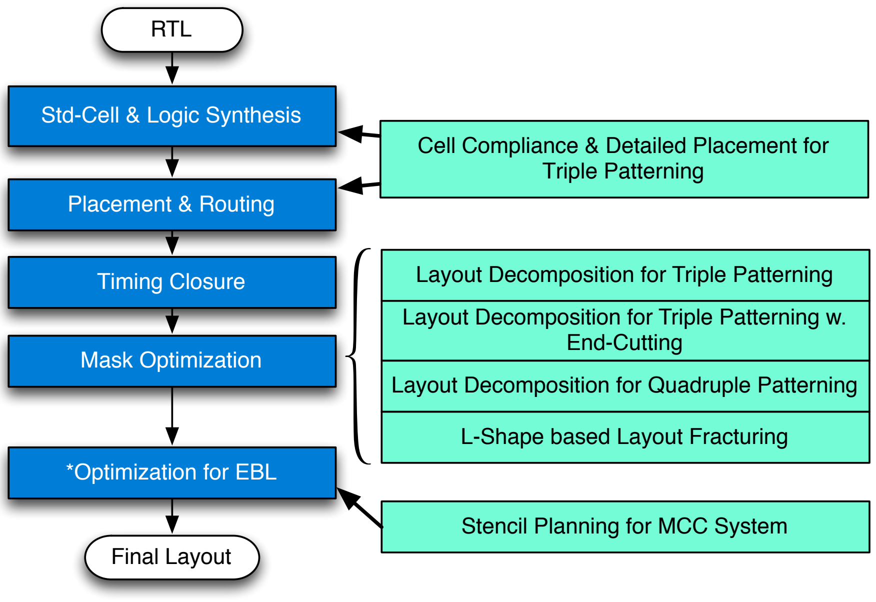

RTL
Std-Cell & Logic Synthesis
Placement & Routing
Timing Closure
Mask Optimization
*Optimization for EBL
Final Layout
Cell Compliance & Detailed Placement for Triple Patterning
Layout Decomposition for Triple Patterning
Layout Decomposition for Triple Patterning w. End-Cutting
Layout Decomposition for Quadruple Patterning
L-Shape based Layout Fracturing
Stencil Planning for MCC System

Figure 1.4: The proposed DFM techniques in their corresponding design stages.

In this dissertation, we present our research results on design for manufacturing (DFM) for MPL and EBL [102–109]. Fig. 1.4 shows the typical design flow and our proposed research works in the corresponding design stages. The goal of this dissertation is to resolve three DFM challenges in advanced lithography: new patterning technique, new design methodology, and new EBL system.

In Chapter 2 we discuss our solutions to the challenges of patterning techniques. We focus on four research topics: (1) layout decomposition for triple patterning; (2) density balanced layout decomposition for triple patterning; (3) layout decomposition for triple patterning with end-cutting; (4) layout decomposition for quadruple patterning and beyond. We present the proof that triple patterning layout decomposition is NP-hard. Besides, we propose a number of CAD optimization and integration techniques to solve each of these problems including: (a) integer linear programming (ILP) formulation to search optimal solution; (b) effective graph based simplification techniques to reduce the problem size; (c) novel semidefinite programming (SDP) based algorithms to achieve further balance in terms of runtime and solution quality; (d) linear runtime heuristic algorithms for extremely fast CPU run-time.

In Chapter 3 we resolve a coherent solution to overcome the challenges in new design methodology. We will show the limitation of traditional design flow. That is, ignoring triple patterning lithography (TPL) in early stages may limit the potential to resolve all the TPL conflicts. We propose a coherent framework, including standard cell compliance and detailed placement, to enable TPL friendly design. Considering TPL constraints during early design stages, such as standard cell compliance, improves the layout decomposability. With the pre-coloring solutions of standard cells, we present a TPL aware detailed placement where the layout decomposition and placement can be resolved simultaneously. In addition, we propose a linear dynamic programming to solve TPL aware detailed placement with maximum displacement, which can achieve good trade-off in terms of runtime and performance.

In Chapter 4 we focus on two topics to improve the throughput of the EBL system: (1) overlapping aware stencil planning under multi-column cell (MCC) system; (2) L-shape based layout fracturing for mask preparation.

We will summarize and conclude this dissertation in Chapter 5.

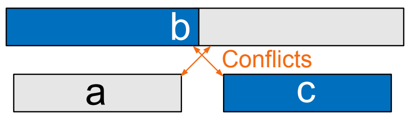
Chapter 2

Layout Decomposition for Triple/Multiple Patterning

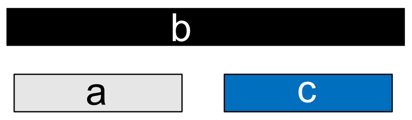
2.1 Introduction

Layout decomposition is a key challenge for multiple patterning lithography (MPL). When the distance between two input features is less than minimum coloring distance min_s, they need to be assigned to different masks to avoid a coloring conflict. Sometimes coloring conflict can be also resolved by inserting stitch to split a pattern into two touching parts. However this introduces stitches, which lead to yield loss because of overlay error. Therefore, two of the main objectives in layout decomposition are conflict minimization and stitch minimization.

In double patterning lithography, layout decomposition is generally regarded as a 2-coloring problem [49, 89, 98–100, 112]. A complete flow was proposed in [49] to optimize splitting locations with integer linear programming (ILP). Xu et al. [98] provided an efficient graph reduction-based algorithm for stitch minimization. [89, 100] proposed min-cut based approaches to reduce stitch number. To enable simultaneous conflict and stitch minimization, ILP was adopted [49] [112] with different feature pre-slicing techniques. A match-



(a)



(b)

Figure 2.1: (a) In double patterning, even stitch insertion can not avoid the native conflicts. (b) Native conflict in double patterning might be resolved by triple patterning.

ing based decomposer was proposed to minimize both the conflict number and the stitch number [99].

It shall be noted that for double patterning, even with stitch insertion, there may be some native conflicts [9]. Fig. 2.1 (a) illustrates a three-way conflict cycle between features a, b and c, where any two of them are within the minimum coloring distance. As a consequence, there is no chance to produce a conflict-free solution with double patterning. However, in triple patterning we can easily resolve this conflict, as shown in Fig. 2.1 (b). Yet this does not mean layout decomposition in triple patterning becomes easier. Actually since the features can be packed closer, the problem turns out to be more difficult [109].

There are investigations on triple patterning aware design [62, 65, 107] and triple patterning layout decomposition [21,23,31,32,56,91,92,105,109,115]. Cork et al. [23] proposed a three coloring algorithm adopting SAT formulation. [38] reused the double patterning techniques. [21,31,56,115] proposed different heuristic methods for the TPLD problem. For row based layout design, [91,92] presented polynomial time decomposition algorithms.

2.2 Layout Decomposition for Triple Patterning

2.2.1 Preliminaries and Problem Formulation

In this subsection we provide some preliminaries on triple patterning layout decomposition, including some definitions, problem formulation, and the introduction to our decomposition flow.

2.2.1.1 Layout Graph and Decomposition Graph

Given an input layout which is specified by features in polygonal shapes, at first a *layout graph* [49] is constructed by Definition 1.

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

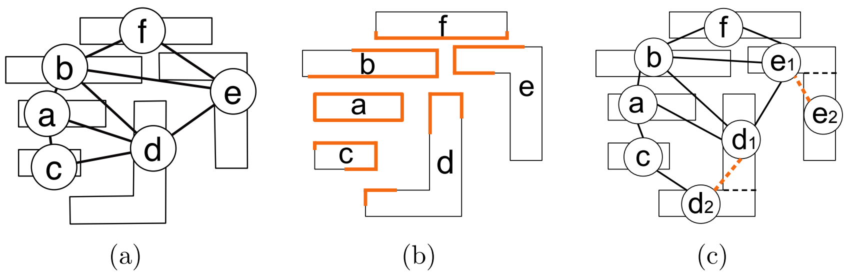


Figure 2.2: Layout graph construction and decomposition graph construction. (a) Layout graph for given input, where all edges are conflict edges; (b) The vertex projection; (c) Corresponding decomposition graph, where dash edges are stitch edges.

One example of layout graph is illustrated in Fig. 2.2 (a). All the edges in a layout graph are called conflict edges. A conflict exists if and only if two vertices are connected by a conflict edge and are in the same mask. In other words, each conflict edge is a conflict candidate. On the layout graph, vertex projection [49] is performed, where projected segments are highlighted by bold lines in Fig. 2.2 (b). Based on the projection result, all the legal splitting locations are computed. Then a *decomposition graph* [110] is constructed by Definition 2.

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

An example of decomposition graph (DG) is shown in Fig. 2.2 (c). Note that the conflict edges are marked as black edges, while stitch edges are marked as dash edges. Here each stitch edge is a stitch candidate.

2.2.1.2 Stitch Candidate Generation

Stitch candidate generation is one of the most important steps to parse a layout, as it not only determines the vertex number in the decomposition graph, but also affects the decomposition result. We use *DP candidates* to represent the stitch candidates generated by all previous double patterning research. [49, 98] propose different methodologies to generate the DP candidates. In this section, we show that DP candidates may be redundant or lose some useful candidates, and they cannot be directly applied in TPLD problem. Therefore, we provide a procedure to generate appropriate stitch candidates for triple patterning lithography.

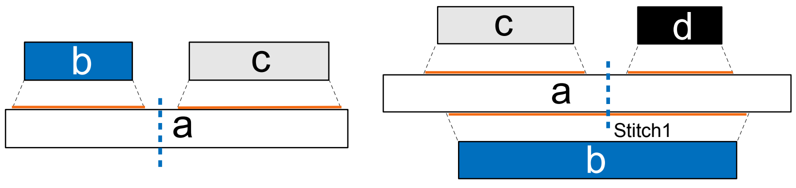


Figure 2.3: Examples of (a) Redundant stitch; (b) Lost stitch.

We provide two examples to demonstrate that DP candidates are not appropriate for triple patterning. First, because of an extra color choice, some DP candidates may be redundant. As shown in Fig. 2.3 (a), the stitch can be removed because no matter what color is assigned to features b and c, the feature a can always be assigned a legal color. We denote this kind of stitch as a *redundant stitch*. After removing these redundant stitches, some extra vertices in the decomposition graph can be merged. In this way, we can reduce the problem size. Besides, DP candidates may cause the stitch loss problem, i.e., some useful stitch candidates cannot be detected and inserted in layout decomposition. In DPL, the stitch candidate has one precondition: it cannot intersect with any projection. For example, as shown in Fig. 2.3 (b), because this stitch intersects with the projection of feature b, it cannot belong to the DP candidates. However, if features b, c and d are assigned with three different colors, only introducing this stitch can resolve the conflict. In other words, the precondition in DPL limits the ability of stitches to resolve the triple patterning conflicts and may result in unnoticed conflicts. We denote the useful stitches forbidden by the DPL precondition as a *lost stitch*.

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

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

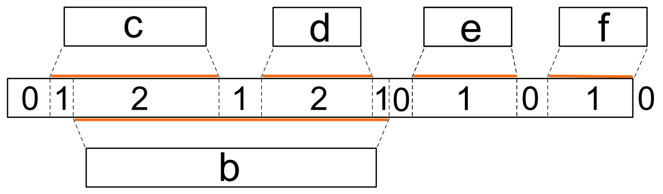


Figure 2.4: The projection sequence of the feature is 01212101010, and the last 0 is a default terminal zero.

Instead of analyzing each feature and all its neighboring features, we can directly carry out stitch candidate generation based on the projection sequence. For convenience, we provide a terminal zero rule, i.e., the beginning and the end of the projection sequence must be 0. To maintain this rule, sometimes a default 0 needs to be added. An example of projection sequence is shown in Fig. 2.4, where the middle feature has five conflict features, b, c, d, e and f. Based on the projection results, the feature is divided into ten segments. Through labeling each segment, we can get its projection sequence: 01212101010. Here a default 0 is added at the end of the feature.

Based on the definition of projection sequence, we summarize the rules for redundant stitches and lost stitches. First, motivated by the case in Fig. 2.3 (a), we can summarize the redundant stitches as follows: if the projection sequence begins with "01010", then the first stitch in DP candidates is redundant. Since the projection of a feature can be symmetric, if the projection sequence ends with "01010", then the last stitch candidate is also redundant. Besides, the rule for lost stitches is as follows, if a projection sequence contains the sub-sequence "xyz", where $x, y, z > 0$ and $x > y, z > y$, then there is one lost stitch at the segment labeled as y. For example, the stitch candidate in Fig. 2.3 (b) is contained in the sub-sequence "212", so it is a lost stitch.

The details of stitch candidate generation for TPL are shown in Algorithm 1. If necessary, at first each multiple-pin feature is decomposed into several two-pin features. Then for each feature, we can calculate its projection sequence. We remove the redundant stitches by checking if the projection sequence begins or ends with "01010". Next we search for and insert stitches, including the lost stitches. Here we define a *sequence bunch.* A sequence bunch is a sub-sequence of a projection sequence, and contains at least three non-0 segments.

An example of the stitch candidate generation is shown in Fig. 2.5. In

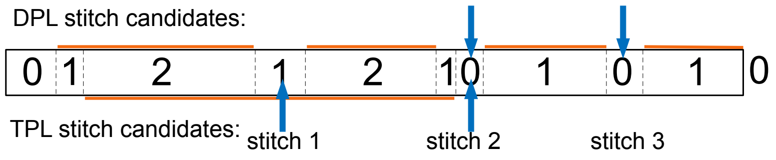


Figure 2.5: Stitch candidates generated for DPL and TPL.

Algorithm 1 Stitch Candidate Generation for TPL

```
Require: Projection results on features.
1: Decompose multiple-pin features;
2: for each feature $w_i$ do
3:     Calculate the projection sequence $ps_i$;
4:     if $ps_i$ begins or ends with "01010" then
5:         Remove redundant stitch(es);
6:     end if
7:     for each sequence bunch of $ps_i$ do
8:         Search and insert at most one stitch candidate;
9:     end for
10: end for
```

the DP candidate generation, there are two stitch candidates generated (stitch 2 and stitch 3). Through our stitch candidate generation, stitch 3 is labeled as a redundant stitch. Besides, stitch 1 is identified as a lost stitch candidate because it is located in a sub-sequence “212”. Therefore, stitch 1 and stitch 2 are chosen as stitch candidates for TPL.

2.2.1.3 Problem Formulation

We define the *triple patterning layout decomposition* (TPLD) problem as follows.

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

In our work we set the cost for each conflict is 1, and the cost for each stitch is α.

TPLD problem is an extension of double patterning layout decomposition (DPLD) problem, and both of them simultaneously minimize the conflict number and the stitch number. For DPLD problem, Xu et al. [99] showed that if the decomposition graph is planar, it can be resolved in polynomial time. At first glance, compared with DPLD, TPLD seems easier as there is now one more color (mask). However, it turns out to harder. On one hand, since the goal of triple patterning is to achieve finer pitches, there will actually be more features to be packed closer to each other which will form a multi-way conflict. In other words, decomposition graphs for triple patterning will become much denser than those in double patterning. On the other hand, in double patterning the conflict detection (2-colorable) is equivalent to odd-cycles checking, which can be resolved in linear time through a breadth-first search. However in triple patterning the conflict minimization, or even the conflict detection, is not straightforward. A planar graph 3-coloring (**PG3C**) problem is to assign three colors to all vertices of a planar graph. A conflict exists if and only if two vertices connected by an edge are in the color. The target of PG3C problem is to minimize the coloring conflict number. We have the following lemma:

Lemma 1. *The PG3C problem is NP-hard.*

The correctness of Lemma 1 stems from the conclusion that deciding whether a planar graph is 3-colorable is NP-complete [37]. For a planar graph, checking whether it is 3-colorable cannot be finished in ploynominal time; therefore, 3-coloring a planar graph with minimum cost cannot be finished in ploynominal time.

Theorem 1. ***TPLD problem is NP-hard.***

Proof. We prove this theorem by showing PG3C $\leq_P$ TPLD, i.e., PG3C can be reduced to TPLD. Given an instance of PG3C, its planar graph $G = (V, E)$ can be transferred to an **Orthogonal Drawing** [55], where the drawing plane is subdivided by horizontal and vertical gridlines of unit spacing λ. The vertices $\in V$ are represented by rectangles that the borders of the rectangles overlap the gridlines by $\lambda/4$. The edges $\in E$ are mapped to non-overlapping paths in the gridlines. Please refer to [55] for more details regarding orthogonal drawing. We construct the corresponding TPLD instance through the following two steps: (1) The width of each path is extended to $\lambda/4$; (2) Break each path in the middle through gap with length $\lambda/8$. If we set min_s to $\lambda/8$ then we can get a TPLD instance, whose decomposition graph is isomorphic to the planar graph of PG3C instance. For example, given a PG3C instance in Fig. 2.6 (a), the corresponding orthogonal drawing and TPLD instance are illustrated in Fig. 2.6 (b) and Fig. 2.6 (c), respectively. Here no stitch candidate is introduced. Since an orthogonal drawing can be constructed in polynomial time [88], the whole reduction can be finished in polynomial time. Thus minimizing conflict number in the original PG3C instance is equal to minimizing conflict number in the constructed TPLD instance, which completes the proof. □

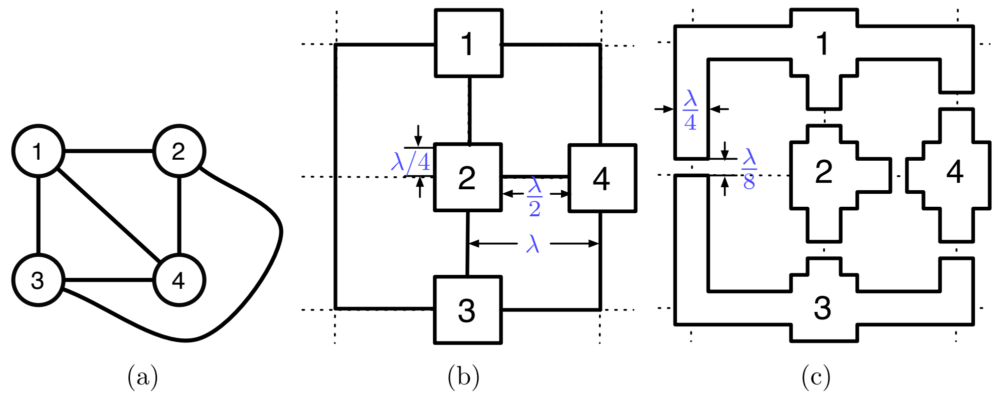


Figure 2.6: Reducing PG3C to TPLD. (a) An instance of PG3C; (b) The transferred orthogonal drawing; (c) The corresponding TPLD instance.

2.2.2 Algorithms

The overall decomposition flow is illustrated in Fig. 2.7. First, we construct layout graph to translate the original layout into graph representations. Two graph division techniques are developed to the layout graph: independent component computation (ICC) and iterative vertex removal (IVR). Second, after vertex projection, we transform the layout graph into decomposition graph and propose two other graph division methods: bridge edge detection/removal and bridge vertex detection/duplication. Third, after these graph based techniques, the decomposition graph is divided into a set of components. To solve the color assignment on each DG component, two approaches are pro-

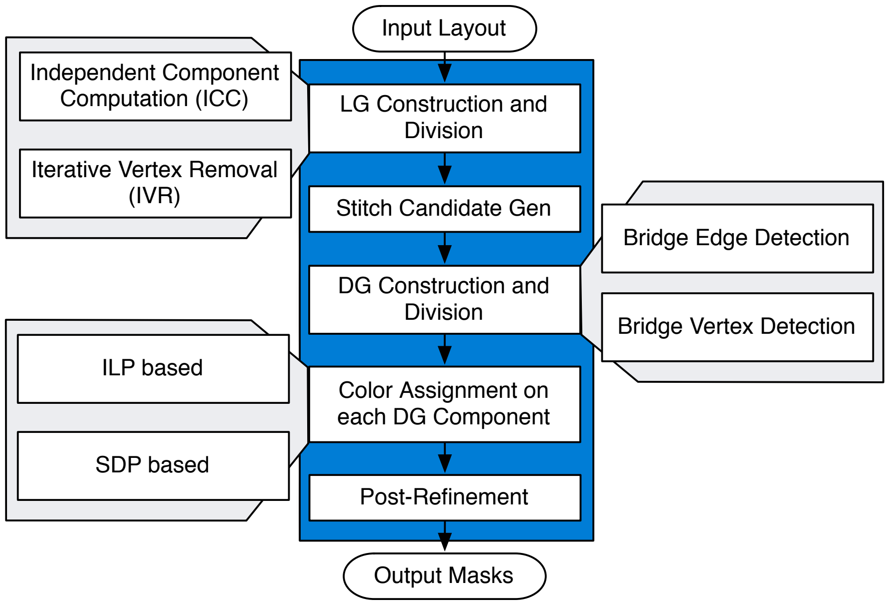

Input Layout
Independent Component Computation (ICC)
Iterative Vertex Removal (IVR)
LG Construction and Division
Stitch Candidate Gen
DG Construction and Division
Bridge Edge Detection
Bridge Vertex Detection
ILP based
SDP based
Color Assignment on each DG Component
Post-Refinement
Output Masks

Figure 2.7: Overview of our decomposition flow.

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

2.2.2.1 ILP Based Color Assignment

On decomposition graph (DG), we carry out color assignment, which is a critical step in the layout decomposition flow. In color assignment, where each vertex would be assigned one of three colors (masks). Firstly, we will

Table 2.1: Notations in ILP Formulation

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

give a general mathematical formulation for the color assignment. Then we will show that it can solved through an integer linear programming (ILP), which is commonly used before for double patterning layout decomposition (DPLD) problem [49] [112].

For convenience, some notations used in this section are listed in Table 2.1. The general mathematical formulation for the TPLD problem is shown in (2.1). The objective is to simultaneously minimize the cost of both the conflict number and the stitch number. The parameter α is a user-defined parameter for assigning relative importance between the conflict and the stitch.

In Eq. (2.1), x_i is a variable for the three colors of rectangles r_i, c_{ij} is a binary variable for conflict edge $e_{ij} \in CE$ and s_{ij} is a binary variable for stitch edge $e_{ij} \in SE$. Constraint (2.1a) is used to evaluate the conflict number

$$
\begin{aligned}
\min \quad & \sum_{e_{ij}\in CE} c_{ij} + \alpha \sum_{e_{ij}\in SE} s_{ij} & & \quad (2.1)\\
\text{s.t.} \quad & c_{ij} \leftarrow (x_i = x_j) & \forall e_{ij} \in CE & \quad (2.1a)\\
& s_{ij} \leftarrow x_i \oplus x_j & \forall e_{ij} \in SE & \quad (2.1b)\\
& x_i \in \{0,1,2\} & \forall i \in V & \quad (2.1c)
\end{aligned}
$$

when touch vertices r_i and r_j are assigned the same color (mask). Constraint (2.1b) is used to calculate the stitch number. If vertices r_i and r_j are assigned different colors (masks), stitch s_{ij} is introduced.

We will now show how to implement (2.1) with ILP. Note that eqs. (1a) and (1b) can be linearized only when x_i is a 0-1 variable [49], which is hard to represent three different colors. To handle this problem, we represent the color of each vertex using two 1-bit 0-1 variables x_{i1} and x_{i2}. In order to limit the number of colors for each vertex to 3, for each pair (x_{i1}, x_{i2}) the value $(1,1)$ is not permitted. In other words, only values $(0,0), (0,1)$ and $(1,0)$ are allowed. Thus, (2.1) can be formulated in (2.2).

The objective function is the same as that in (2.1), which minimizes the weighted summation of the conflict number and the stitch number. Constraint (2.2a) is used to limit the number of colors for each vertex to 3. In other words, only three bit-pairs $(0,0), (0,1), (1,0)$ are legal.

Constraints (2.2b) to (2.2f) are equivalent to constraint (2.1a), where 0-1 variable c_{ij1} demonstrates whether x_{i1} equals to x_{j1}, and c_{ij2} demonstrates whether x_{i2} equals to x_{j2}. 0-1 variable c_{ij} is true only if two vertices connected

$$
\begin{aligned}
\min \; & \sum_{e_{ij}\in CE} c_{ij} + \alpha \sum_{e_{ij}\in SE} s_{ij} & & (2.2)\\
\text{s.t.} \; & x_{i1} + x_{i2} \le 1 & & (2.2a)\\
& x_{i1} + x_{j1} \le 1 + c_{ij1} & \forall e_{ij} \in CE & \quad (2.2b)\\
& (1 - x_{i1}) + (1 - x_{j1}) \le 1 + c_{ij1} & \forall e_{ij} \in CE & \quad (2.2c)\\
& x_{i2} + x_{j2} \le 1 + c_{ij2} & \forall e_{ij} \in CE & \quad (2.2d)\\
& (1 - x_{i2}) + (1 - x_{j2}) \le 1 + c_{ij2} & \forall e_{ij} \in CE & \quad (2.2e)\\
& c_{ij1} + c_{ij2} \le 1 + c_{ij} & \forall e_{ij} \in CE & \quad (2.2f)\\
& x_{i1} - x_{j1} \le s_{ij1} & \forall e_{ij} \in SE & \quad (2.2g)\\
& x_{j1} - x_{i1} \le s_{ij1} & \forall e_{ij} \in SE & \quad (2.2h)\\
& x_{i2} - x_{j2} \le s_{ij2} & \forall e_{ij} \in SE & \quad (2.2i)\\
& x_{j2} - x_{i2} \le s_{ij2} & \forall e_{ij} \in SE & \quad (2.2j)\\
& s_{ij} \ge s_{ij1}, s_{ij} \ge s_{ij2} & \forall e_{ij} \in SE & \quad (2.2k)\\
& x_{ij} \text{is binary} & & (2.2l)
\end{aligned}
$$

by conflict edge e_{ij} are in the same color, e.g. both c_{ij1} and c_{ij2} are true.

Constraints (2.2g) to (2.2k) are equivalent to constraint (2.1b). 0-1 variable s_{ij1} demonstrates whether x_{i1} is different from x_{j1}, and s_{ij2} demonstrates whether x_{i2} is different from x_{j2}. Stitch s_{ij} is true if either s_{ij1} or s_{ij2} is true.

2.2.2.2 SDP Based Color Assignment

Although ILP formulation (2.2) can optimally solve the color assignment problem theoretically, for practical design it may suffer from runtime overhead problem. In this section we show that instead of expensive ILP, the color assignment can be also formulated as a vector programming, with three

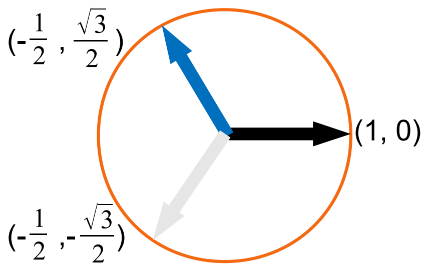

(-$\frac{1}{2}$, $\frac{\sqrt{3}}{2}$)
(1, 0)
(-$\frac{1}{2}$,-$\frac{\sqrt{3}}{2}$)

Figure 2.8: Three vectors $(1,0), (-\frac{1}{2}, \frac{\sqrt{3}}{2}), (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$ represent three different colors.

unit vectors to represent three different colors. Then the vector programming is relaxed and solved through semidefinite programming (SDP). Given the solutions of SDP, we develop a mapping process to obtain the final color assignment solutions. Note that our algorithm is fast that both SDP formulation and mapping process can be finished in polynomial time.

In color assignment, there are three possible colors. We set a unit vector $\vec{v_i}$ for every vertex i. If e_{ij} is a conflict edge, we want vertices $\vec{v_i}$ and $\vec{v_j}$ to be far apart. If e_{ij} is a stitch edge, we hope vertices $\vec{v_i}$ and $\vec{v_j}$ to be the same. As shown in Fig. 2.8, we associate all the vertices with three different unit vectors: $(1,0), (-\frac{1}{2}, \frac{\sqrt{3}}{2})$ and $(-\frac{1}{2}, -\frac{\sqrt{3}}{2})$. Note that the angle between any two vectors of the same color is 0, while the angle between vectors with different colors is $2\pi/3$.

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

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

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

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

Based on the above property, we can formulate the color assignment as the following vector program [96]:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3}(\vec{v_i} \cdot \vec{v_j} + \frac{1}{2}) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} (1 - \vec{v_i} \cdot \vec{v_j}) \tag{2.3}$$

$$\text{s.t.} \quad \vec{v_i} \in \{(1,0), (-\frac{1}{2}, \frac{\sqrt{3}}{2}), (-\frac{1}{2}, -\frac{\sqrt{3}}{2})\} \tag{2.3a}$$

Formula (2.3) is equivalent to mathematical formula (2.1): the left part is the cost of all conflicts, and the right part gives the total cost of the stitches. Since the TPLD problem is NP-hard, this vector programming is also NP-hard. In the next part, we will relax (2.3) to a semidefinite programming (SDP), which can be solved in polynomial time.

Constraint (2.3a) requires solutions of (2.3) be discrete. After removing this constraint, we generate formula (2.4) as follows:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3}(\vec{y_i} \cdot \vec{y_j} + \frac{1}{2}) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} (1 - \vec{y_i} \cdot \vec{y_j}) \tag{2.4}$$

$$\text{s.t.} \quad \vec{y_i} \cdot \vec{y_i} = 1, \quad \forall i \in V \tag{2.4a}$$

$$\vec{y_i} \cdot \vec{y_j} \geq -\frac{1}{2}, \quad \forall e_{ij} \in CE \tag{2.4b}$$

This formula is a relaxation of (2.3) since we can take any feasible solution $\vec{v_i} = (v_{i1}, v_{i2})$ to produce a feasible solution of (2.4) by setting $\vec{y_i} = (v_{i1}, v_{i2}, 0, 0, \cdots, 0)$, i.e., $\vec{y_i} \cdot \vec{y_j} = 1$ and $\vec{y_i} \cdot \vec{y_j} = \vec{v_i} \cdot \vec{v_j}$ in this solution. Here the dimension of vector $\vec{y_i}$ is $|V|$, that is, the vertex number in current DG component. If Z_R is the value of an optimal solution of formula (2.4) and OPT is an optimal value of formula (2.3), it must satisfy: $Z_R \leq OPT$. In other words, solution of (2.4) provides a lower bound approximation to that in (2.3). After removing the constant in objective function, we redraw the following vector programming.

$$\begin{aligned} &\min \sum_{e_{ij} \in CE} (\vec{y_i} \cdot \vec{y_j}) - \alpha \sum_{e_{ij} \in SE} (\vec{y_i} \cdot \vec{y_j}) \\ &\text{s.t.} \quad (2.4a) - (2.4b) \end{aligned} \tag{2.5}$$

Without discrete constraint (2.3a), programs (2.4) and (2.5) are not NP-hard now. To solve (2.5) in polynomial time, we will show that it is equivalent to a semidefinite programming (SDP). SDP is similar to LP that has a linear objective function and linear constraints. However, a square symmetric matrix of variables can be constrained to be positive semidefinite. Although semidefinite programs are more general than linear programs, both of them can be solved in polynomial time. Besides, the relaxation based on SDP has better theoretical results than those based on LP [93].

Consider the following standard SDP:

$$\begin{aligned} \text{SDP: } \min \quad & A \bullet X && (2.6) \\ & x_{ii} = 1, \quad \forall i \in V && (2.6a) \\ & x_{ij} \geq -\frac{1}{2}, \quad \forall e_{ij} \in CE && (2.6b) \\ & X \succeq 0 && (2.6c) \end{aligned}$$

where $A \bullet X$ is the inner product between two matrices A and X, i.e. $\sum_i \sum_j a_{ij} x_{ij}$. Here a_{ij} is the entry that lies in the i-th row and the j-th column of matrix A.

$$a_{ij} = \begin{cases} 1, & \forall e_{ij} \in CE \\ -\alpha, & \forall e_{ij} \in SE \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

Constraint (2.6c) means matrix X should be positive semidefinite. Similarly, x_{ij} is the i-th row and the j-th column entry of X. Note that the solution of SDP is represented as a positive semidefinite matrix X, while solutions of relaxed vector programming are stored in a list of vectors. However, we can show that they are equivalent.

Lemma 2. *A symmetric matrix X is positive semidefinite if and only if $X = VV^T$ for some matrix V.*

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

Theorem 2. ***The semidefinite program (2.6) and the vector program (2.5) are equivalent.***

Proof. Given solutions $\{\vec{y_1}, \vec{y_2}, \cdots \vec{y_m}\}$ of (2.5), the corresponding matrix X is defined as $x_{ij} = \vec{y_i} \cdot \vec{y_j}$. In the other direction, based on Lemma 2, given a matrix X from (2.6), we can find a matrix V satisfying $X = VV^T$ by using the Cholesky decomposition. The rows of V are vectors $\{v_i\}$ that form the solutions of (2.5). $\square$

After solving the SDP formulation (2.6), we get a set of continuous solutions in matrix X. Since each value x_{ij} in matrix X corresponds to $\vec{y_i} \cdot \vec{y_j}$, and $\vec{y_i} \cdot \vec{y_j}$ is an approximative solution of $\vec{v_i} \cdot \vec{v_j}$ in (2.3), we can draw the conclusion that x_{ij} is an approximation to $\vec{v_i} \cdot \vec{v_j}$. Instead of trying to calculate all $\vec{v_i}$ through Cholesky decomposition, we pay attention to x_{ij} value itself. Essentially, if x_{ij} is close to 1, then vertices i and j tend to be in the same color; if x_{ij} is close to -0.5, vertices i and j tend to be in different colors.

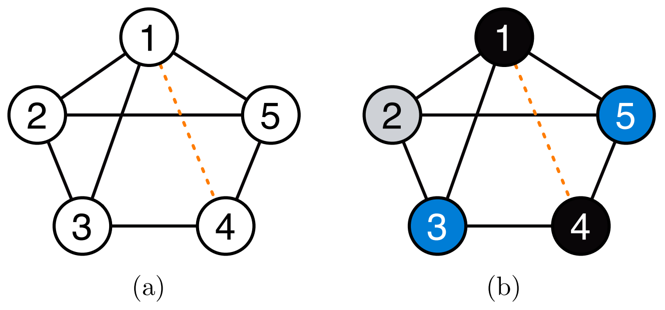


Figure 2.9: A simple example of SDP. (a) Input DG component; (b) Color assignment result with 0 conflict and 0 stitch.

For most of cases, SDP can provide reasonable solutions that each x_{ij} is either close to 1 or close to -0.5. A decomposition graph example is illustrated in Fig. 2.9. It contains seven conflict edges and one stitch edge. Moreover, the graph is not 2-colorable since it contains several odd cycles. To solve the corresponding color assignment through SDP formulation, we construct matrix A as equation (2.7) as follows:

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

Note that here we set α as 0.1. After solving the SDP (2.6), we can get a matrix X as follows:

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

Here we only list the upper part of the matrix X. Because X_{14} is 1.0, vertices 1 and 4 should be in the same color. Similarly, vertices 3 and 5 should also be in the same color. In addition, because of all other -0.5 values, we know that no more vertices can be in same color. Thus the final color assignment result for this example is shown in Fig. 2.9 (b).

The matrix X generated from Fig. 2.9 is an ideal case, that is, all values are either 1 or -0.5. Therefore from X we can derive the final color assignment easily. Our preliminary results show that with reasonable threshold such as $0.9 < x_{ij} \leq 1$ for same mask, and $-0.5 \leq x_{ij} < -0.4$ for different mask, more than 80% of vertices can be decided by the global SDP optimization. However, for practical layout, especially those essentially contain conflicts and stitches, some values in the matrix X are not so clear.

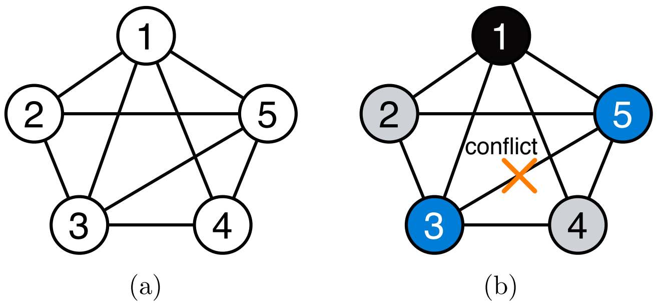


Figure 2.10: A complex example. (a) Input DG component; (b) Color assignment result with 1 conflict.

We use Fig. 2.10 for illustration. The decomposition graph in Fig. 2.10 (a) contains a 4-clique structure $\{1, 3, 4, 5\}$, therefore at least one conflict would be reported. Through solving the SDP formulation (2.6), we can achieve matrix X as in (2.8).

$$X = \begin{pmatrix} 1.0 & -0.5 & -0.13 & -0.5 & -0.13 \\ & 1.0 & -0.5 & 1.0 & -0.5 \\ & & 1.0 & -0.5 & -0.13 \\ & \dots & & 1.0 & -0.5 \\ & & & & 1.0 \end{pmatrix} \tag{2.8}$$

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

The details of the mapping are shown in Algorithm 2. Given the solutions from program (2.6), some triplets are constructed and sorted to store all x_{ij} information (lines 1–2). Then our mapping can be divided into two steps. In the first stage (lines 3–8), if x_{ij} is close to 1 or -0.5, the relationship between vertices r_i and r_j can be directly determined. Here th_{unn} and th_{sp} are user-defined threshold values, where th_{unn} should be close to 1, or th_{sp} should be close to -0.5. If $x_{ij} > th_{unn}$, which means that x_{ij} is close to 1, then we apply operation **Union**(i, j) to merge vertices r_i and r_j in a large vertex (i.e., they are in the same color). Similarly, if $x_{ij} < th_{sp}$, which means that x_{ij} is close to -0.5, then operation **Separate**(i, j) is used to label that vertices r_i and r_j are incompatible. If r_i and r_j are incompatible, or r_i is imcompatible with any vertex that is in r_j's group, vertices r_i and r_j cannot be assigned into the same color and function **Compatible**(i, j) will return $false$. In the second step (lines 9–12), we continue to union the vertices i and j with largest x_{ij} until all vertices are assigned into three colors.

We use the *disjoint-set* data structure to group vertices into three colors. Implemented with *union by rank* and *path compression*, the running time per operation of disjoint-set is almost constant [24]. Let n be the number of vertices, then the number of triplets is $O(n^2)$. Sorting all the triplets requires $O(n^2 logn)$. Since all triplets are sorted, each of them can be visited at most once. Because the runtime of each operation can be finished almost in constant time, the complexity of Algorithm 2 is $O(n^2 logn)$. Applying Algorithm 2 to the matrix in (2.8), we can get the final color assignment (see Fig. 2.10

Algorithm 2 Mapping Algorithm

```
Require: Solution matrix $X$ of the program (2.6).
 1: Label each non-zero entry $X_{i,j}$ as a triplet $(x_{ij}, i, j)$;
 2: Sort all $(x_{ij}, i, j)$ by $x_{ij}$;
 3: for all triples with $x_{ij} > th_{unn}$ do
 4:     Union(i, j);
 5: end for
 6: for all triples with $x_{ij} < th_{sp}$ do
 7:     Separate(i, j);
 8: end for
 9: while number of groups > 3 do
10:     Pick triple with maximum $x_{ij}$ and Compatible(i, j);
11:     Union (i, j);
12: end while
```

(b)), where one conflict between vertices 3 and 5 is reported.

2.2.2.3 Graph Division

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

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

The first division technique is called *independent component computation* (ICC). In a layout graph of real design, we observe many isolated clusters. By breaking down the whole layout graph into several independent components, we partition the initial layout graph into several small ones. After

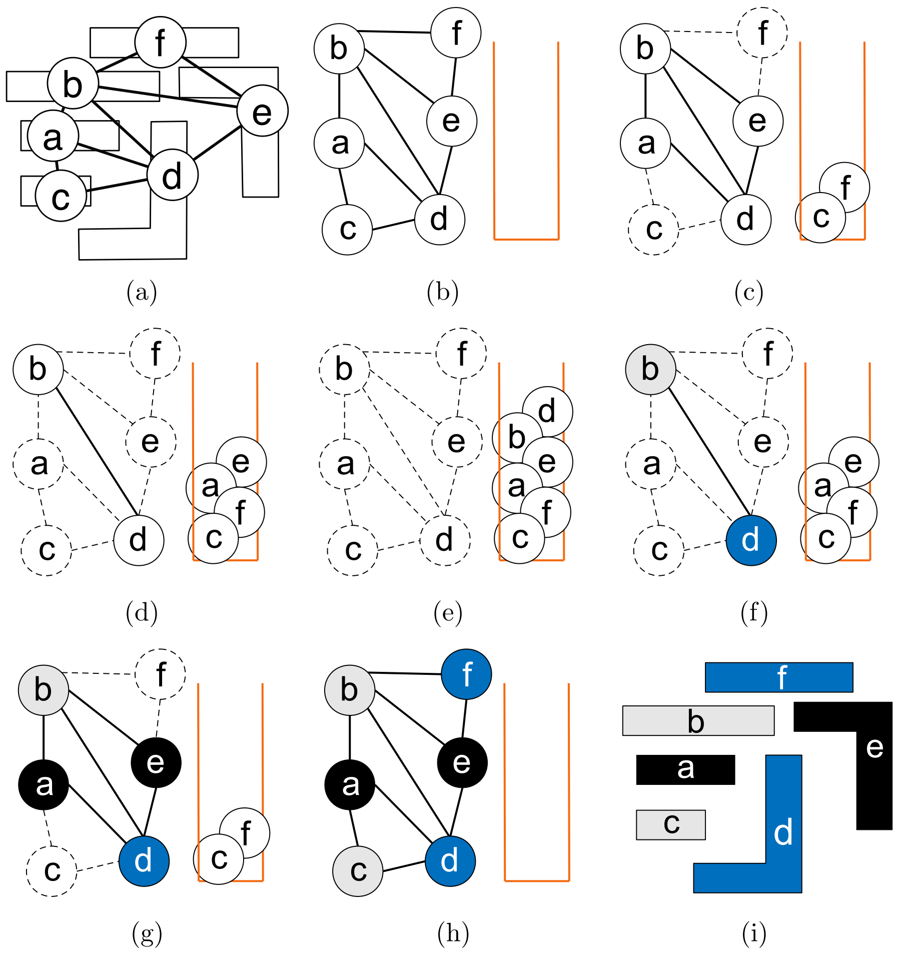


Figure 2.11: An example of iterative vertex removal (IVR), where the TPLD problem can be solved in linear time: (a) Layout graph; (b)(c)(d)(e) Iteratively remove and push in vertices with degree less than three; (f)(g)(h) After color assignment for the remanent vertices, iteratively pop up and recover vertices, and assign any legal color; (i) TPLD can be finished after the iterative vertex recover.

solving the TPLD problem for each isolated component, the overall solution can be taken as the union of all the components without affecting the global optimality. It shall be noted that ICC is a well-known technique which has been applied in many previous studies.

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

Algorithm 3 IVR and Color Assignment

```
Require: Layout graph $G$, stack $S$.
 1: while $\exists n \in G$ s.t. $degree(n) \leq 2$ do
 2:     $S.push(n)$;
 3:     $G.delete(n)$;
 4: end while
 5: Construct DG for the remanent vertices;
 6: for each component in DG do
 7:     Apply color assignment;
 8: end for
 9: while $!S.empty()$ do
10:     $n = S.pop()$;
11:     $G.add(n)$;
12:     Assign $n$ a legal color;
13: end while
```

If all the vertices in one layout graph can be temporarily removed (pushed onto the stack S), TPLD problem is solved optimally in linear time.

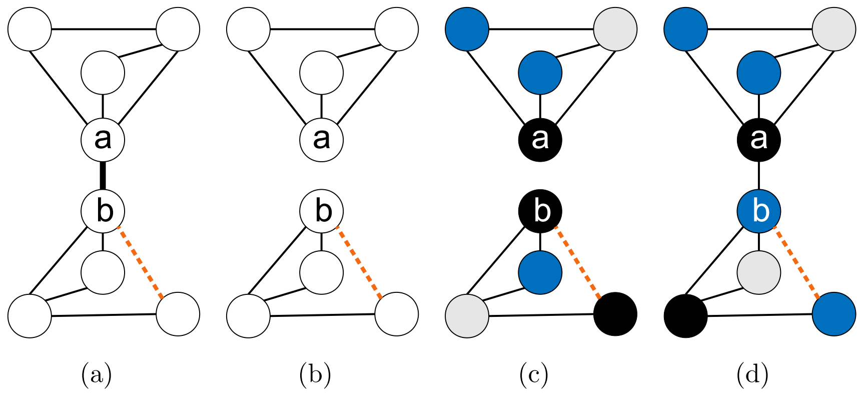


Figure 2.12: Bridge edge detection and removal. (a) Initial decomposition graph. (b) After bridge edge detection, remove edge e_{ab}. (c) In two components we carry out layout decomposition. (d) Rotate colors in the lower component to add bridge.

An example is illustrated in Fig.2.11, where all the vertices can finally be pushed onto stack. Even there are still some vertices remained, our iterative vertex removal technique can minimize problem size dramatically. Additionally, we observe that this technique can further partition the layout graph into several independent components.

On the layout graph simplified by ICC and IVR, projection is carried out to calculate all the potential stitch positions. Then we construct the decomposition graph, which include the conflict edges in the layout graph and the stitch edges. Here the stitch edges are based on the projection result. Note that ICC can be still applied here to partition a decomposition graph into several smaller ones. We further propose two new techniques to reduce the size of each decomposition graph. The first one is bridge edge detection and removal, and the second one is bridge vertex detection and duplication.

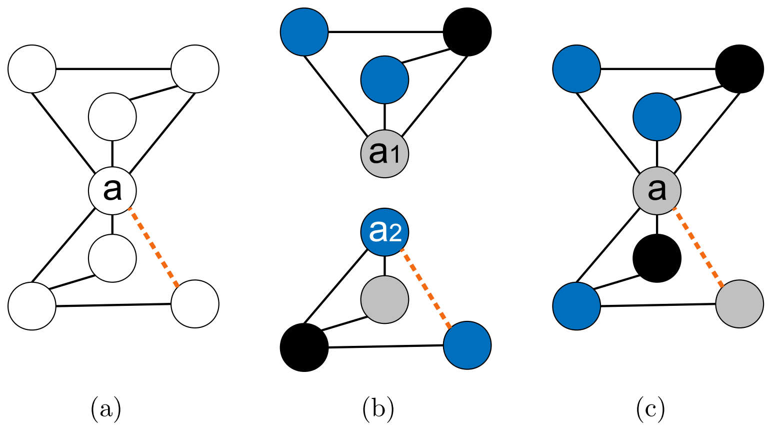


Figure 2.13: Bridge vertex detection and duplication. (a) Initial decomposition graph. (b) After bridge vertex detection, duplicate vertex a. (c) Rotate the colors in lower sub-graph to merge vertices a_1 and a_2.

A bridge edge of a graph is an edge whose removal disconnects the graph into two components. Removing the bridge edge can divide the whole problem into two independent sub-problems.

An example of the bridge edge detection is shown in Fig. 2.12. Conflict edge e_{ab} is found to be a bridge edge. Removing the bridge divides the decomposition graph into two sides. After layout decomposition for each component, if vertices a and b are assigned the same color, without loss of generality, we can rotate colors of all vertices in the lower side. Similar method can be adopted when bridge is a stitch edge. We adopt an $O(|V| + |E|)$ algorithm [90] to detect all bridge edges in decomposition graph.

A bridge vertex of a graph is a vertex whose removal disconnects the graph into two or more components. Similar to bridge edge detection, we can further simplify the decomposition graph by removing all the bridge vertices.

An example of bridge vertex computation is illustrated in Fig. 2.13. This simplification method is effective because for standard cell layouts, usually we can choose the power and ground lines as the bridge vertices. By this way we can significantly partition the layouts by rows. All bridge vertices can be detected using an $O(|V| + |E|)$ search algorithm.

2.2.2.4 Post Refinement

Although the graph division techniques can dramatically reduce the computational time to solve the TPLD problem, Kuang et al. [56] pointed out that for some cases iterative vertex removal (IVR) may loss some optimality. One example is illustrated in Fig. 2.14. The simplified layout graph (Fig. 2.14(b)) can be inserted stitch candidates and assigned legal colors (see Fig. 2.14(b)). However, when recover removed vertices, the vertex degree of a is increased to 3, and there is no available color for it (see Fig. 2.14(c)). The reason for this conflict is that during stitch candidate generation, vertex a is not considered.

We propose a post refinement to resolve the conflicts caused by iterative vertex removal. First we check whether the conflict can be cleared by splitting a, if yes, one or more stitches would be introduced to a, then stop. Otherwise, we re-calculate the color assignment on this portion, as illustrated in Fig. 2.15. The initial layout graph would be extended to include vertex a (Fig. 2.15(a)), thus the position of vertex a would be considered during stitch candidate generation. As shown in Fig. 2.15 (c), no additional conflict

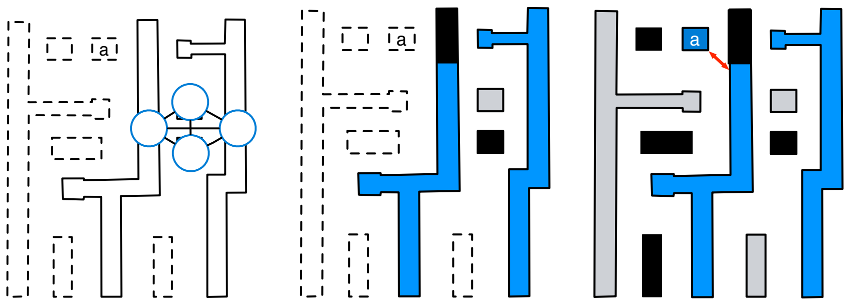


Figure 2.14: Iterative vertex removal may introduce additional conflicts. (a) Layout graph after iterative vertex removal; (b) Stitch generation and color assignment on the graph; (c) After adding back the simplified vertices, one additional conflict is introduced to vertex a.

would be introduced after recovering all vertices from stack. It shall be noted although re-solving color assignment requires more computational time, our initial results show that only small part of DG components need to apply the post-stage.

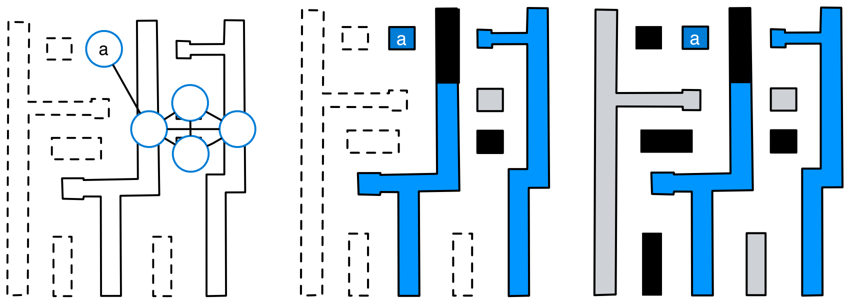


Figure 2.15: An example of post refinement. (a) Extend layout graph to include a; (b) Stitch generation and color assignment on the new graph; (c) No additional conflict at final solution.

2.2.3 Experimental Results

We implement our algorithm in C++ and test it on an Intel Core 2.9GHz Linux machine. We choose GUROBI [43] as the ILP solver, while CSDP [15] as the SDP solver. ISCAS benchmarks from [100, 112] are scaled down and modified as our test cases. The metal one layer is used for experimental purposes, because it is one of the most complex layers in terms of layout decomposition. The minimum coloring spacing min_s is set as 120 for the first ten cases and as 100 for the last five cases, as in [31, 32, 109]. Parameter α is set as 0.1, thus the decomposition cost is calculated by $cn\# + 0.1 \cdot st\#$, where $cn\#$ and $st\#$ denote the conflict number and the stitch number, respectively.

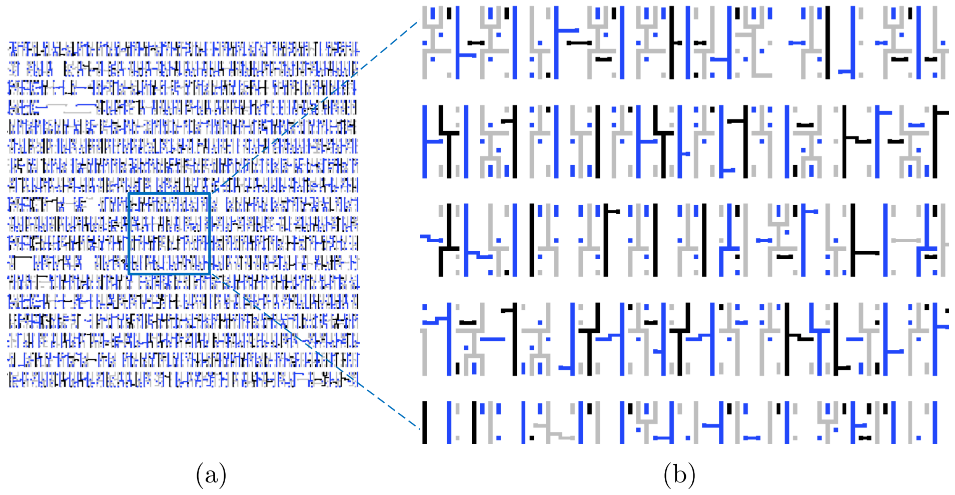


Figure 2.16: Part of S1488 decomposition result.

Fig. 2.16 illustrates part of the decomposition result for case S1488, which can be decomposed in 0.1 second.

First we demonstrate the effectiveness of our stitch candidate genera-

Table 2.2: DP Stitch v.s. TP Stitch

Circuit	ILP w/o. TP Stitch				ILP w. TP Stitch			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
C432	1	3	3.1	0.47	4	1	1.4	0.62
C499	0	0	0	0.05	0	0	0	0.24
C880	5	3	3.5	0.33	8	1	1.8	0.49
C1355	3	0	0.3	0.09	3	0	0.3	0.14
C1908	1	0	0.1	0.25	1	0	0.1	0.31
C2670	3	4	4.3	0.45	7	1	1.7	0.5
C3540	6	5	5.6	0.81	8	2	2.8	1.31
C5315	5	4	4.5	0.76	9	1	1.9	0.94
C6288	87	149	157.7	13.9	217	18	39.7	16.2
C7552	20	8	10	1.28	27	2	4.7	2.37
S1488	2	0	0.2	0.12	2	0	0.2	0.09
S38417	63	25	31.3	4.03	76	19	26.6	5.14
S35932	83	59	67.3	9.36	99	47	56.9	13.15
S38584	135	49	62.5	8.6	157	43	58.7	11.6
S15850	121	54	66.1	8.56	130	40	53	10.7
avg.	35.7	24.2	27.8	3.27	49.9	11.7	16.7	4.25
ratio	1.0	1.0	**1.0**	**1.0**	1.40	0.48	**0.60**	**1.30**

tion. Table 2.2 compares the performance and runtime of ILP on two different stitch candidates, i.e., DP stitch and TP stitch. "**ILP w/o. TP stitch**" and "**ILP w. TP stitch**" apply DP stitch and TP stitch, respectively. Note that here all graph division techniques are applied here. The columns "st#" and "cn#" denote the sttich number and the conflict number. Column "CPU(s)" is computational time in seconds. As discussed in Section 2.2.1.2, through applying TP stitch more stitch candidates would be generated, therefore we can see from Table 2.2 that 30% more runtime would be introduced. However, TP stitch overcomes the lost stitch problem in DP stitch, thus the decomposition cost is reduced by 40%. In other words, compared with DP stitch, TP stitch can provide higher performance in terms of the conflict number and the stitch number.

Secondly we show the effectiveness of the graph division, which consists of a set of techniques: independent component computation (ICC), iterative vertex removal (IVR) and other two bridge detection. Through applying these division techniques, the decomposition graph size can be reduced. Generally speaking, smaller size of decomposition graph, less runtime the ILP needs. Table 2.3 compares the performance and runtime of ILP on two different decomposition graphs. Here "**ILP w. ICC**" means the decomposition graphs are only simplified by the ICC, while "**ILP w. 4SPD**" means all the division techniques are used. Columns "TSE#" and "TCE#" denote the total stitch edge number and total conflict edge number, respectively. From Table 2.3 we can see that compared with only using ICC technique, further applying iterative vertex removal and bridges detection is more effective: the stitch edge number can be reduced by 92%, while the conflict number can be reduced by 93%. The columns "st#" and "cn#" show the stitch number and the conflict number in the final decomposition results. "CPU(s)" is computational time in seconds. Compared with the "ILP w. ICC", the "ILP w. 4SPD" can achieve the same results with much less of the runtime for some smaller cases. For some big circuits, the runtimes of "ILP w. ICC" are unacceptable, i.e., longer than two hours. Note that if no ICC technique is used, even for small circuits like C432, the runtime for ILP is unacceptable.

Table 2.3: Effectiveness of Graph Division

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

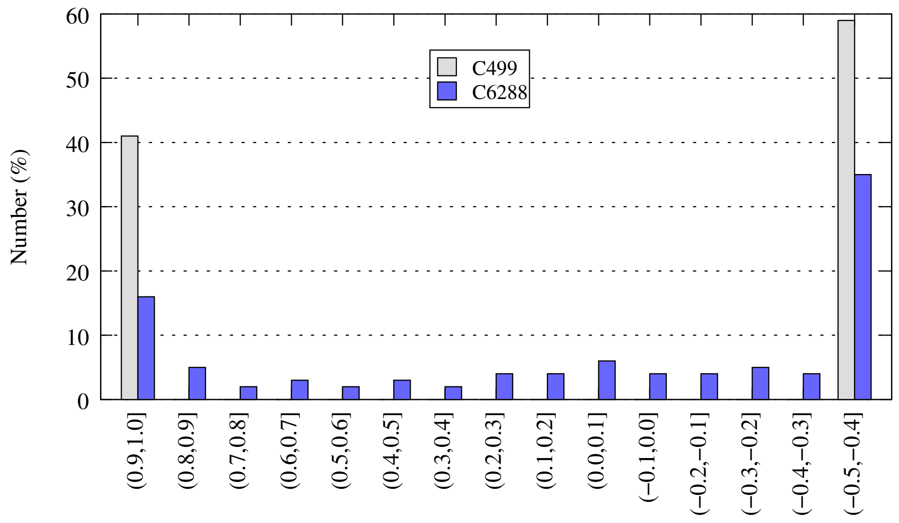


Figure 2.17: Value distribution in matrix X for cases C499 and C6288.

Here we show some more details of solutions in SDP. As discussed before, if the value X_{ij} is close to 1 or -0.5, it can be directly rounded to an integer value. Otherwise, we have to rely on some mapping methods. Fig. 2.17 illustrates the X_{ij} value distributions in circuit C499 and C6288. As we can see that all for C499, the values are either in the range of $[0.9, 1.0]$ or in $[-0.5, -0.4]$. In other words, here SDP is effective and its results can be directly used as final decomposition results. For the case C6288, since its result consists of several stitches and conflicts, some X_{ij} values are vague. But most of the values are still distinguishable.

We further demonstrate the effectiveness of the post-refinement. Table 2.4 lists the decomposition results of two SDP based algorithm, and columns "**SDP w/o. Refinement**" and "**SDP w. Refinement**" mean SDP without and with post-refinemnt, respectively. As shown in Table 2.4, by additional

Table 2.4: Effectiveness of Post-Refinement

Circuit	SDP w/o. Refinement				SDP w. Refinement			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
C432	4	1	1.4	0.25	4	0	0.4	0.25
C499	0	0	0	0.12	0	0	0	0.12
C880	8	1	1.8	0.14	8	0	0.8	0.15
C1355	3	0	0.3	0.11	3	0	0.3	0.11
C1908	1	0	0.1	0.17	1	0	0.1	0.18
C2670	7	1	1.7	0.26	8	1	1.8	0.26
C3540	9	3	3.9	0.45	9	1	1.9	0.51
C5315	9	1	1.9	0.52	9	0	0.9	0.57
C6288	213	18	39.3	3.26	210	6	27	3.42
C7552	26	1	3.6	0.93	26	0	2.6	1.04
S1488	2	0	0.2	0.1	2	0	0.2	0.11
S38417	71	19	26.1	2.32	71	19	26.1	2.38
S35932	85	46	54.5	6.3	79	45	52.9	6.6
S38584	144	43	57.4	6.36	146	36	50.6	6.52
S15850	123	43	55.3	5.77	120	36	48	6.36
avg.	47	11.8	16.5	1.804	46.4	9.6	14.24	1.91
ratio	1.0	1.0	**1.0**	**1.0**	0.99	0.81	**0.86**	**1.06**

post-refinement stage, the decomposition costs can be reduced by 14%, while only 6% of computational time is introduced.

Finally we compare our decomposition algorithms with the state-of-the-art layout decomposers [31, 32], as shown in Table 2.5. Columns "**ILP w. All**" and "**SDP w. All**" denote ILP based algorithm and SDP based algorithm, respectively. Here "w. All" means all other techniques, e.g., TP stitch, graph division, and post-refinement, are all applied. From Table 2.5 we can see that compared with SDP based methods, the decomposers [31,32] are faster, they introduce 35% more and 18% more decomposition costs. Although

Table 2.5: Comparison with other decomposers

Circuit	DAC'12 [32]				TCAD [31]				ILP w. All				SDP w. All			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
C432	6	0	0.6	0.03	6	0	0.6	0.11	4	0	0.4	0.99	4	0	0.4	0.25
C499	0	0	0	0.1	0	0	0	0.02	0	0	0	0.27	0	0	0	0.12
C880	15	1	2.5	0.11	8	1	1.8	0.12	8	0	0.8	0.5	8	0	0.8	0.15
C1355	7	1	1.7	0.11	4	1	1.4	0.14	3	0	0.3	0.18	3	0	0.3	0.11
C1908	0	1	1.0	0.07	0	1	1	0.08	1	0	0.1	0.3	1	0	0.1	0.18
C2670	12	2	3.2	0.1	11	2	3.1	0.1	8	1	1.8	0.54	8	1	1.8	0.26
C3540	14	3	4.4	0.11	11	3	4.1	0.11	8	1	1.8	1.43	9	1	1.9	0.51
C5315	11	3	4.1	0.17	11	3	4.1	0.17	9	0	0.9	1.2	9	0	0.9	0.57
C6288	342	20	54.2	0.17	243	20	44.3	0.17	210	6	27	18.4	210	6	27	3.42
C7552	46	3	7.6	0.27	37	3	6.7	0.28	26	0	2.6	2.8	26	0	2.6	1.04
S1488	4	0	0.4	0.09	4	0	0.4	0.1	2	0	0.2	0.11	2	0	0.2	0.11
S38417	122	20	32.2	0.82	82	20	28.2	0.74	71	19	26.1	5.3	71	19	26.1	2.38
S35932	103	46	56.3	2.1	63	46	52.3	2.1	79	45	52.9	13.9	79	45	52.9	6.6
S38584	280	36	64	2.3	176	36	53.6	2.3	146	36	50.6	12.1	146	36	50.6	6.52
S15850	201	36	56.1	2.04	146	36	50.6	2	126	35	47.6	11.1	120	36	48	6.36
avg.	77.5	11.5	19.22	0.57	53.5	11.5	16.8	0.57	46.7	9.53	14.21	4.61	46.4	9.6	14.24	1.91
ratio	1.67	1.19	**1.35**	**0.30**	1.15	1.19	**1.18**	**0.30**	0.99	1.01	**1.00**	**2.42**	1.0	1.0	**1.0**	**1.0**

Table 2.6: Comparisons on Very Dense Layouts

Circuit	DAC'12 [32]				TCAD [31]				ILP w. All				SDP w. All			
	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)	st#	cn#	cost	CPU(s)
c5_total	433	248	291.3	0.16	374	248	285.4	0.15	504	21	71.4	120.2	497	30	79.7	12.0
c6_total	954	522	617.4	0.3	869	521	607.9	0.26	N/A	N/A	N/A	>3600	1056	179	284.6	33.97
c7_total	1111	623	734.1	0.3	938	624	717.8	0.33	1113	283	394.3	451.4	1103	304	414.3	22.2
c8_total	1910	727	918	0.53	1671	727	894.1	0.51	1676	409	576.6	260.1	1636	433	596.6	28.4
c9_total	834	525	608.4	0.4	635	525	588.5	0.4	865	256	342.5	79.8	853	267	352.3	11.8
c10_total	2463	1144	1390.3	0.86	2120	1146	1358	0.88	2510	355	606	802.6	2472	403	650.2	50.4
avg.	1284.2	631.5	759.9	0.425	1101.2	631.8	742.0	0.42	N/A	N/A	N/A	885.7	1269.5	269.3	396.3	26.5
ratio	1.01	2.34	**1.92**	**0.02**	0.87	2.35	**1.87**	**0.02**	-	-	-	>**33.5**	1.0	1.0	**1.0**	**1.0**

ILP based algorithm has performance in terms of conflict number, it introduces the worst runtime. Compared with ILP, SDP based algorithm provides much better tradeoff between runtime and performance, i.e., it can achieve very comparable results (1% of conflict difference), but more than 3× speed-up.

In order to further evaluate the scalability of all the decomposers, we create six additional benchmarks ("c5_total" - "c10_total") to compare different algorithms on very dense layouts. Table 2.6 lists the comparison results. As we can see, compared with SDP based method, although ILP can achieve the best decomposition results, its high runtime complexity makes it impossible to solve one large dense layout, even all the graph division techniques are adopted. On the other hand, although the decomposers [31, 32] are faster that all the cases can be finished in one second, they introduce 92% more and 87% more decomposition costs. It can be observed that for some cases hundreds of additional conflicts would be reported. Each conflict may require manual layout modification or high ECO efforts, which are very time consuming. Therefore, we can see that for these dense layouts, SDP based algorithm can achieve good tradeoff in terms of runtime and performance.

2.2.4 Summary

We have shown that triple patterning layout decomposition (TPLD) problem is NP-hard, and runtime required to solve it exactly increases dramatically with the problem size. To reduce the problem size, we presented a set of graph division techniques. Then we proposed a general integer linear programming (ILP) formulation to simultaneously minimize the conflicts and stitches. Furthermore, we proposed a novel vector program, and its semidefinite programming (SDP) relaxation to improve scalability for very dense layouts. Experimental results showed that our methods are very effective.

2.3 Density Balanced Layout Decomposition for Triple Patterning

In this section, we propose a high performance layout decomposer for TPL. Compared with previous works, our decomposer provides not only less conflict and stitch number, but also more balanced density. We focus on the coloring algorithms and leave other layout related optimizations to post-coloring stages, such as compensation for various mask overlay errors introduced by scanner and mask write control processes. However, we do explicitly consider balancing density during coloring, since it is known that mask write overlay control generally benefits from improved density balance.

Our key contributions include the following. (1) Accurately integrate density balance into the mathematical formulation; (2) Develop a three-way partition based mapping, which not only achieves less conflicts, but also more balanced density; (3) Propose several techniques to speedup the layout decomposition; (4) Our experiments show the best results in solution quality while maintaining better balanced density (i.e., less EPE).

2.3.1 Preliminaries and Problem Formulation

2.3.1.1 Why Balanced Density?

In layout decomposition, especially for TPL, density balance should also be considered, along with the conflict and stitch minimization. A good pattern density balance is also expected to be a consideration in mask CD and registration control [64], while unbalanced density would cause lithography hotspots as well as lowered CD uniformity due to irregular pitches [100]. However, from the algorithmic perspective, achieving a balanced density in TPL could be harder than that in DPL. (1) In DPL, two colors can be more implicitly balanced; while in TPL, often times existing/previous strategies may try to do DPL first, and then do some "patch" with the third mask, which causes a big challenge to "explicitly" consider the density balance. (2) Due to the one more color, the solution space is much larger [23]. (3) Instead of global density balance, local density balance should be considered to reduce the potential hotspots, since neighboring patterns are one of the main sources of hotspots. As shown in Fig. 2.18 (a)(b), when only global density balance is considered, feature a is assigned white color. Since two black features are close to each other, hotspot may be introduced. To consider the local density balance, the layout is partitioned into four bins $\{b_1, b_2, b_3, b_4\}$ (see Fig. 2.18 (c)). Feature a is covered by bins b_1 and b_2, therefore it is colored as blue to maintain the local density balances for both bins (see Fig. 2.18 (d)).

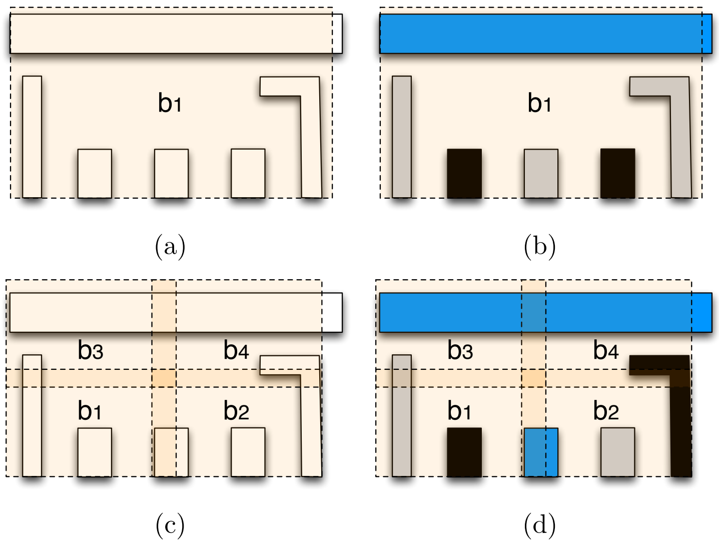


Figure 2.18: Decomposed layout with (a) (b) global balanced density. (c) (d) local balanced density in all bins.

2.3.1.2 Problem Formulation

Given input layout which is specified by features in polygonal shapes, we partition the layout into n bins $B = \{b_1, \ldots, b_n\}$. Note that neighboring bins may share some overlapping. For each polygonal feature r_i, we denote its area as den_i, and its area covered by bin b_k as den_{ki}. Clearly $den_i \geq den_{ki}$ for any bin b_k. During layout decomposition, all polygonal features are divided into three masks. For each bin b_k, we define three densities (d_{k1}, d_{k2}, d_{k3}), where $d_{kc} = \sum den_{ki}$, for any feature r_i assigned to color c. Therefore, we can define the local density uniformity as follows:

Definition 4 (Local Density Uniformity)**.** *For the bin $b_k \in S$, the local density uniformity is $max\{d_{kc}\}/min\{d_{kc}\}$ given three densities d_{k1}, d_{k2} and d_{k3} for three masks and is used to measure the ratio difference of the densities. A lower value* *means better local density balance. The local density uniformity is denoted by* DU_k.

For convenience, we use the term density uniformity to refer to local density uniformity in the rest of this section. It is easy to see that DU_k is always larger than or equal to 1. To keep a more balanced density in bin b_k, we expect DU_k as small as possible, i.e., close to 1.

Problem 2 (**Density Balanced Layout Decomposition**). *Given a layout which is specified by features in polygonal shapes, the layout graphs and the decomposition graphs are constructed. Our goal is to assign all vertices in the decomposition graph into three colors (masks) to minimize the stitch number and the conflict number, while keeping all density uniformities* DU_k *as small as possible.*

2.3.2 Algorithms

The overall flow of our density balanced TPL decomposer is illustrated in Fig. 2.19. It consists of two stages: graph construction / simplification, and color assignment. Given input layout, layout graphs and decomposition graphs are constructed, then graph simplifications [109] [32] are applied to reduce the problem size. Two additional graph simplification techniques are introduced. During stitch candidate generation, the methods described in [56] are applied to search all stitch candidates for TPL. In second stage, for each decomposition graph, color assignment is proposed to assign each vertex one

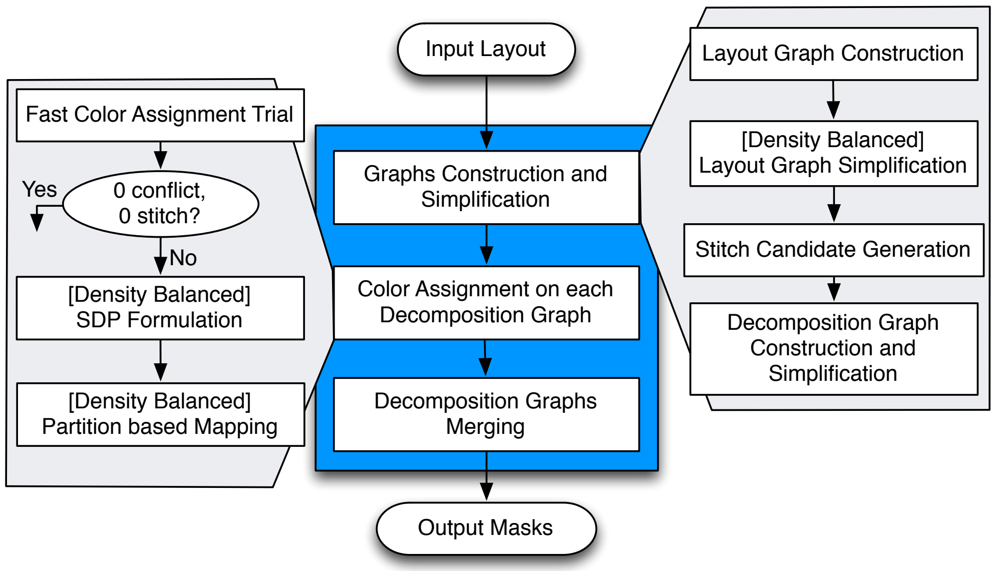

Input Layout
Fast Color Assignment Trial
Yes
0 conflict, 0 stitch?
No
[Density Balanced] SDP Formulation
[Density Balanced] Partition based Mapping
Graphs Construction and Simplification
Color Assignment on each Decomposition Graph
Decomposition Graphs Merging
Output Masks
Layout Graph Construction
[Density Balanced] Layout Graph Simplification
Stitch Candidate Generation
Decomposition Graph Construction and Simplification

Figure 2.19: Overall flow of proposed density balanced decomposer.

color. Before calling SDP formulation, fast color assignment trial is proposed to achieve better speedup (see Section 2.3.2.4).

Stitch candidate generation is one important step to parse a layout. [32] [38] pointed out that the stitch candidates generated by previous DPL works cannot be directly applied in TPL layout decomposition. Therefore, we provide a procedure to generate appropriate stitch candidates for TPL. The main idea is that after projection, each feature is divided into several segments each of which is labeled with a number representing how many other features are projected onto it. If one segment is projected by less than two features, then a stitch can be introduced. Note that to reduce the problem size, we restrict the maximum stitch candidate number on each feature.

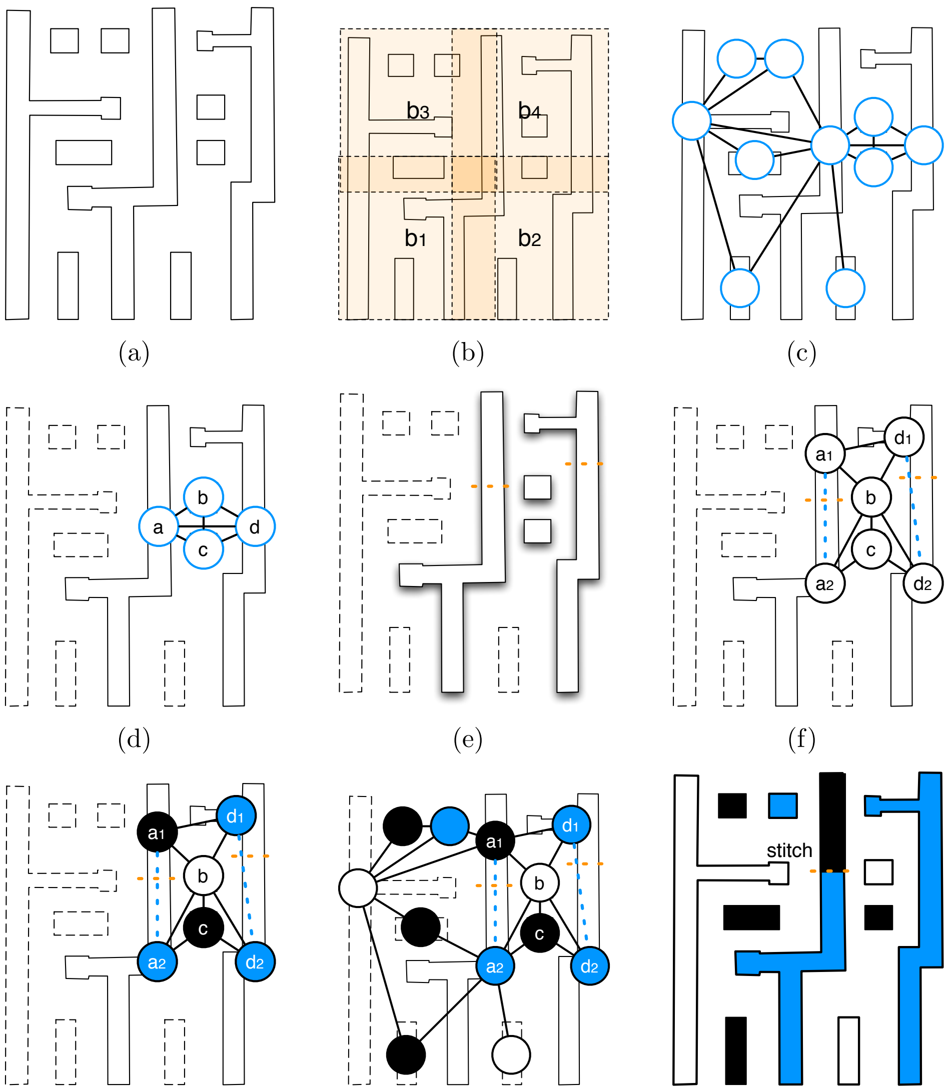


Figure 2.20: An example of the layout decomposition flow.

Fig. 2.20 illustrates an example to show the decomposition process step by step. Given the input layout as in Fig. 2.20(a), we partition it into a set of bins $\{b_1, b_2, b_3, b_4\}$ (see Fig. 2.20(b)). Then the layout graph is constructed (see Fig. 2.20(c)), where the ten vertices representing the ten features in the input layout, and each vertex represents a polygonal feature (shape) where there is an edge (conflict edge) between two vertices if and only if those two vertices are within the minimum coloring distance min_s. During the layout graph simplification, the vertices whose degree equal or smaller than two are iteratively removed from the graph. The simplified layout graph, shown in Fig. 2.20(d), only contains vertices a, b, c and d. Fig. 2.20(d) shows the projection results. Followed by stitch candidate generation [56], there are two stitch candidates for TPL (see Fig. 2.20(e)). Based on the two stitch candidates, vertices a and d are divided into two vertices, respectively. The constructed decomposition graph is given in Fig. 2.20(f). It maintains all the information about conflict edges and stitch candidates, where the solid edges are the conflict edges while the dashed edges are the stitch edges and function as stitch candidates. In each decomposition graph, a color assignment, which contains semidefinite programming (SDP) formulation and partition based mapping, is carried out. During color assignment, the six vertices in the decomposition graph are assigned into three groups: $\{a_1, c\}, \{b\}$ and $\{a_2, d_1, d_2\}$ (see Fig. 2.20(g) and Fig. 2.20(h)). Here one stitch on feature a is introduced. After iteratively recover the removed vertices, the final decomposed layout is shown in Fig. 2.20(i). Our last process should be decomposition graphs merging, which combines the results on all decomposition graphs. Since this example has only one decomposition graph, this process is skipped.

Density balance, especially local density balance, is seamlessly inte-

Table 2.7: Notations used

CE	the set of conflict edges
SE	the set of stitch edges
V	the set of features
B	the set of local bins

grated into each step of our decomposition flow. In this section, we first elaborate how to integrate the density balance into the mathematical formulation and corresponding SDP formulation. Followed by some discussion for density balance in all other steps.

2.3.2.1 Density Balanced SDP Algorithm

For each decomposition graph, density balanced color assignment is carried out. Some notations used are listed in Table 2.7.

The mathematical formulation for the general density balanced layout decomposition is shown in (2.9), where the objective is to simultaneously minimize the conflict number, the stitch number and the density uniformity of all bins. Here α and β are user-defined parameters for assigning the relative weights among the three values.

Here x_i is a variable representing the color (mask) of feature r_i, c_{ij} is a binary variable for the conflict edge $e_{ij} \in CE$, and s_{ij} is a binary variable for the stitch edge $e_{ij} \in SE$. The constraints (2.9a) and (2.9b) are used to evaluate the conflict number and stitch number, respectively. The constraint (2.9e) is nonlinear, which makes the program (2.9) hard to be formulated into integer

$$\min \sum_{e_{ij}\in CE} c_{ij} + \alpha \sum_{e_{ij}\in SE} s_{ij} + \beta \cdot \sum_{b_k \in B} DU_k \tag{2.9}$$

$$\begin{aligned}
\text{s.t.}\quad & c_{ij} = (x_i == x_j) && \forall e_{ij} \in CE && (2.9a)\\
& s_{ij} = x_i \oplus x_j && \forall e_{ij} \in SE && (2.9b)\\
& x_i \in \{1,2,3\} && \forall r_i \in V && (2.9c)\\
& d_{kc} = \sum_{x_i = c} den_{ki} && \forall r_i \in V,\quad b_k \in B && (2.9d)\\
& DU_k = \max\{d_{kc}\}/\min\{d_{kc}\} && \forall b_k \in B && (2.9e)
\end{aligned}$$

linear programming (ILP) as in [109]. Similar nonlinear constraints occur in the floorplanning problem [19], where Tayor expansion is used to linearize the constraint into ILP. However, Tayor expansion will introduce the penalty of accuracy. Compared with the traditional time consuming ILP, semidefinite programming (SDP) has been shown to be a better approach in terms of runtime and solution quality tradeoffs [109]. However, how to integrate the density balance into the SDP formulation is still an open question. In the following we will show that instead of using the painful Tayor expansion, this nonlinear constraint can be integrated into SDP without losing any accuracy.

In SDP formulation, the objective function is the representation of vector inner products, i.e., $\vec{v_i} \cdot \vec{v_j}$. At the first glance, the constraint (2.9e) cannot be formulated into an inner product format. However, we will show that density uniformity DU_k can be optimized through considering another form $DU_k^* = d_{k1} \cdot d_{k2} + d_{k1} \cdot d_{k3} + d_{k2} \cdot d_{k3}$. This is based on the following observation: maximizing DU_k^* is equivalent to minimizing DU_k.

Lemma 3. *$DU_k^* = 2/3 \cdot \sum_{i,j \in V} den_{ki} \cdot den_{kj} \cdot (1 - \vec{v_i} \cdot \vec{v_j})$, where den_{ki} is the density of feature r_i in bin b_k.*

Proof. First of all, let us calculate $d_1 \cdot d_2$. For all vectors $\vec{v_i} = (1, 0)$ and all vectors $\vec{v_j} = (-\frac{1}{2}, \frac{\sqrt{3}}{2})$, we can see that

$$\begin{aligned} &\sum_i \sum_j len_i \cdot len_j \cdot (1 - \vec{v_i} \cdot \vec{v_j}) = \sum_i \sum_j len_i \cdot len_j \cdot 3/2 \\ =&3/2 \cdot \sum_i len_i \sum_j len_j = 3/2 \cdot d_1 \cdot d_2 \end{aligned}$$

So $d_1 \cdot d_2 = 2/3 \cdot \sum_i \sum_j len_i \cdot len_j \cdot (1 - \vec{v_i} \cdot \vec{v_j})$, where $\vec{v_i} = (1, 0)$ and $\vec{v_j} = (-\frac{1}{2}, \frac{\sqrt{3}}{2})$. We can also calculate $d_1 \cdot d_3$ and $d_2 \cdot d_3$ using similar methods. Therefore,

$$\begin{aligned} DU_2 &= d_1 \cdot d_2 + d_1 \cdot d_3 + d_2 \cdot d_3 \\ &= 2/3 \cdot \sum_{i,j \in V} len_i \cdot len_j \cdot (1 - \vec{v_i} \cdot \vec{v_j}) \end{aligned}$$

□

Because of Lemma 3, the DU_k^* can be represented as a vector inner product, then we have achieved the following theorem.

Theorem 3. *Maximizing DU_k^* can achieve better density balance in bin b_k.*

Note that we can remove the constant $\sum_{i,j \in V} den_{ki} \cdot den_{kj} \cdot 1$ in DU_k^* expression. Similarly, we can eliminate the constants in the calculation of the conflict and stitch numbers. The simplified vector program is as follows:

Formulation (2.10) is equivalent to the mathematical formulation (2.9), and it is still NP-hard to be solved exactly. Constraint (2.10b) requires the

$$\min \sum_{e_{ij}\in CE} (\vec{v_i}\cdot\vec{v_j}) - \alpha \sum_{e_{ij}\in SE} (\vec{v_i}\cdot\vec{v_j}) - \beta\cdot\sum_{b_k\in B} DU_k^* \tag{2.10}$$

$$\text{s.t.}\quad DU_k^* = -\sum_{i,j\in V} den_{ki}\cdot den_{kj}\cdot(\vec{v_i}\cdot\vec{v_j}) \quad \forall b_k\in B \tag{2.10a}$$

$$\vec{v_i}\in\{(1,0),(-\frac{1}{2},\frac{\sqrt{3}}{2}),(-\frac{1}{2},-\frac{\sqrt{3}}{2})\} \tag{2.10b}$$

solutions to be discrete. To achieve a good tradeoff between runtime and accuracy, we can relax (2.10) into a SDP formulation, as shown in Theorem 4.

Theorem 4. *Relaxing vector program (2.10) can get the SDP formulation (2.11).*

$$\text{SDP:}\ \min \quad A\bullet X \tag{2.11}$$
$$X_{ii}=1,\quad \forall i\in V \tag{2.11a}$$
$$X_{ij}\geq -\frac{1}{2},\quad \forall e_{ij}\in CE \tag{2.11b}$$
$$X\succeq 0 \tag{2.11c}$$

where A_{ij} is the entry that lies in the i-th row and the j-th column of matrix A:

$$A_{ij}=\begin{cases} 1+\beta\cdot\sum_k den_{ki}\cdot den_{kj}, & \forall b_k\in B, e_{ij}\in CE\\ -\alpha+\beta\cdot\sum_k den_{ki}\cdot den_{kj}, & \forall b_k\in B, e_{ij}\in SE\\ \beta\cdot\sum_k den_{ki}\cdot den_{kj}, & \text{otherwise}\end{cases}$$

Due to space limit, the detailed proof is omitted. The solution of (2.11) is continuous instead of discrete, and provides a lower bound of vector program (2.10). en other words, (2.11) provides an approximated solution to (2.10).

2.3.2.2 Density Balanced Mapping

Each X_{ij} in solution of (2.11) corresponds to a feature pair (r_i, r_j). The value of X_{ij} provides a guideline, i.e., whether two features r_i and r_j should be in same color. If X_{ij} is close to 1, features r_i and r_j tend to be in the same color (mask); while if it is close to -0.5, r_i and r_j tend to be in different colors (masks). With these guidelines a mapping procedure is adopted to finally assign all input features into three colors (masks).

In [109], a greedy approach was applied for the final color assignment. The idea is straightforward: all X_{ij} values are sorted, and vertices r_i and r_j with larger X_{ij} value tend to be in the same color. The X_{ij} can be classified into two types: clear and vague. If most of the X_{ij}s in matrix X are clear (close to 1 or -0.5), this greedy method may achieve good result. However, if the decomposition graph is not 3-colorable, some values in matrix X are vague. For the vague X_{ij}, e.g., 0.5, the greedy method may not be so effective.

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

Algorithm 4 Partition based Mapping

```
Require: Solution matrix $X$ of the program (2.11).
 1: Label each non-zero entry $X_{i,j}$ as a triplet $(X_{ij}, i, j)$;
 2: Sort all $(X_{ij}, i, j)$ by $X_{ij}$;
 3: for all triples with $X_{ij} > th_{unn}$ do
 4:     Union(i, j);
 5: end for
 6: for all triples with $X_{ij} < th_{sp}$ do
 7:     Separate(i, j);
 8: end for
 9: Construct graph $G_M$;
10: if graph size $\leq 3$ then
11:     return;
12: else if graph size $\leq 7$ then
13:     Backtracking based three-way partitioning;
14: else
15:     FM based three-way partitioning;
16: end if
```

Algorithm 4 shows our partition based mapping procedure. Given the solutions from program (2.11), some triplets are constructed and sorted to maintain all non-zero X_{ij} values (lines 1–2). The mapping incorporates two stages to deal with the two different types. The first stage (lines 3–8) is similar to that in [109]. If X_{ij} is clear then the relationship between vertices r_i and r_j can be directly determined. Here th_{unn} and th_{sp} are user-defined threshold values. For example, if $X_{ij} > th_{unn}$, which means that r_i and r_j should be in the same color, then function Union(i, j) is applied to merge them into a large vertex. Similarly, if $X_{ij} < th_{sp}$, then function Separate(i, j) is used to label r_i and r_j as incompatible. In the second stage (lines 9–16) we deal with the vague X_{ij} values. During the previous stage some vertices have been merged,

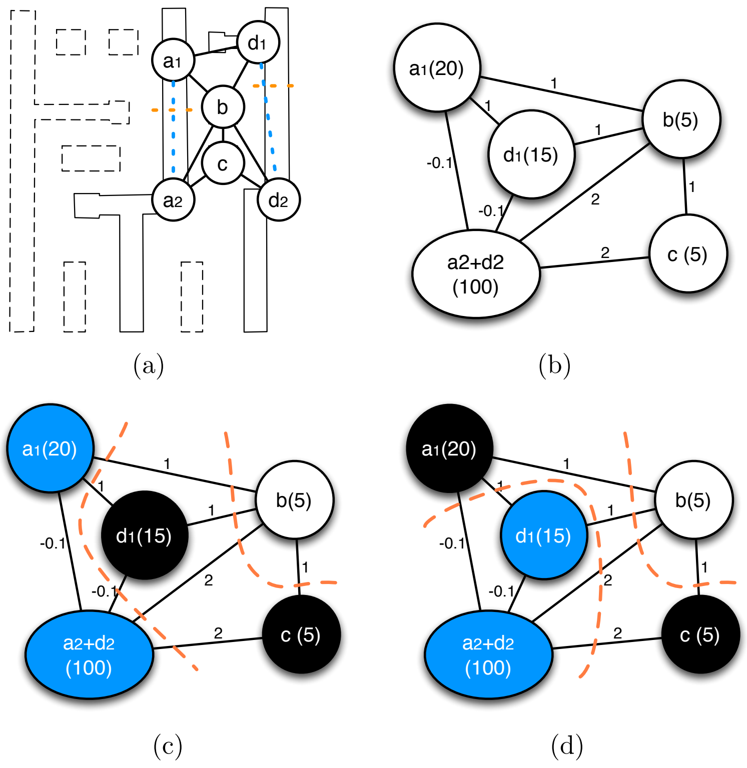


Figure 2.21: Density Balanced Mapping. (a) Decomposition graph. (b) Construct graph G_M. (c) Mapping result with cut value 8.1 and density uniformities 24. (d) A better mapping with cut 8.1 and density uniformities 23.

therefore the total vertex number is not large. Here we construct a graph G_M to represent the relationships among all the remanent vertices (line 9). Each edge e_{ij} in this graph has a weight representing the cost if vertices i and j are assigned into same color. Therefore, the color assignment problem can be formulated as a maximum-cut partitioning problem on G_M (line 10–16).

Through assigning a weight to each vertex representing its density, graph G_M is able to balance density among different bins. Based on the

G_M, a partitioning is performed to simultaneously achieve a maximum-cut and balanced weight among different parts. Note that we need to modify the gain function, then in each move, we try to achieve a more balanced and larger cut partitions.

An example of the density balanced mapping is shown in Fig. 2.21. Based on the decomposition graph (see Fig. 2.21 (a)), SDP is formulated. Given the solutions of SDP, after the first stage of mapping, vertices a_2 and d_2 are merged in to a large vertex. As shown in Fig. 2.21(b), the graph G_M is constructed, where each vertex is associated with a weight. There are two partition results with the same cut value 8.1 (see Fig. 2.21 (c) and Fig. 2.21 (d)). However, their density uniformities are 24 and 23, respectively. To keep a more balanced density result, the second partitioning in Fig. 2.21 (c) is adopted as color assignment result.

It is well known that the maximum-cut problem, even for a 2-way partition, is NP-hard. However, we observe that in many cases, after the global SDP optimization, the graph size of G_M could be quite small, i.e., less than 7. For these small cases, we develop a backtracking based method to search the entire solution space. Note that here backtracking can quickly find the optimal solution even through three-way partitioning is NP-hard. If the graph size is larger, we propose a heuristic method, motivated by the classic FM partitioning algorithm [33] [82]. Different from the classic FM algorithm, we make the following modifications. (1) In the first stage of mapping, some vertices are labeled as incomparable, therefore before moving a vertex from one partition to another, we should check whether it is legal. (2) Classical FM algorithm is for min-cut problem, we need to modify the gain function of each move to achieve a maximum cut.

The runtime complexity of graph construction is $O(m)$, where m is the vertex number in G_M. The runtime of three-way maximum-cut partitioning algorithm is $O(mlogm)$. Besides, the first stage of mapping needs $O(n^2logn)$ [109]. Since m is much smaller than n, the complexity of density balanced mapping is $O(n^2logn)$.

2.3.2.3 Density Balanced Graph Division

Here we show that the layout graph simplification, which was proposed in [109], can consider the local density balance as well. During layout graph simplification, we iteratively remove and push all vertices with degree less than or equal to two. After the color assignment on the remained vertices, we iteratively recover all the removed vertices and assign legal colors. Instead of randomly picking one, we search a legal color which is good for the density uniformities.

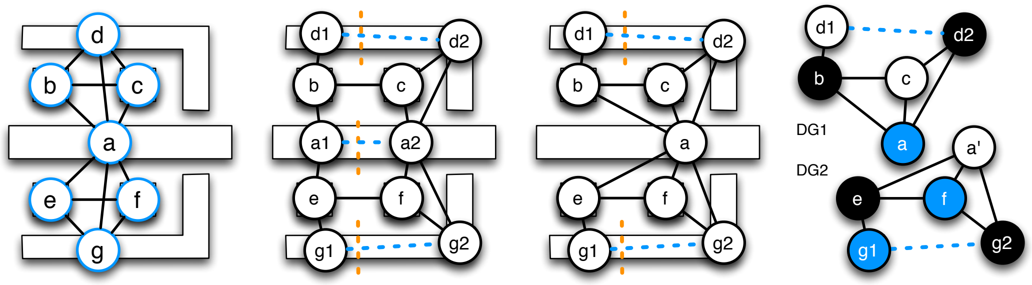
2.3.2.4 Speedup Techniques

Our layout decomposer applies a set of graph simplification techniques proposed by recent works:

- Independent Component Computation [109] [32] [56];

- Vertex with Degree Less than 3 Removal [109] [32] [56];
- 2-Edge-Connected Component Computation [109] [32] [56];
- 2-Vertex-Connected Component Computation [32] [56].

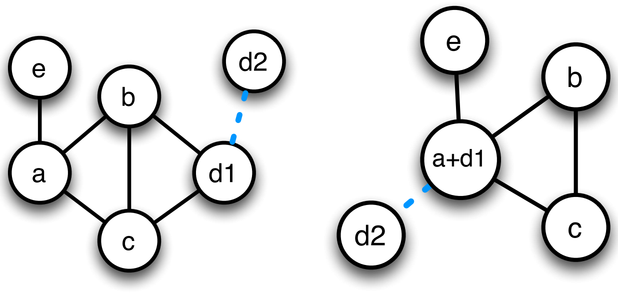
Apart from the above graph simplifications, our decomposer proposes a set of novel speedup techniques, which would be introduced in the following.



(a) (b) (c) (d)

Figure 2.22: Layout graph cut vertex stitch forbiddance.

Our first technique is called **LG Cut Vertex Stitch Forbiddance**. A vertex of a graph is called a cut vertex if its removal decomposes the graph into two or more connected components. Cut vertices can be identified through the process of bridge computation [109]. During stitch candidate generation, forbidding any stitch candidate on cut vertices can be helpful for later decomposition graph simplification. Fig. 2.22 (a) shows a layout graph, where feature a is a cut vertex, since its removal can partition the layout graph into two parts: {b, c, d} and {e, f, g}. If stitch candidates are introduced within a, the corresponding decomposition graph is illustrated in Fig. 2.22 (b), which is



(a) (b)

Figure 2.23: DG vertex clustering to reduce the decomposition graph size.

hard to be further simplified. If we forbid the stitch candidate on a, the corresponding decomposition graph is shown in Fig. 2.22 (c), where a is still cut vertex in decomposition graph. Therefore we can apply 2-connected component computation [32] to simplify the problem size, and apply color assignment separately (see Fig. 2.22 (d)).

Our second technique, **Decomposition graph vertex clustering**, is a speedup technique to further reduce the decomposition graph size. As shown in Fig. 2.23 (a), vertices a and d_1 share the same conflict relationships against b and c. Besides, there is no conflict edges between a and d_1. If no conflict is introduced, vertices a and d_1 should be assigned the same color, therefore we can cluster them together, as shown in Fig. 2.23 (b). Note that the stitch and conflict relationships are also merged. Applying vertex clustering in decomposition graph can further reduce the problem size.

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

Algorithm 5 Fast Color Assignment Trial

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

The fast color assignment trial is shown in Algorithm 5. First, we iteratively remove the vertex with conflict degree (d_{conf}) less than 3 and stitch degree (d_{stit}) less than 2 (lines 1–3). If some vertices cannot be removed, we recover all the vertices in stack S, then return $false$; Otherwise, the vertices in S are iteratively popped (recovered) (lines 8–12). For each vertex n popped, since it is connected with at most one stitch edge, we can always assign one color without introducing conflict or stitch.

2.3.3 Experimental Results

We implement our decomposer in C++ and test it on an Intel Xeon 3.0GHz Linux machine with 32G RAM. ISCAS 85&89 benchmarks from [109] are used, where the minimum coloring spacing dis_m was set the same with previous studies [109] [32]. Besides, to perform a comprehensive comparison, we also test on other two benchmark suites. The first suite is with six dense benchmarks ("c9_total"-"s5_total"), while the second suite is two synthesized OpenSPARC T1 designs "mul_top" and "exu_ecc" with Nangate 45nm standard cell library [3]. When processing these two benchmark suites we set the minimum coloring distance $dis_m = 2 \cdot w_{min} + 3 \cdot s_{min}$, where w_{min} and s_{min} denote the minimum wire width and the minimum spacing, respectively. The parameter α is set as 0.1. The size of each bin is set as $10 \cdot dis_m \times 10 \cdot dis_m$. We use CSDP [15] as the solver for the semidefinite programming (SDP).

In the first experiment, we compare our decomposer with the state-of-the-art layout decomposers which are not balanced density aware [109] [32] [56]. We obtain the binary files from [109] and [32]. Since currently we cannot obtain the binary for decomposer in [56], we directly use the results listed in [56]. Here our decomposer is denoted as "**SDP+PM**", where "PM" means the partition based mapping. The β is set as 0. In other words, SDP+PM only optimizes for stitch and conflict number. Table 2.8 shows the comparison

[1]The results of DAC'13 decomposition are from [56].

Table 2.8: Comparison of Runtime and Performance.

Circuit	ICCAD'11 [109]				DAC'12 [32]				DAC'13 [56][1]				SDP+PM			
	cn#	st#	cost	CPU(s)	cn#	st#	cost	CPU(s)	cn#	st#	cost	CPU(s)	cn#	st#	cost	CPU(s)
C432	3	1	3.1	0.09	0	6	0.6	0.03	0	4	0.4	0.01	0	4	0.4	0.2
C499	0	0	0	0.07	0	0	0	0.04	0	0	0	0.01	0	0	0	0.2
C880	1	6	1.6	0.15	1	15	2.5	0.05	0	7	0.7	0.01	0	7	0.7	0.3
C1355	1	2	1.2	0.07	1	7	1.7	0.07	0	3	0.3	0.01	0	3	0.3	0.3
C1908	0	1	0.1	0.07	1	0	1	0.1	0	1	0.1	0.01	0	1	0.1	0.3
C2670	2	4	2.4	0.17	2	14	3.4	0.16	0	6	0.6	0.04	0	6	0.6	0.4
C3540	5	6	5.6	0.27	2	15	3.5	0.2	1	8	1.8	0.05	1	8	1.8	0.5
C5315	7	7	7.7	0.3	3	11	4.1	0.27	0	9	0.9	0.05	0	9	0.9	0.7
C6288	82	131	95.1	3.81	19	341	53.1	0.3	14	191	33.1	0.25	1	213	22.3	2.7
C7552	12	15	13.5	0.77	3	46	7.6	0.42	1	21	3.1	0.1	0	22	2.2	1.1
S1488	1	1	1.1	0.16	0	4	0.4	0.08	0	2	0.2	0.01	0	2	0.2	0.3
S38417	44	55	49.5	18.8	20	122	32.2	1.25	19	55	24.5	0.42	19	55	24.5	7.9
S35932	93	18	94.8	89.7	46	103	56.3	4.3	44	41	48.1	0.82	44	48	48.8	21.4
S38584	63	122	75.2	92.1	36	280	38.8	3.7	36	116	47.6	0.77	37	118	48.8	22.2
S15850	73	91	82.1	79.8	36	201	56.1	3.7	36	97	45.7	0.76	34	101	44.1	20.0
avg.	25.8	30.7	28.9	19.1	11.3	60.87	17.42	0.978	10.1	37.4	13.8	0.22	9.07	39.8	13.0	5.23
ratio			**2.2**	**3.65**			**1.34**	**0.19**			**1.06**	**0.04**			**1.0**	**1.0**

Table 2.9: Comparison on Very Dense Layouts

Circuit	ICCAD 2011 [109]				DAC 2012 [32]				SDP+PM			
	cn#	st#	cost	CPU(s)	cn#	st#	cost	CPU(s)	cn#	st#	cost	CPU(s)
mul_top	836	44	840.4	236	457	0	457	0.8	118	271	145.1	57.6
exu_ecc	119	1	119.1	11.1	53	0	53	0.7	22	64	28.4	4.3
c9_total	886	228	908.8	47.4	603	641	667.1	0.52	117	1009	217.9	7.7
c10_total	2088	554	2143.4	52	1756	1776	1933.6	1.1	248	1876	435.6	19
s2_total	2182	390	2221	936.8	1652	5976	2249.6	4	703	5226	1225.6	70.7
s3_total	6844	72	6851.2	7510.1	4731	13853	6116.3	13.1	958	10572	2015.2	254.5
s4_total	NA	NA	NA	>10000	3868	13632	5231.2	13	1151	11091	2260.1	306
s5_total	NA	NA	NA	>10000	4650	16152	6265.2	12.9	1391	13683	2759.3	350.4
avg.	NA	NA	NA	>3600	2221.3	6503.8	2871.6	5.8	588.5	5474	1135.9	134
ratio			-	**>27.0**			**2.53**	**0.05**			**1.0**	**1.0**

in terms of runtime and performance. For each decomposer we list its stitch number, conflict number, cost and runtime. The columns "cn#" and "st#" denote the conflict number and the stitch number, respectively. "cost" is the cost function, which is set as cn# $+0.1\times$ st#. "CPU(s)" is computational time in seconds.

First, we compare SDP+PM with the decomposer in [109], which is based on SDP formulation as well. From Table 2.8 we can see that the new stitch candidate generation (see [56] for more details) and partition-based mapping can achieve better performance (reducing the cost by around 55%). Besides, SDP+PM can get nearly $4\times$ speed-up. The reason is that, compared with [109], a set of speedup techniques, i.e., 2-vertex-connected component computation, layout graph cut vertex stitch forbiddance, decomposition graph vertex clustering, and fast color assignment trial, are proposed. Second, we compare SDP+PM with the decomposer in [32], which applies several graph based simplifications and maximum independent set (MIS) based heuristic. From Table 2.8 we can see that although the decomposer in [32] is faster, MIS based heuristic has worse solution qualities (around 33% cost penalty compared to SDP+PM). Compared with the decomposer in [56], although SDP+PM is slower, it can reduce the cost by around 6%.

In addition, we compare SDP-PM with other two decomposers [109] [32] for some very dense layouts, as shown in Table 2.9. We can see that for some cases the decomposer in [109] cannot finish in 1000 seconds. Compared with [32] work, SDP+PM can reduce cost by 65%. It is observed that compared

Table 2.10: Balanced density impact on EPE

Circuit	SDP+PM			SDP+PM+DB		
	cost	CPU(s)	EPE#	cost	CPU(s)	EPE#
C432	0.4	0.2	0	0.4	0.2	0
C499	0	0.2	0	0	0.2	0
C880	0.7	0.3	10	0.7	0.3	7
C1355	0.3	0.3	18	0.3	0.3	15
C1908	0.1	0.3	130	0.1	0.3	58
C2670	0.6	0.4	168	0.6	0.4	105
C3540	1.8	0.5	164	1.8	0.5	79
C5315	0.9	0.7	225	1.0	0.7	115
C6288	22.3	2.7	31	32.0	2.8	15
C7552	2.2	1.1	273	2.5	1.1	184
S1488	0.2	0.3	72	0.2	0.3	44
S38417	24.5	7.9	420	24.5	8.5	412
S35932	48.8	21.4	1342	49.8	24	1247
S38584	48.8	22.2	1332	49.1	23.7	1290
S15850	44.1	20	1149	47.3	21.3	1030
avg.	13.0	5.23	355.6	14.0	5.64	306.7
ratio	1.0	1.0	**1.0**	1.07	1.08	**0.86**

with other decomposers, SDP+PM demonstrates much better performance when the input layout is dense. The reason may be that when the input layout is dense, through graph simplification, each independent problem size may still be quite large, then SDP based approximation can achieve better results than heuristic. It can be observed that for the last three cases our decomposer could reduce thousands of conflicts. Each conflict may require manual layout modification or high ECO efforts, which are very time consuming. Therefore, even our runtime is more than [32], it is still acceptable (less than 6 minutes for the largest benchmark).

In the second experiment, we test our decomposer for the density balancing. We analyze edge placement error (EPE) using Calibre-Workbench [2] and industry-strength setup. For analyzing the EPE in our test cases, we use systematic lithography process variation, such as focus $\pm$50nm and dose $\pm$5%. In Table 2.10, we compare SDP+PM with "**SDP+PM+DB**", which is our density balanced decomposer. Here β is set as 0.04 (we have tested different β values, we found that bigger β does not help much any more; meanwhile, we still want to give conflict and stitch higher weights). Column "cost" also lists the weighted cost of conflict and stitch, i.e., cost = cn#+0.1$\times$st#.

From Table 2.10 we can see that by integrating density balance into our decomposition flow, our decomposer (SDP+PM+DB) can reduce EPE hotspot number by 14%. Besides, density balanced SDP based algorithm can maintain similar performance to the baseline SDP implementation: only 7% more cost of conflict and stitch, and only 8% more runtime. In other words, our decomposer can achieve a good density balance while keeping comparable conflicts/stitches.

We further compare the density balance, especially EPE distributions for very dense layouts. As shown in Table 2.11, our density balanced decomposer (SDP+PM+DB) can reduce EPE distribution number by 7%. Besides, for very dense layouts, density balanced SDP approximation can maintain similar performance with plain SDP implementation: only 4% more runtime.

In addition, we demonstrate the scalability of our decomposer, especially the SDP formulation. Penrose benchmarks from [23] are used to explore

Table 2.11: Additional Comparison for Density Balance

Circuit	SDP+PM			SDP+PM+DB		
	cost	CPU(s)	EPE#	cost	CPU(s)	EPE#
mul_top	145.1	57.6	632	147.5	63.8	630
exu_ecc	28.4	4.3	140	33.9	4.8	138
c9_total	217.9	7.7	60	218.6	8.3	60
c10_total	435.6	19	77	431.3	19.6	76
s2_total	1225.6	70.7	482	1179.3	75	433
s3_total	2015.2	254.5	1563	1937.5	274.5	1421
s4_total	2260.1	306	1476	2176.3	310	1373
s5_total	2759.3	350.4	1270	2673.9	352	1171
avg.	1135.9	134	712.5	1099.8	138.5	662.8
ratio	1.0	1.0	**1.0**	0.97	1.04	**0.93**

the scalability of SDP runtime. No graph simplification is applied, therefore all runtime is consumed by solving SDP formulation. Fig. 2.24 illustrates the relationship between graph (problem) size against SDP runtime. Here the X axis denotes the number of nodes (e.g., the problem size), and the Y axis shows the runtime. We can see that the runtime complexity of SDP is less than O($n^{2.2}$).

2.3.4 Summary

We have proposed a high performance TPL layout decomposer with balanced density. Density balancing was integrated into all the key steps of our decomposition flow. In addition, we proposed a set of speedup techniques, such as layout graph cut vertex stitch forbiddance, decomposition graph vertex clustering, and fast color assignment trial. Compared with state-of-the-art frameworks, our decomposer demonstrates the best performance in minimizing

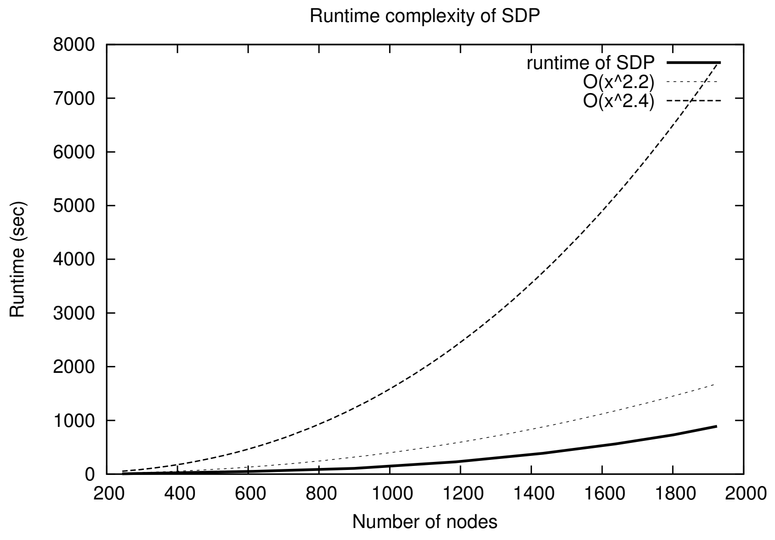


Figure 2.24: Scalability of SDP Formulation.

the cost of conflicts and stitches. Furthermore, our balanced decomposer can obtain less EPE while maintaining very comparable conflict and stitch results. As TPL may be adopted by industry for 14nm/11nm nodes, we believe more research will be needed to enable TPL-friendly design and mask synthesis.

2.4 Layout Decomposition for Triple Patterning with End-Cutting

So far we have discussed the conventional process of TPL, called LELELE, which is with the same principle of litho-etch-litho-etch (LELE) type double patterning lithography (DPL). Here "L" and "E" represent one lithography process and one etch process, respectively. Although LELELE process has been widely studied by industry and academia, there are twofold issues de-

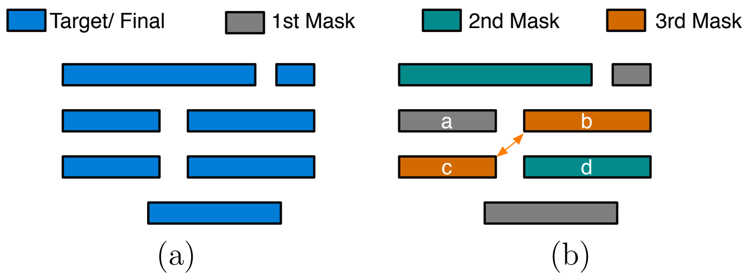


Figure 2.25: Process of LELELE type triple patterning lithography (a) Target features; (b) Layout decomposition with one conflict introduced.

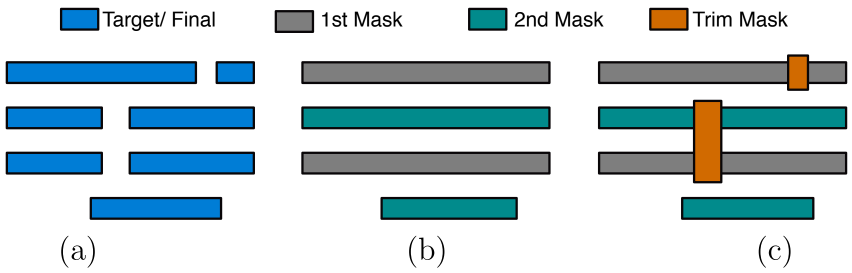


Figure 2.26: Process of LELE-EC type triple patterning lithography (a) Target features; (b) First and second mask patterns; (c) Trim mask, and final decomposition without conflict.

rived. On one side, even with stitch insertion, there are some native conflicts in LELELE, like 4-clique conflict [109]. For example, Fig. 2.25 illustrates a 4-clique conflict among features a, b, c, and d. No matter how to assign the colors, there is at least one conflict. Since this 4-clique structure is common in advanced standard cell design, LELELE type TPL still suffers from the native conflict problem. On the other side, compared with LELE type double patterning, there are more serious overlapping problem in LELELE [12].

To overcome all these limitations from LELELE, recently Lin [61] proposed a new TPL manufacturing process, called LELE-end-cutting (LELE-

EC). As a TPL, this new manufacturing process contains three mask steps, namely first mask, second mask, and *trim* mask. Fig. 2.26 illustrates an example of the LELE-EC process. To generate target features in Fig. 2.26(a), the first and second masks are used for pitch splitting, which is similar to LELE type DPL process. These two masks are shown in Fig. 2.26(b). Finally, a trim mask is applied to trim out the desired region as in Fig. 2.26(c). In other words, the trim mask is used to generate some end-cuts to further split feature patterns. Although the target features are not LELELE-friendly, they are LELE-EC process friendly that with LELE-EC the features can be decomposed without any conflict. In addition, if all cuts are properly designed or distributed, LELE-EC can introduce better printability then conventional LELELE process [61].

For a design with four short features, Fig. 2.27 and Fig. 2.28 present its simulated images through LELELE and LELE-EC processes, respectively. The lithography simulations are computed based on the partially coherent imaging system, where the 193nm illumination source is modeled as a kernel matrix given by [13]. To model the photoresist effect with the exposed light intensity, we use the constant threshold model with threshold 0.225. We can make several observations from these simulated images. First, there are some round-offs around the line ends (see Fig. 2.27 (c)). Second, to reduce the round-off issues, as illustrated in Fig. 2.28 (b), in LELE-EC process the short lines can be merged into longer lines, then the trim mask is used to cut off some spaces. It shall be noted that there might be some corner roundings due

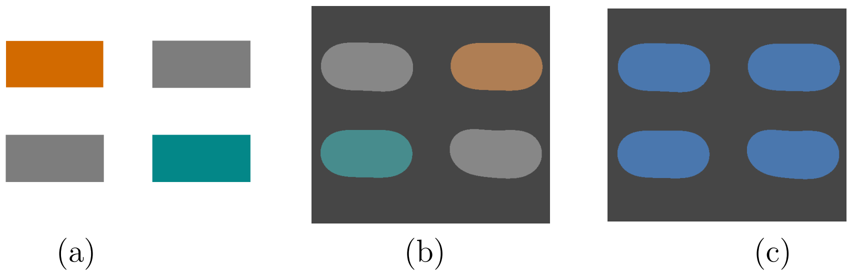


Figure 2.27: LELELE process example. (a) Decomposed result; (b) Simulated images for different masks; (c) Combined simulated image as the final printed patterns.

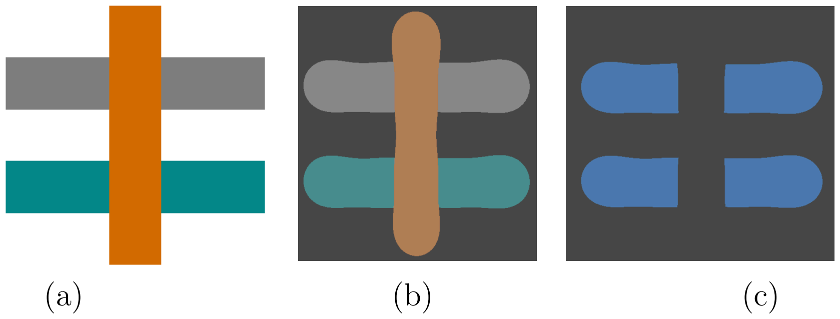


Figure 2.28: LELE-EC process example. (a) Decomposed result; (b) Simulated images for different masks, where orange pattern is trim mask; (c) Combined simulated image as the final printed patterns.

to the edge shorting of trim mask patterns. However, since the line shortening or rounding is a strong function of the line width [67], and we observe that usually trim mask patterns can be much longer than line-end width, thus we assume the rounding caused by the trim mask is insignificant. This assumption is demonstrated as in Fig. 2.28 (c).

Many research have been carried out to solve the corresponding design problems for LELELE type TPL. The layout decomposition problem has been well studied [23,32,56,92,105,106,109,115]. In addition, the related constraints have been considered in early physical design stages, like routing [62, 65], standard cell design [91, 107], and detailed placement [107]. However, only few attempts have been made to address the LELE-EC layout decomposition problem. It shall be noted that although trim mask can bring about better printability, it does introduce more design challenge, especially in layout decomposition stage. In this section, we propose a comprehensive study for LELE-EC layout decomposition. Given a layout which is specified by features in polygonal shapes, we extract the geometrical relationships and construct the conflict graphs. Furthermore, the compatibility of all end-cuts candidates are also modeled in the conflict graphs. Based on the conflict graphs, integer linear programming (ILP) is formulated to assign each vertex into one layer. Our goal in the layout decomposition is to minimize the conflict number, and at the same time minimize the overlapping errors.

2.4.1 Preliminaries and Problem Formulation

2.4.1.1 Layout Graph

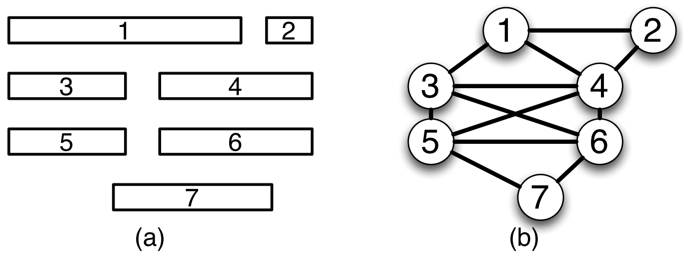


Figure 2.29: Layout graph construction. (a) Input layout; (b) Layout graph with conflict edges;

Given a layout which is specified by features in polygonal shapes, layout graph [109] is constructed. As shown in Fig. 2.29, the layout graph is an undirected graph with a set of vertices V and a set of conflict edges CE. Each vertex in V represents one input feature. There is an edge in CE if and only if the two features are within minimum coloring distance dis_m of each other. In other words, each edge in CE is a conflict candidate. Fig. 2.29(a) shows one input layout, and the corresponding layout graph is in Fig. 2.29(b). Here the vertex set $V = \{1, 2, 3, 4, 5, 6, 7\}$, while the conflict edge set $CE = \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7)\}$. For each edge (conflict candidate), we check whether there is an end-cut candidate. For each end-cut candidate $i - j$, if it is applied, then features i and j will be merged into one feature. By this way the corresponding conflict edge can be removed. If stitch is considered in layout decomposition, some vertices in layout graph can be split into several segments. The segments in one layout graph vertex are connected through *stitch edges*. All these stitch edges are included in a set, called SE. Please refer to [48] for the details of stitch candidate generation.

2.4.1.2 End-Cut Graph

Since all the end-cuts are manufactured through one single exposure process, they should be distributed far away from each other. That is, two end-cuts have conflict if they are within minimum end-cut distance dis_c of each other. Note that these conflict relationships among end-cuts are not available in layout graph, therefore we construct *end-cut graph* to store the

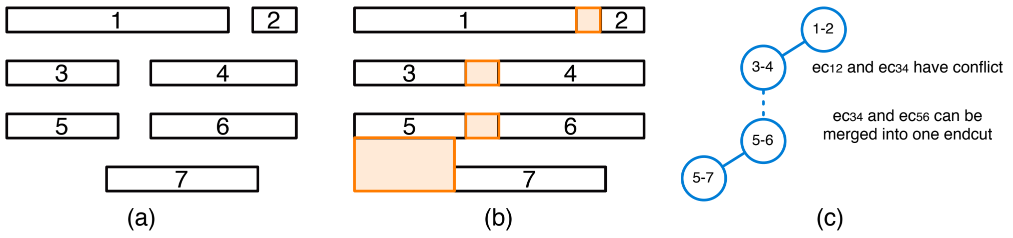

1
2
3
4
5
6
7
(a)
1
2
3
4
5
6
7
(b)
1-2
3-4
5-6
5-7
ec12 and ec34 have conflict
ec34 and ec56 can be merged into one endcut
(c)

Figure 2.30: End-cut graph construction. (a) Input layout; (b) Generated end-cut candidates. (c) End-cut graph.

relationships. Fig. 2.30(a) gives an input layout example, with all end-cut candidates pointed out in Fig. 2.30 (b); and the corresponding end-cut graph is shown in Fig. 2.30 (c). Each vertex in the end-cut graph represents one end-cut. There is an solid edge if and only if the two end-cuts conflict to each other. There is an dash edge if and only if they are close to each other, and they can be merged into one larger end-cut.

2.4.1.3 Problem Formulation

Here we give the problem formulation of layout decomposition for triple patterning with End-Cutting (LELE-EC).

Problem 3 (LELE-EC Layout Decomposition). *Given a layout which is specified by features in polygonal shapes, the layout graph and the end-cut graph are constructed. The LELE-EC layout decomposition assigns all vertices in layout graph into one of two colors, and select a set of end-cuts in end-cut graph. The objectives is to minimize the number of conflict and/or stitch.*

With the end-cut candidates generated, the LELE-EC layout decomposition is more complicated since more constraints are derived. Even there is no end-cut candidate, LELE-EC layout decomposition is similar to the LELE type DPL layout decomposition. Sun et al. in Ref. [85] showed that LELE layout decomposition with minimum conflict and minimum stitch is NP-hard, thus it is not hard to see that LELE-EC layout decomposition is NP-hard as well. NP-hard problem is a set of computational search problems that are difficult to solve [24]. No NP-hard problem can be solved in polynomial time in the worst case under the assumption that $P \neq NP$.

2.4.2 Algorithms

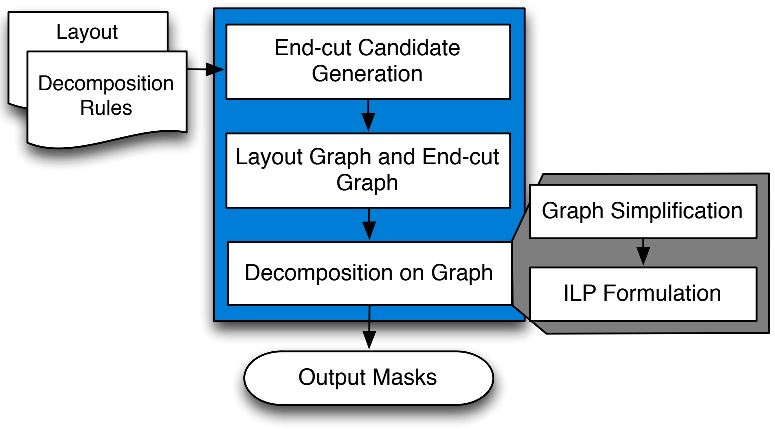


Figure 2.31: Overall flow of our layout decomposer.

The overall flow of our layout decomposer is illustrated in Fig. 2.31. First we generate all end-cut candidates to find out all possible end-cuts. Then we construct the layout graph and the end-cut graph to transform the original geometric problem into a graph problem, and thus the LELE-EC layout decomposition can be modeled as a coloring problem in layout graph and the end-cut selection problem in end-cut graph. Both the coloring problem and the end-cut selection problem can be solved through one ILP formulation. Since the ILP formulation may suffer from runtime overhead problem, we propose a set of graph simplification techniques. Besides, to further reduce the problem size, some end-cut candidates are pre-selected before ILP formulation. All the steps in the flow are detailed in the following sections.

2.4.2.1 End-Cut Candidate Generation

In this section we will explain the details of our algorithm to generate all end-cut candidates. An end-cut candidate is generated between two conflicting polygonal shapes. It should be stressed that compared with the end-cut generation in Ref. [104], our methodology has the following two differences.

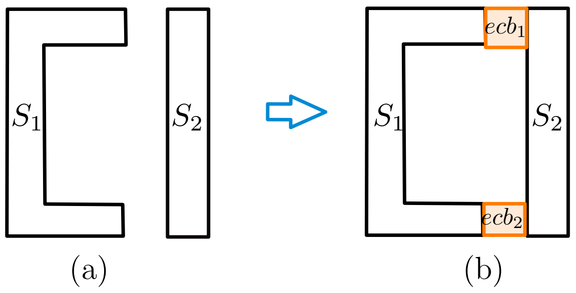


Figure 2.32: An end-cut can have multiple end-cut boxes.

- An end-cut can be a collection of multiple end-cut boxes depending on the corresponding shapes. For instance, two end-cut boxes (ecb_1 and ecb_2) need to be generated between shapes S_1 and S_2 as shown in Fig

2.32. We propose a shape-edge dependent algorithm to generate the end-cuts with multiple end-cut boxes.

- We consider the overlapping and variations caused by end-cuts. Two lithography simulations are illustrated in Fig. 2.33 and Fig. 2.34, respectively. In Fig. 2.33 we find some bad patterns or hotspots, due to the cuts between two long edges. In Fig. 2.34 we can see that the final patterns are in better shape. Therefore, to reduce the manufacturing hotspot from trim mask, during end-cut candidate generation we avoid the cuts along two long edges.

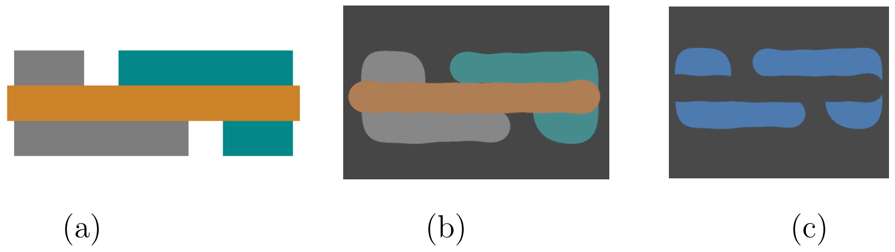

(a) (b) (c)

Figure 2.33: (a) Decomposition example where cuts are along long edges; (b) Simulated images for different masks; (c) Combined simulated image with some hotspots.

Algorithm 6 presents the key steps of generating end-cut between two polygonal shapes S_1 and S_2. A polygonal shape consists of multiple edges. For each of the shape-edge-pair, one taken from S_1 and another from S_2, the possibility of generation of end-cut box ($ecBox$) is explored and we store all such end-cut boxes in $ecBoxSet$ (Lines 2-9). The function 'generateEndCutBox(se_1, se_2)' generates an end-cut box $ecBox$ between the shape-edges se_1 and se_2.

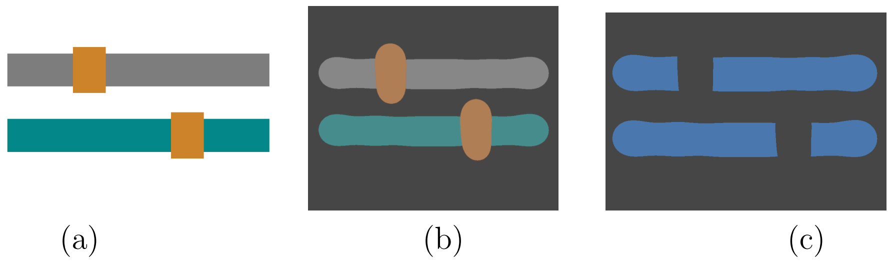


Figure 2.34: (a) Decomposition example where cuts are between line ends; (b) Simulated images for different masks; (c) Combined simulated image with good printability.

Fig. 2.35 shows how end-cut boxes are generated between two shape-edges under different situations. In Fig. 2.35(a), the end-cut box is between two shape-edges which are oriented in same direction and overlap in its x-coordinates. This type of end-cut box is called as edge-edge end-cut box. For Fig. 2.35(b) the shape edges are in same direction but they do not overlap, and in Fig. 2.35(c), the shape-edges are oriented in different directions. The end-cut boxes generated in these two cases are called corner-corner end-cut boxes. No end-cut box is generated in case of Fig. 2.35(d). In addition, end-cut boxes are not generated for the following cases: (i) the end-cut box is overlapping with any existing polygonal shape in the layout, (ii) the height (h) or width (w) of the box is not within some specified range, *i.e.*, when h, w do not obey the following constraints $h_{low} \leq h \leq h_{high}$ and $w_{low} \leq w \leq w_{high}$.

Then all generated end-cut boxes between two shapes are divided into independent components IC (Line 10) based on finding connected components of a graph $G = (V, E)$ with $V = \{v_i\} =$ set of all end-cut boxes and $(v_i, v_j) \in E$, if v_i overlaps with v_j. The overlap between two end-cut boxes is classified into

Algorithm 6 Shape-edge dependent end-cut generation algorithm between two shapes S_1 and S_2

```
1: Procedure generateEndCut ($S_1$, $S_2$);
2: for all $se_1 \in$ edges($S_1$) do
3:     for all $se_2 \in$ edges($S_2$) do
4:         $ecBox$ = generateEndCutBox ($se_1$, $se_2$);
5:         if $ecBox \neq$ NULL then
6:             Store $ecBox$ in $ecBoxSet$;
7:         end if
8:     end for
9: end for
10: Divide $ecBoxSet$ into independent components ($IC$);
11: if $|ecBoxSet| = |V|$ then
12:     Print all boxes;
13:     return true;
14: end if
15: for all $ic \in IC$ do
16:     Remove corner-corner end-cut boxes overlapping with edge-edge end-
    cut box;
17:     if $\exists$ set of $type_2$ overlaps then
18:         Generate minimum area box;
19:     else
20:         Generate all end-cut boxes
21:     end if
22: end for
23: return true;
24: end Procedure
```

$type_1$ and $type_2$ overlaps. When two boxes overlap only in an edge or in a point but not in space, we call this $type_1$ overlap, where as the overlap in space is termed as $type_2$ overlap as shown in Fig. 2.36. Each of the $ic \in IC$ may contain multiple end-cut boxes. If the total number of end-cut-boxes ($|V|$) is equal to $|IC|$, that implies there is no overlap between the end-cut boxes and

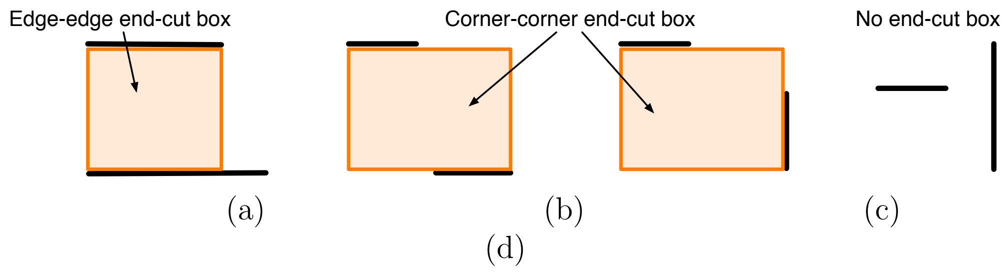


Figure 2.35: End-cut box generation between any two shape-edges

we generate all of them (Lines 11-14).

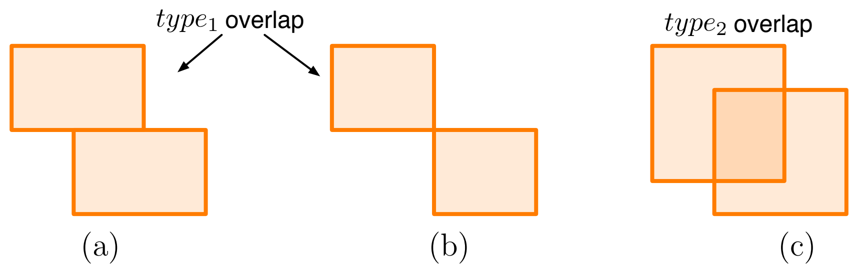


Figure 2.36: Types of overlaps between end-cut boxes

For multiple boxes in each ic, if there is an overlap between corner-corner and edge-edge end-cut boxes, the corner-corner end-cut box is removed (Line 16). After doing this, either there will be a set of $type_1$ overlaps or a set of $type_2$ overlaps in each ic. In case of $type_2$ overlaps, the end-cut box with the minimum area is chosen as shown in Fig. 2.37. For $type_1$ overlaps in each ic, all end-cut boxes are generated (Line 20).

2.4.2.2 ILP Formulations

After the construction of layout graph and end-cut graph, LELE-EC layout decomposition problem can be transferred into a coloring problem on

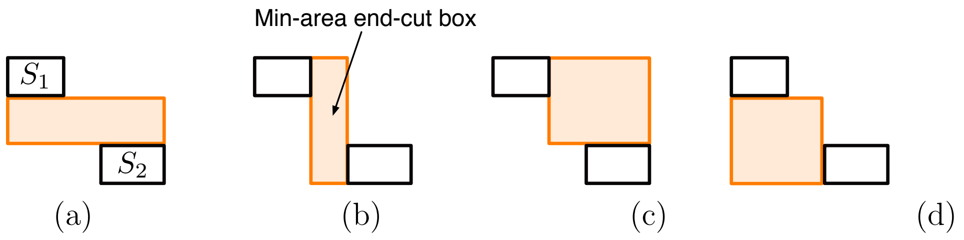


Figure 2.37: Minimum area end-cut box is chosen for $type_2$ overlaps

layout graph and a selection problem on end-cut graph. At the first glance the coloring problem is similar to that in LELE layout decomposition. However, since the conflict graph can not be guaranteed to be plannar, the face graph based methodology [99] cannot be applied here. Therefore, we formulate integer linear programming (ILP) to solve both coloring problem and selection problem simultaneously. For convenience, some notations in the ILP formulation are listed in Table 2.12.

Table 2.12: Notations in LELE-EC Layout Decomposition

CE	set of conflict edges
EE	set of end-cut conflict edges
SE	set of stitch edges.
n	number of input layout features.
r_i	the i_{th} layout feature
x_i	variable denoting the coloring of r_i
ec_{ij}	0-1 variable, $ec_{ij} = 1$ when a end-cut between r_i and r_j
c_{ij}	0-1 variable, $c_{ij} = 1$ when a conflict between r_i and r_j
s_{ij}	0-1 variable, $s_{ij} = 1$ when a stitch between r_i and r_j

First we discuss the ILP formulation when no stitch candidate is generated in layout graph. Given a set of input layout features $\{r_1, \ldots, r_n\}$, we construct layout graph, and end-cut graph. Every conflict edge e_{ij} is in CE, while each end-cut candidate ec_{ij} is in SE. x_i is a binary variable representing the color of r_i. c_{ij} is a binary variable for conflict edge $e_{ij} \in CE$. To minimize the conflict number, our objective function to minimize $\sum_{e_{ij} \in CE} c_{ij}$.

To evaluate the conflict number, we provide the following constraints:

$$\begin{cases} x_i + x_j \leq 1 + c_{ij} + ec_{ij} & \text{if } \exists\ ec_{ij} \in EE \\ (1 - x_i) + (1 - x_j) \leq 1 + c_{ij} + ec_{ij} & \text{if } \exists\ ec_{ij} \in EE \\ x_i + x_j \leq 1 + c_{ij} & \text{if } \nexists\ ec_{ij} \in EE \\ (1 - x_i) + (1 - x_j) \leq 1 + c_{ij} & \text{if } \nexists\ ec_{ij} \in EE \end{cases} \tag{2.12}$$

Here ec_{ij} is a binary variable for end-cut candidate. If there is no end-cut candidate between adjacent features r_i and r_j, if $x_i \neq x_j$ then one conflict would be reported ($c_{ij} = 1$). Otherwise, we will try to enable the end-cut candidate ec_{ij} first. If the end-cut candidate ec_{ij} cannot be applied ($ec_{ij} = 0$), then one conflict will be also reported.

If end-cuts ec_{ij} and ec_{pq} are conflict with each other, at most one of them will be applied. To enable this, we introduce the following constraint.

$$ec_{ij} + ec_{pq} \leq 1, \quad \forall e_{ijpq} \in EE \tag{2.13}$$

To forbid useless end-cut, we introduce the following constraints. That is, if features x_i and x_j are in different colors, $ec_{ij} = 0$.

$$\begin{cases} ec_{ij} + x_i - x_j \leq 1 & \forall e_{ij} \in CE \\ ec_{ij} + x_j - x_i \leq 1 & \forall e_{ij} \in CE \end{cases} \tag{2.14}$$

Therefore, without the stitch candidate, the LELE-EC layout decomposition can be formulated as shown in Eq. (2.15).

$$\begin{aligned} \min \sum_{e_{ij} \in CE} c_{ij} \\ \text{s.t.} \quad (2.12), (2.13), (2.14) \end{aligned} \tag{2.15}$$

If the stitch insertion is considered, the ILP formulation is as in Eq. (2.16). Here the objective is to simultaneously minimize both the conflict number and the stitch number. The parameter α is a user-defined parameter for assigning relative importance between the conflict number and the stitch number. The constraints (2.16a) - (2.16b) are used to calculate the stitch number.

$$\begin{aligned} \min \quad & \sum_{e_{ij} \in CE} c_{ij} + \alpha \times \sum_{e_{ij} \in SE} s_{ij} && && (2.16) \\ \text{s.t.} \quad & x_i - x_j \le s_{ij} && \forall e_{ij} \in SE && (2.16a) \\ & x_j - x_i \le s_{ij} && \forall e_{ij} \in SE && (2.16b) \\ & (2.12), (2.13), (2.14) \end{aligned}$$

2.4.2.3 Graph Simplification Techniques

ILP is a classical NP-hard problem, i.e., there is no polynomial time optimal algorithm to solve it [24]. Therefore, for large layout cases, solving ILP may suffer from long runtime penalty to achieve the results. In this section, we provide a set of speed-up techniques. Note that these techniques can keep optimality. In other words, with these speed-up techniques, ILP formulation can achieve the same results comparing with those not applying speed-up.

The first speedup technique is called **independent component computation**. By breaking down the whole layout graph into several independent components, we partition the initial layout graph into several small ones. Then each component can be resolved through ILP formulation independently. At last, the overall solution can be taken as the union of all the components without affecting the global optimality. Note that this is a well-known technique which has been applied in many previous studies (e.g., Ref. [48, 100, 111]).

Our second technique is called **Bridge Computation**. A bridge of a graph is an edge whose removal disconnects the graph into two components. If the two components are independent, removing the bridge can divide the whole problem into two independent sub-problems. We search all bridge edges in layout graph, then divide the whole layout graph through these bridges. Note that all bridges can be found in $O(|V| + |E|)$, where $|V|$ is the vertex number, and $|E|$ is the edge number in the layout graph.

Our third technique is called **End-Cut Pre-Selection**. Some end-cut candidates have no conflict end-cuts. For the end-cut candidate ec_{ij} that has no conflict end-cut, it would be pre-selected in final decomposition results. That is, the features r_i and r_j are merged into one feature. By this way, the problem size of ILP formulation can be further reduced. End-cut pre-selection can be finished in linear time.

2.4.3 Experimental Results

We implement our algorithms in C++ and test on an Intel Xeon 3.0GHz Linux machine with 32G RAM. 15 benchmark circuits from Ref. [109] are used. GUROBI [43] is chosen as the ILP solver. The minimum coloring spacing min_s is set as 120 for the first ten cases and as 100 for the last five cases, as in Ref. [109] and Ref. [32]. The width threshold w_{th}, which is used in end-cut candidate generation, is set as dis_m.

In the first experiment, we show the decomposition results of the ILP formulation. Table 2.13 compares two ILP formulations "**ILP w/o. stitch**" and "**ILP w. stitch**". Here "ILP w/o. stitch" is the ILP formulation based on the graph without stitch edges, while "ILP w. stitch" considers the stitch insertion in the ILP. Note that all speed-up techniques are applied to both. Columns "Wire#" and "Comp#" reports the total feature number, and the divided component number, respectively. For each method we report the conflict number, stitch number, and computational time in seconds("CPU(s)"). "Cost" is the cost function, which is set as conflict# $+0.1\times$ stitch#.

From Table 2.13 we can see that compared with "ILP w/o. stitch", when stitch candidates are considered in the ILP formulation, the cost can be reduced by 2%, while the runtime is increased by 5%. It shall be noted that stitch insertion has been shown to be an effective method to reduce the cost for both LELE layout decomposition and LELELE layout decomposition. However, we can see that for LELE-EC layout decomposition, stitch insertion is not that effective. In addition, due to the overlap variation derived from

Table 2.13: Comparison between w. stitch and w/o. stitch

Circuit	Wire#	Comp#	ILP w/o. stitch				ILP w. stitch			
			conflict#	stitch#	cost	CPU(s)	conflict#	stitch#	cost	CPU(s)
C1	1109	123	1	0	1	1.19	1	0	1	1.32
C2	2216	175	1	0	1	2.17	1	0	1	2.89
C3	2411	270	0	0	0	3.11	0	0	0	3.62
C4	3262	467	0	0	0	3.2	0	0	0	3.75
C5	5125	507	4	0	4	8.72	4	0	4	8.81
C6	7933	614	1	0	1	11.24	1	0	1	11.1
C7	10189	827	2	0	2	14.57	2	0	2	15.98
C8	14603	1154	2	0	2	21.2	2	0	2	23.07
C9	14575	2325	23	0	23	24.36	12	12	13.2	28.06
C10	21253	1783	7	0	7	28.42	7	0	7	32.02
S1	4611	272	0	0	0	6.23	0	0	0	7.04
S2	67696	5116	166	0	166	179.05	166	1	166.1	218.37
S3	157455	15176	543	0	543	506.55	530	13	531.3	563.65
S4	168319	15354	443	0	443	464.84	436	7	436.7	494.4
S5	159952	12626	419	0	419	464.11	415	6	415.6	514.56
avg.	-	-	107.5	0	107.5	115.9	105.1	2.6	105.4	128.6
ratio	-	-	-	-	**1.0**	**1.0**	-	-	**0.98**	**1.10**

stitch, stitch insertion for LELE-EC may not be an effective method.

In the second experiment, we analyze the effectiveness of the proposed speed-up techniques. Fig. 2.38 compares two ILP formulations "**w/o. stitch w/o. speedup**" and "**w/o. stitch w. speedup**", where "w/o. stitch w/o. speedup" only applies independent component computation, while "w. speedup" involves all three speed-up techniques. Besides, none of them consider the stitch in layout graph. From Fig. 2.38 we can see that with speed-up techniques (bridge computation and end-cut pre-selection), the runtime can be reduced by around 60%.

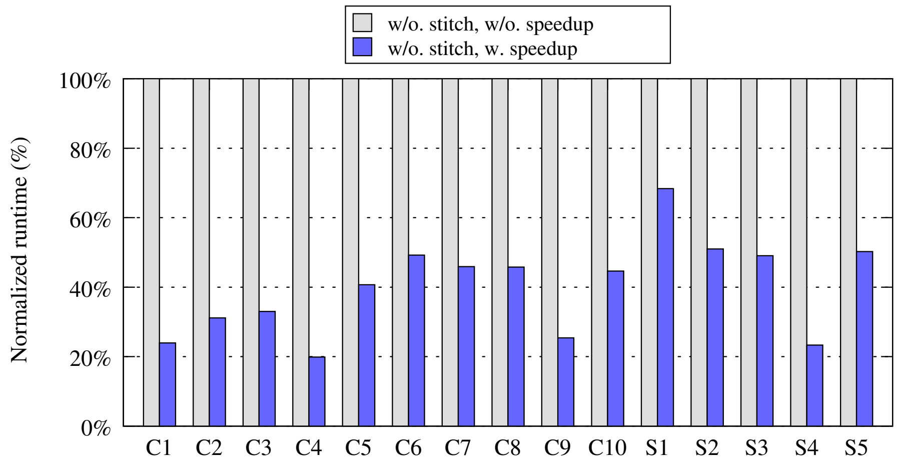


Figure 2.38: Effectiveness of Speed-up techniques when no stitch is introduced.

Fig. 2.39 demonstrates the similar effectiveness of speed-up techniques between "**w. stitch w. speedup**" and "**w/o. stitch w. speedup**". Here stitch candidates are introduced in layout graph. We can see that for these two ILP formulation the bridge computation and the end-cut pre-selection can reduce runtime by around 56%.

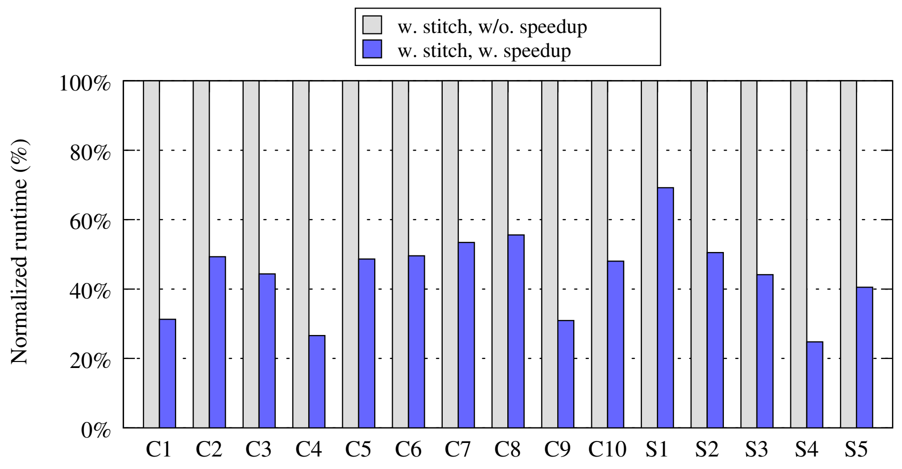


Figure 2.39: Effectiveness of Speed-up techniques when stitch is introduced.

Fig. 2.40 shows four conflict examples in decomposed layout, where conflict pairs are labeled with red arrows. We can observe that some conflicts (see Fig. 2.40 (a) and (c)) are introduced due to the end-cuts existing in neighboring. For these two cases, the possible reason is the patterns are irregular, therefore some end-cuts that close to each other cannot be merged into a larger one. We can also observe some conflicts (see Fig. 2.40 (b) and (d)) come from via shapes. For these two cases, one possible reason is that it is hard to find end-cut candidates around via, comparing with long wires.

2.4.4 Summary

In this section we have proposed an improved framework and algorithms to solve the LELE-EC layout decomposition problem. New end-cut candidates are generated considering potential hotspots. The layout decomposition is formulated as an integer linear programming (ILP). The experimental results

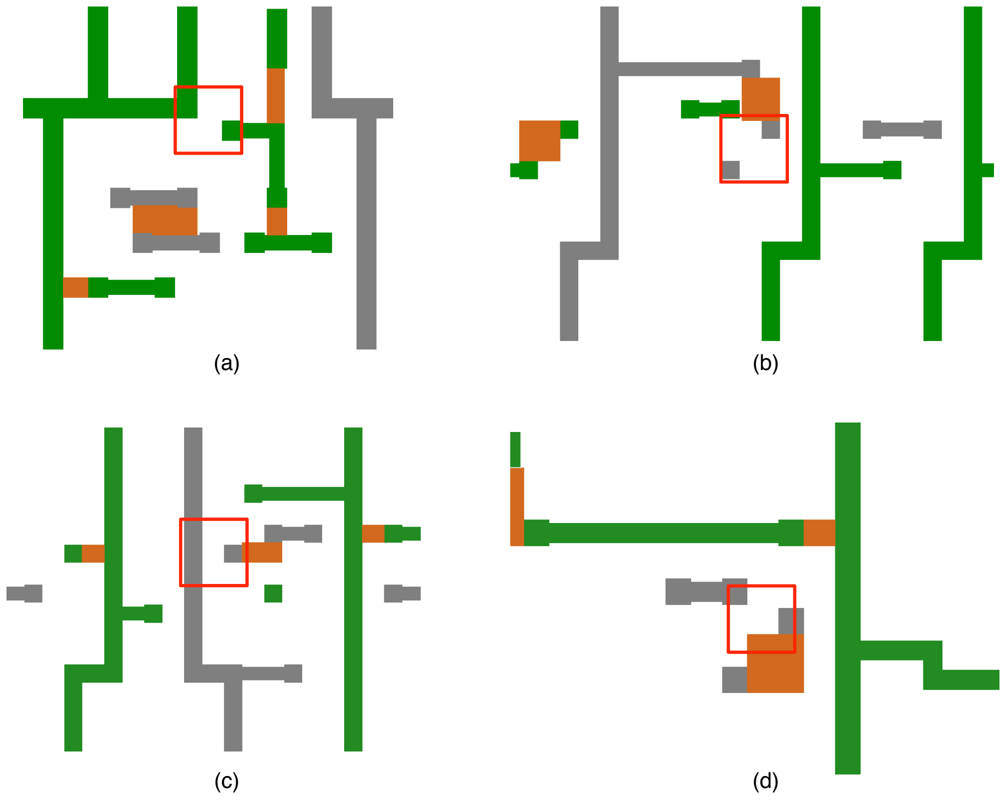


Figure 2.40: Conflict examples in decomposed results. (a)(c) Conflicts because no additional end-cut can be inserted due to the existing neighboring end-cuts. (b)(d) Conflicts because no end-cut candidates between irregular vias.

show the effectiveness of our algorithms. It shall be noted that our framework is very generic that it can provide high quality solutions to both uni-directional and bi-directional layout patterns. However, if all the layout patterns are uni-directional, there might be some faster solutions. Since end-cutting can provide better printability than traditional LELELE process, we expect to see more works on the LELE-EC layout decomposition and LELE-EC aware design.

2.5 Layout Decomposition for Quadruple Patterning and Beyound

The process of QPL brings up several critical yet open design challenges, such as layout decomposition, where the original layout is divided into four masks (colors). However, how to effectively solve the quadruple patterning, or even general multiple patterning problems, is still an open question. In this section, we deal with the quadruple patterning layout decomposition (QPLD) problem. Our contributions are highlighted as follows. (1) To our best knowledge, this is the first layout decomposition research for QPLD problem. We believe this work will invoke more future research into this field thereby promoting the scaling of technology node. (2) Our framework consists of holistic algorithmic processes, such as semidefinite programming based algorithm, linear color assignment, and novel GH-tree based graph division. (3) We demonstrate the viability of our algorithm to suit with general K-patterning (K$\geq$4) layout decomposition, which could be advanced guidelines for future technology.

2.5.1 Preliminaries and Problem Formulation

2.5.1.1 Why Quadruple Patterning

Quadruple patterning lithography (QPL) is a natural extension along the paradigm of double/triple patterning. In the QPL manufacturing, there are four exposure/etching processes, through which the initial layout can be produced. Compared with triple patterning lithography, QPL introduces one more mask. Although increasing the number of processing steps by 33% over triple patterning, there are several reasons/advantages for QPL. Firstly, due to the delay or uncertainty of other lithography techniques, such as EUV, semiconductor industry needs CAD tools to be prepared and understand the complexity/implication of QPL. Even from theoretical perspective, studying the general multiple patterning is valuable. Secondly, it is observed that for triple patterning lithography, even with stitch insertion, there are several common native conflict patterns. As shown in Fig. 2.41 (a), contact layout within the standard cell may generate some 4-clique patterns, which are indecomposable. This conflict can be easily resolved if four masks are available (see Fig. 2.41 (b)). Thirdly, with one more mask, some stitches may be avoided during manufacturing. By this way it is potential to resolve the overlapping and yield issues derived from the stitches.

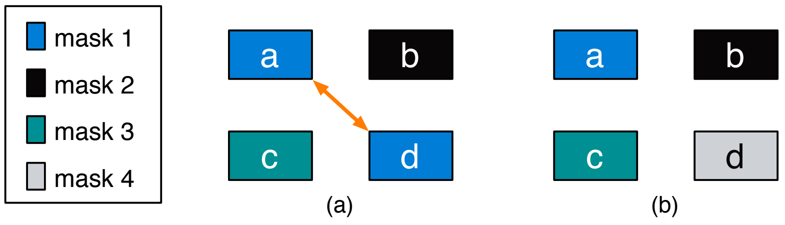


Figure 2.41: (a) A common native conflict from triple patterning lithography; (b) The conflict can be resolved through quadruple patterning lithography.

2.5.1.2 Problem Formulation

Given input layout which is specified by features in polygonal shapes, a *decomposition graphs* [109, 111] is constructed by Definition 2.

Now we give the problem formulation of quadruple patterning layout decomposition (QPLD).

Problem 4 (QPLD). *Given an input layout which is specified by features in polygonal shapes and minimum coloring distance min_s, the decomposition graph is constructed. Quadruple patterning layout decomposition (QPLD) assigns all the vertices into one of four colors (masks) to minimize conflict number and stitch number.*

The QPLD problem can be extended to general K-patterning layout decomposition problem as follows.

Problem 5 (K-Patterning Layout Decomposition).
Given an input layout, the decomposition graph is constructed. Each vertex in graph would be assigned into one of K colors (masks) to minimize conflict number and stitch number.

2.5.2 Algorithms

The overall flow of our layout decomposition is summarized in Fig. 2.42. We first construct decomposition graph to transform the original geometric patterns into a graph model. By this way, the QPLD problem can be formulated as 4 coloring on the decomposition graph. To reduce the problem size, graph division techniques (see Section 2.5.2.3) are applied to partition the graph into a set of components. Then the color assignment problem can be solved independently for each component, to minimize both the conflict number and the stitch number. In this following, we propose two color assignment

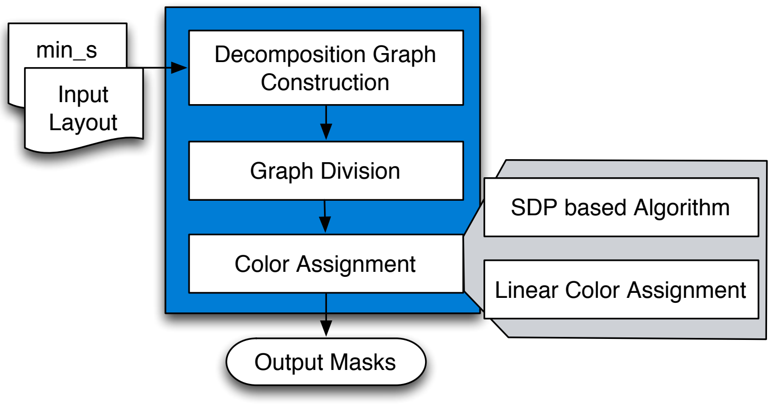


Figure 2.42: Proposed layout decomposition flow.

algorithms, i.e., semidefinite programming (SDP) based algorithm, and linear color assignment.

2.5.2.1 SDP Based Color Assignment

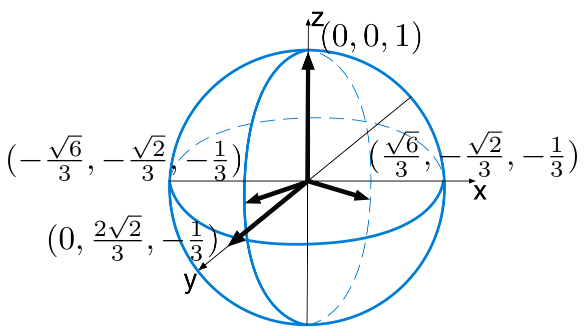

z
$(0, 0, 1)$
$(-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$
$(\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$
x
$(0, \frac{2\sqrt{2}}{3}, -\frac{1}{3})$
y

Figure 2.43: Four vectors correspond to four colors.

Semidefinite programming (SDP) has been successfully applied to triple patterning layout decomposition [105, 109]. Here we will show that SDP formulation can be extended to solve QPLD problem. To represent four different colors (masks), as illustrated in Fig. 2.43, four unit vectors are introduced [54]: $(0,0,1)$, $(0,\frac{2\sqrt{2}}{3},-\frac{1}{3})$, $(\frac{\sqrt{6}}{3},-\frac{\sqrt{2}}{3},-\frac{1}{3})$ and $(-\frac{\sqrt{6}}{3},-\frac{\sqrt{2}}{3},-\frac{1}{3})$. We construct the vectors in such a way that inner product for any two vectors $\vec{v_i}, \vec{v_j}$ satisfying: $\vec{v_i}\cdot\vec{v_j} = 1$ if $\vec{v_i} = \vec{v_j}$; $\vec{v_i}\cdot\vec{v_j} = -\frac{1}{3}$ if $\vec{v_i} \neq \vec{v_j}$.

Based on the vector definition, the QPLD problem can be formulated as the following vector programming:

$$\begin{aligned}
\min \sum_{e_{ij}\in CE} \frac{3}{4}(\vec{v_i}\cdot\vec{v_j} + \frac{1}{3}) + \frac{3\alpha}{4} \cdot \sum_{e_{ij}\in SE} (1 - \vec{v_i}\cdot\vec{v_j}) \qquad (2.17)\\
\text{s.t.} \quad \vec{v_i} \in \{(0,0,1),(0,\frac{2\sqrt{2}}{3},-\frac{1}{3}),(\frac{\sqrt{6}}{3},-\frac{\sqrt{2}}{3},-\frac{1}{3}),\\
(-\frac{\sqrt{6}}{3},-\frac{\sqrt{2}}{3},-\frac{1}{3})\}
\end{aligned}$$

where the objective function is to minimize the conflict number and the stitch number. α is a user-defined parameter, which is set as 0.1 in this work. After relaxing the discrete constraints in (2.17) and removing the constant in objective function, we redraw the following semidefinite programming (SDP) formulation.

$$\begin{aligned}
\min \sum_{e_{ij}\in CE} \vec{v_i}\cdot\vec{v_j} - \alpha \sum_{e_{ij}\in SE} \vec{v_i}\cdot\vec{v_j} \qquad (2.18)\\
\text{s.t.} \quad \vec{v_i}\cdot\vec{v_i} = 1, \quad \forall i \in V\\
\vec{v_i}\cdot\vec{v_j} \geq -\frac{1}{3}, \quad \forall e_{ij} \in CE
\end{aligned}$$

After solving the SDP, we get a set of continuous solutions in matrix X, where each value x_{ij} in matrix X corresponds to $v_i \cdot v_j$. If x_{ij} is close to

1, vertices v_i, v_j are tend to be in the same mask (color). A greedy mapping algorithm [109] can be directly applied here to get color assignment solution. However, the performance of greedy method may not be good.

Algorithm 7 SDP + Backtrack

```
Require: SDP solution $x_{ij}$, threshold value $t_{th}$;
 1: for all $x_{ij} \geq t_{th}$ do
 2:     Combine vertices $v_i, v_j$ into one larger vertex;
 3: end for
 4: Construct merged graph $G' = \{V', CE', SE'\}$;
 5: BACKTRACK$(0, G')$;
 6: return color assignment result in $G'$;

 7: function BACKTRACK$(t, G')$
 8:     if t $\geq$ size$[G']$ then
 9:         if  Find a better color assignment  then
10:             Store current color assignment;
11:         end if
12:     else
13:         for all legal color $c$ do;
14:             $G'[t] \leftarrow c$;
15:             BACKTRACK$(t + 1, G')$;
16:             $G'[t] \leftarrow -1$;
17:         end for
18:     end if
19: end function
```

To overcome the limitation of the greedy method, in our framework a backtrack based algorithm (see Algorithm 7) is proposed to consider both SDP results and graph information. The backtrack based method accepts two arguments of the SDP solution $\{x_{ij}\}$ and a threshold value t_{th}. In our work t_{th} is set as 0.9. As discussed above, if x_{ij} is close to be 1, two vertices v_i and v_j tend to be in the same color (mask). Therefore, we scan all pairs, and combine some vertices into one larger vertex (lines $1-3$). After the combination, the vertex number can be reduced, thus the graph has be simplified (line 4). The simplified graph is called *merged graph* [105]. On the merged graph, *BACKTRACK* algorithm is presented to search an optimal color assignment (lines $7-19$).

2.5.2.2 Linear Color Assignment

Backtrack based method may still involve runtime overhead, especially for complex case where SDP solution cannot provide enough merging candidates. Therefore, an efficient color assignment is required. At first glance, the color assignment for quadruple patterning can be solved through *four color map theorem* [10] that every planar graph is 4-colorable. However, in emerging technology node, the designs are so complex that we observe many K_5 or $K_{3,3}$ structures, where K_5 is the complete graph on five vertices, while $K_{3,3}$ is the complete bipartite graph on six vertices. Due to *Kuratowski's theorem* [58], the decomposition graph is not planar, thus classical four coloring technique [80] is hard to be applied.

Here we propose an efficient color assignment algorithm. Note that our method is targeting general graph, not just planar graph. In addition, different from classical four coloring method that needs quadratic runtime [80], our color assignment is a linear runtime algorithm.

The details of linear color assignment are summarized in Algorithm 8, which involves three stages. The first stage is iteratively vertex removal. For

Algorithm 8 Linear Color Assignment

```
Require: Decomposition graph $G = \{V, CE, SE\}$, Stack $S$;
1: while $\exists v_i \in V$ s.t. $d_{conf}(v_i) < 4$ & $d_{stit}(v_i) < 2$ do
2:     $S$.push($v_i$);
3:     $G$.delete($v_i$);
4: end while
5: Construct vector $vec$;
6: C1 = SEQUENCE-COLORING($vec$);
7: C2 = DEGREE-COLORING($vec$);
8: C3 = 3ROUND-COLORING($vec$);
9: C = best coloring solution among {C1, C2, C3};
10: POST-REFINEMENT($vec$);
11: while !$S$.empty() do
12:     $v_i = S$.pop();
13:     $G$.add($v_i$);
14:     $c(v_i) \leftarrow$ a legal color;
15: end while
```

each vertex v_i, we denote its conflict degree $d_{conf}(v_i)$ as number of conflict edges incident to v_i, while its stitch degree $d_{stit}(v_i)$ as number of stitch edges. The main idea is that the vertices with conflict degree less than 4 and stitch degree less than 2 are identified as non-critical, thus can be temporarily removed and pushed into stack S (lines 1-4). After coloring remaining vertices, each vertex in stack S would be pop up one by one and assigned one legal color (lines 11-15). This strategy is safe in terms of conflict number. In other words, when a vertex is pop up from S, there is always one color available without introducing new conflict.

In the second stage (lines 5-9), all remaining vertices would be assigned colors one by one. However, color assignment through one specific order may be stuck at *local optimum* which stems from the greedy nature. For example,

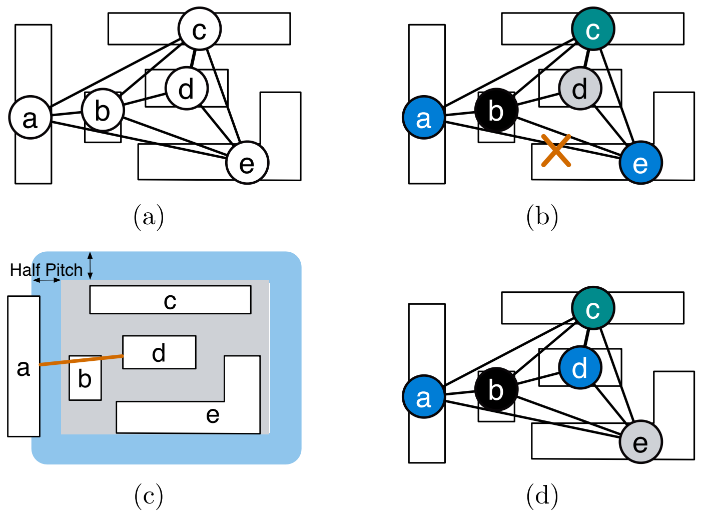


Figure 2.44: (a) Decomposition graph; (b) Greedy coloring with one conflict; (c) a is detected as color-friendly to d; (d) Coloring considering color-friendly rules.

given a decomposition graph in Fig. 2.44 (a), if the coloring order is a-b-c-d-e, when vertex d is greedily selected grey color, the following vertex e cannot find any color without conflict (see Fig. 2.44 (b)). In other words, vertex ordering significantly impacts the coloring result.

To alleviate the impact of vertex ordering, two strategies are proposed. The first strategy is called **color-friendly rules**, as in Definition 5. In Fig. 2.44 (c), all conflict neighbors of pattern d are labeled inside a grey box. Since the distance between a and d is within the range of $(min_s, min_s + hp)$, a is color-friendly to d. Interestingly, we discover a rule that for a complex/dense layout, color-friendly patterns tend to be with the same color. Based on these rules, during linear color assignment, to determine one vertex color, instead of just comparing its conflict/stitch neighbors, the colors of its color-friendly vertices would also be considered. Detecting color-friendly vertices is similar to the conflict neighbor detection, thus it can be finished during decomposition graph construction without much additional efforts.

Definition 5 (Color-Friendly)**.** *A pattern a is color-friendly to pattern b, iff their distance is larger than* min_s*, but smaller than* $min_s + hp$*. Here hp is the half pitch.*

Our second strategy is called **peer selection**, where three different vertex orders would be processed simultaneously, and the best one would be selected as the final coloring solution (lines 6-8). Although color assignment is solved thrice, since for each order the coloring is in linear time, the total computational time is still linear.

In the third stage (line 10), post-refinement greedily checks each vertex to see whether the solution can be further improved.

1. **SEQUENCE-COLORING**: Vertices are assigned colors based on the initial order.

2. **DEGREE-COLORING**: Vertices are assigned colors based on a nearly conflict degree descending order.

3. **3ROUND-COLORING**: Vertices are solved through conflict degree descending order. In addition, the vertex with only one legal possible color would be assigned first.

Vector *vec* is constructed to provide the vertices with conflict degree descending order. Instead of completely sorting all vertices which needs $O(nlogn)$, vector *vec* contains $vec[1]$, $vec[2]$, and $vec[3]$, successively. The vector vec[i], $1 \leq i \leq 3$, contains vertices with special conflict degree, defined as follows:

$$\begin{aligned} vec[1] =& \{v_i | \forall v_i \in V, d_{conf}(v_i) > 6\} \\ vec[2] =& \{v_i | \forall v_i \in V, 4 < d_{conf}(v_i) <= 6\} \\ vec[3] =& \{v_i | \forall v_i \in V, d_{conf}(v_i) <= 4\} \end{aligned}$$

The main idea is that the vertices in *vec* are nearly conflict degree descending order, but the construction is in linear time.

The details of the method *3ROUND-COLORING* are shown in Algorithm 9, where the vertex with less coloring choices tends to be resolved first. In other words, we prefer to assign color to a vertex with "less flexibility". At the beginning, all vertices are labeled as UNSOLVED (line 1), then the vector *vec* is scanned thrice. In the first round scanning, for each vertex $v_i \in vec$, a greedy color with minimum cost would be assigned (lines 2-4). When all four colors have been applied at least once, the first round is stopped. In the second round scanning, only those vertex with only one legal color would be assigned color. By this way, we prefer to assign colors to those vertices with "less flexibility". When a vertex is assigned one color, it is labeled as SOLVED. In the third scanning, we greedily assign colors to those UNSOLVED vertices.

For a decomposition graph with color-friendly information and n vertices, in the first stage vertex removal/pop up can be finished in $O(n)$. In the

Algorithm 9 3ROUND-COLORING(vec)

```
Require: Vector $vec$ containing all vertices;
 1: Label each $v_i \in vec$ as UNSOLVED;
 2: for all vertex $v_i \in vec$ do                              ▷ 1st round
 3:     $c(v_i) \leftarrow$ a minimum cost color;
 4:     Label $v_i$ as SOLVED;
 5:     if All four colors have been applied at least once then
 6:         Break;                                           ▷ End 1st round
 7:     end if
 8: end for
 9: for all UNSOLVED vertex $v_i \in vec$ do                     ▷ 2nd round
10:     if $v_i$ has only one legal color c then
11:         $c(v_i) \leftarrow$ c;
12:         Label $v_i$ as SOLVED;
13:     end if
14: end for
15: for all UNSOLVED vertex $v_i \in vec$ do                     ▷ 3rd round
16:     $c(v_i) \leftarrow$ a minimum cost color;
17:     Label $v_i$ as SOLVED;
18: end for
19: return $c(v_i), \forall v_i \in vec$;
```

second stage, as mentioned above the coloring needs $O(n)$. In post-refinement stage, all vertices are traveled once, which requires $O(n)$ time. Therefore, the total complexity is $O(n)$.

2.5.2.3 GH-Tree based 3-Cut Removal

Graph division is a technique that partitions the whole decomposition graph into a set of components, then the color assignment on each component can be solved independently. In our framework, the techniques extended from previous work are summarized as follows, (1) Independent Component

Computation [32,56,89,92,99,105,109,111]; (2) Vertex with Degree Less than 3 Removal [32, 56, 105, 109] [2]; (3) 2-Vertex-Connected Component Computation [32, 56, 105].

Another technique, *cut removal*, has been proven powerful in double/patterning layout decomposition [32, 89, 109]. A cut of a graph is an edge whose removal disconnects the graph into two components. The definition of cut can be extended to 2-cur (3-cut), which is a double (triplet) of edges whose removal would disconnect the graph. However, different from the 1-cut and 2-cut detection that can be finished in linear time [32], 3-cut detection is much more complicated. Here we propose an effective 3-cut detection method. Besides, our method can be easily extended to detect any K-cut (K ≥ 3).

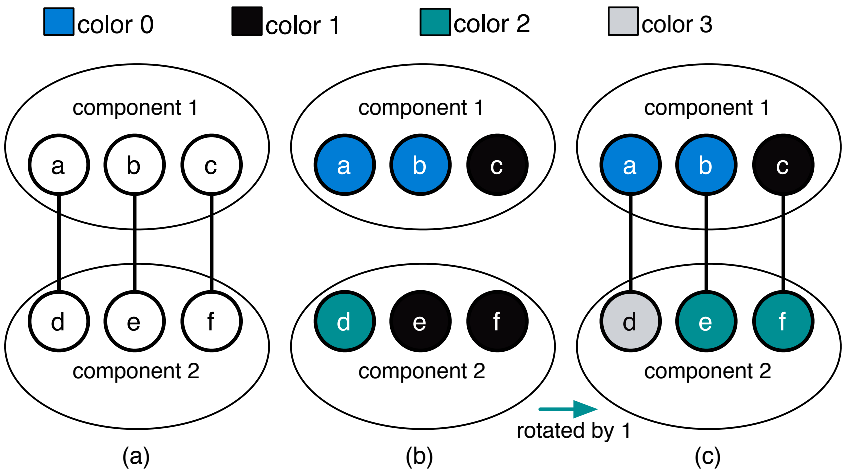


Figure 2.45: An example of 3-cut detection and removal.

Fig . 2.45 (a) shows a graph with a 3-cut $(a-d, b-e, c-f)$, and two

[2]In QPLD problem, the vertices with degree less than 4 would be detected and removed temporally.

components can be derived by removing this 3-cut. After color assignment on two components, for each cut edge, if the colors of the two endpoints are different, the two components can be merged directly. Otherwise, a *color rotation* operation is required to one component. For vertex v in graph, we denote $c(v)$ as its color, where $c(v) \in \{0, 1, 2, 3\}$. Vertex v is said to be rotated by i, if $c(v)$ is changed to $(c(v) + i)\%4$. It is easy to see that all vertices in one component should be rotated by the same value, so no additional conflict is introduced within the component. An example of such color rotation operation is illustrated in Fig. 2.45 (b)-(c), where conflict between vertices c, f would be removed to interconnect two components together. Here all the vertices in component 2 are rotated by 1 (see Fig. 2.45 (c)). We have the following Lemma:

Lemma 4. *In QPLD problem, color rotation after interconnecting 3-cut does not increase the conflict number.*

Proof. We denote the three edges in a 3-cut as $(a_1 - b_1, a_2 - b_2, a_3 - b_3)$. Without loss of generality, we rotate the colors of b_1, b_2, b_3 to remove any conflict derived on the edges. Since all vertices should be rotated by the same value, there are four rotation options for the whole component. For one edge $\{a_1 - b_1\}$, one option is infeasible that would causes one conflict. At most three options are infeasible on the 3-cut. Therefore, at least one option left is feasible without any conflict introduced on the 3-cut. $\square$

In addition, to detect all 3-cuts, we have the following Lemma:

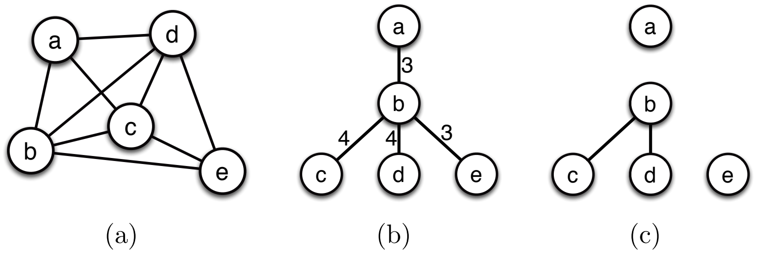


Figure 2.46: (a) Decomposition graph; (b) Corresponding GH-tree; (c) Components after 3-cut removal.

Lemma 5. *If the minimum cut between two vertices v_i and v_j is less than 4, then v_i, v_j belong to different components that divided by a 3-cut.*

Based on Lemma 5, we can see that if the cut between vertices v_i, v_j is larger or equal to 4 edges, v_i, v_j should belong to the same component. One straightforward 3-cut detection method is to compute the minimum cuts for all the $\{s-t\}$ pairs. However, for a decomposition graph with n vertices, there are $n(n-1)/2$ pairs of vertices. Computing all these cut pairs may spend too long runtime, which is impractical for complex layout design.

Gomory and Hu [40] showed that the cut values between all the pairs of vertices can be computed by solving only $n-1$ network flow problems on graph G. Furthermore, they showed that the flow values can be represented by a weighted tree T on the n vertices, where for any pair of vertices (v_i, v_j), if e is the minimum weight edge on the path from v_i to v_j in T, then the minimum cut value from v_i to v_j in G is exactly the weight of e. Such a weighted tree T is called Gomory-Hu tree (GH-tree). For example, given the decomposition graph in Fig. 2.46 (a), the corresponding GH-tree is shown in Fig. 2.46 (b), where the value on edge e_{ij} is the minimum cut number between vertices v_i and v_j. Because of Lemma 5, to divide the graph through 3-cut removal, all the edges with value less than 4 would be removed. The final three components are in Fig. 2.46 (c).

Algorithm 10 GH-tree based 3-Cut Removal

```
Require: Decomposition graph $G = \{V, CE, SE\}$;
1: Construct GH-tree as in [44];
2: Remove the edges with weight $< 4$;
3: Compute connected components on remaining GH-tree;
4: for each component do
5:     Color assignment on this component;
6: end for
7: Color rotation to interconnect all components;
```

The procedure of the 3-cut removal is shown in Algorithm 10. Firstly we construct GH-tree based on the algorithm by [44] (line 1). Dinic's blocking flow algorithm [27] is applied to help GH-tree construction. Then all edges in the GH-tree with weights less than four are removed (line 2). After solving the connected component problem (line 3), we can assign colors to each component separately (lines $4 - 5$). At last color rotation is applied to interconnect all 3-cuts back (line 6).

2.5.3 Experimental Results

We implemented the proposed layout decomposition algorithms in C++, and tested on a Linux machine with 2.9GHz CPU. We choose GUROBI [43] as the integer linear programming (ILP) solver, and CSDP [15] as the SDP solver. The benchmarks in [32, 109] are used as our test cases. We scale down the Metal1 layer to 20nm half pitch. Both the minimum feature width w_m and the minimum spacing between features s_m are 20nm. From Fig. 2.47 we can see that when minimum coloring distance $min_s = 2 \cdot s_m + w_m = 60$nm, even one dimension regular patterns can be a K_5 structure, which is not 4-colorable or planar [58]. In our experiments, for quadruple patterning min_s is set as $2 \cdot s_m + 2 \cdot w_m = 80$nm, while for pentuple patterning min_s is set as $3 \cdot s_m + 2.5 \cdot w_m = 110$nm. When larger min_s is applied, there are too many native conflicts in layouts, as the benchmarks are not multiple patterning friendly.

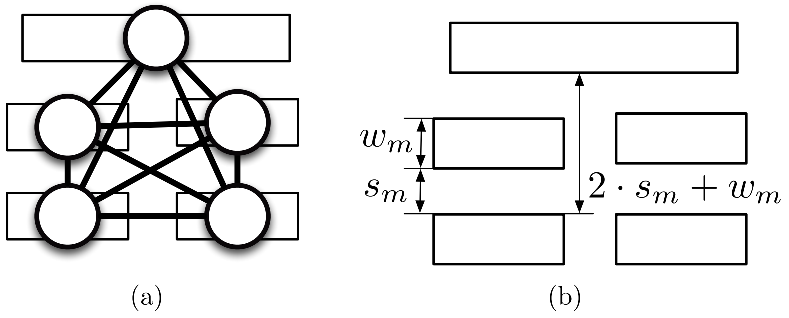


Figure 2.47: $min_s = 2 \cdot s_m + w_m$ may cause K_5 structure.

First we compare different color assignment algorithms for quadruple patterning, and the results are listed in Table 2.14. “**ILP**”, “**SDP+Backtrack**”, “**SDP+Greedy**” and “**Linear**” denote ILP formulation, SDP followed by backtrack mapping (Section 2.5.2.1), SDP followed by greedy mapping, and linear color assignment (Section 2.5.2.2), respectively. Here we implement an ILP formulation extended from the triple patterning work [109]. In SDP+Greedy, a greedy mapping from [109] is applied. All the graph division techniques, in-

Table 2.14: Comparison for Quadruple Patterning

Circuit	ILP			SDP+Backtrack			SDP+Greedy			Linear		
	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)
C432	2	0	0.6	2	0	0.24	2	0	0.02	2	1	0.001
C499	1	4	0.7	1	4	0.16	1	4	0.05	1	4	0.001
C880	1	0	0.3	1	0	0.02	1	0	0.02	1	2	0.001
C1355	0	4	0.6	0	4	0.1	0	4	0.04	0	4	0.001
C1908	2	3	1.0	2	3	0.28	2	3	0.09	2	4	0.001
C2670	0	6	1.1	0	6	0.16	0	6	0.1	0	7	0.001
C3540	1	3	1.1	1	3	0.09	2	2	0.05	1	3	0.001
C5315	1	13	2.8	1	13	0.6	2	12	0.24	1	15	0.002
C6288	9	0	2.3	9	0	0.36	9	0	0.17	9	1	0.001
C7552	2	13	3.4	2	13	0.6	3	12	0.22	2	18	0.003
S1488	0	6	0.7	0	6	0.05	4	2	0.01	0	6	0.001
S38417	20	549	1226.7	20	551	6.6	142	429	2.7	21	576	0.03
S35932	N/A	N/A	>3600	50	1745	28.7	460	1338	16.4	64	1927	0.15
S38584	N/A	N/A	>3600	41	1653	21.1	470	1224	10.4	47	1744	0.12
S15850	N/A	N/A	>3600	42	1462	18	420	1084	7.8	48	1571	0.11
avg.	-	-	>802.7	11.5	364.0	5.14	101.2	274.7	2.56	13.3	392.2	0.03
ratio	-	-	**>156.3**	**1.0**	**1.0**	**1.0**	**8.83**	**0.75**	**0.49**	**1.15**	**1.08**	**0.005**

Table 2.15: Comparison for Pentuple Patterning

Circuit	SDP+Backtrack			SDP+Greedy			Linear		
	cn#	st#	CPU(s)	cn#	st#	CPU(s)	cn#	st#	CPU(s)
C6288	19	2	2.4	19	2	0.49	19	5	0.005
C7552	1	1	0.3	1	1	0.05	1	4	0.001
S38417	0	4	1.45	0	4	0.21	0	4	0.001
S35932	5	20	8.11	5	20	0.62	5	25	0.009
S38584	3	4	1.66	7	3	0.3	3	6	0.008
S15850	6	5	2.7	7	5	0.4	5	15	0.007
avg.	5.7	6.0	2.77	6.5	5.83	0.35	5.5	9.8	0.005
ratio	**1.0**	**1.0**	**1.0**	**1.15**	**0.97**	**0.12**	**0.97**	**1.64**	**0.002**

cluding GH-tree based division, are applied. The columns "cn#" and "st#" denote the conflict number and the stitch number, respectively. Column "CPU(s)" is color assignment time in seconds.

From Table 2.14 we can see that for small cases the ILP formulation can achieve best performance in terms of conflict number and stitch number. However, for large cases (S38417, S35932, S38584, S15850) ILP suffers from long runtime problem that none of them can be finished in one hour. Compared with ILP, SDP+Backtrack can achieve near-optimal solutions, i.e., in every case the conflict number is optimal, while only in one case 2 more stitches are introduced. SDP+Greedy method can achieve 2× speedup against SDP+Backtrack. But the performance of SDP+Greedy is not good that for complex designs hundreds of additional conflicts are reported. The linear color assignment can achieve around 200× speedup against SDP+Backtrack, while only 15% more conflicts and 8% more stitches are reported.

We further compare the algorithms for pentuple patterning, that is,

$K = 5$. To our best knowledge there is no exact ILP formulation for pentuple patterning in literature. Therefore we consider three baselines, i.e., SDP+Backtrack, SDP+Greedy, and Linear. All the graph division techniques are applied. Table 2.15 evaluates six most dense cases. We can see that compared with SDP+Backtrack, SDP+Greedy can achieve around 8× speedup, but 15% more conflicts are reported. In terms of runtime, linear color assignment can achieve 500× and 60× speedup, against SDP+Backtrack and SDP+Greedy, respectively. In terms of performance, linear color assignment reports the best conflict number minimization, but more stitches may be introduced.

Interestingly, we observe that when a layout is multiple patterning friendly, color-friendly rules can provide a good guideline, thus linear color assignment can achieve high performance in terms of conflict number. However, when a layout is very complex or involving many native conflicts, linear color assignment reports more conflicts than SDP+Backtrack. One possible reason is that the color-friendly rules are not good in modeling global conflict minimization, but both SDP and backtrack provide a global view.

At last, Fig. 2.48 and Fig. 2.49 compare the performance of different vertex orders, in terms of the conflict number and the stitch number. Here *SEQUENCE-COLORING*, *DEGREE-COLORING*, and *3ROUND-COLORING* denote the coloring through three different orders, respectively. *Peer-Selection* is the method proposed in our linear color assignment. From Fig. 2.48 we can clearly see that peer selection can achieve less conflict number than any single vertex order. The reason is that, for each test case, the whole decomposition graph would be divided into several components. For each component one specific order may be dominant, then this order would be adopted by peer selection scheme. Therefore, for the whole layout, peer selection would be better than any single vertex ordering rule.

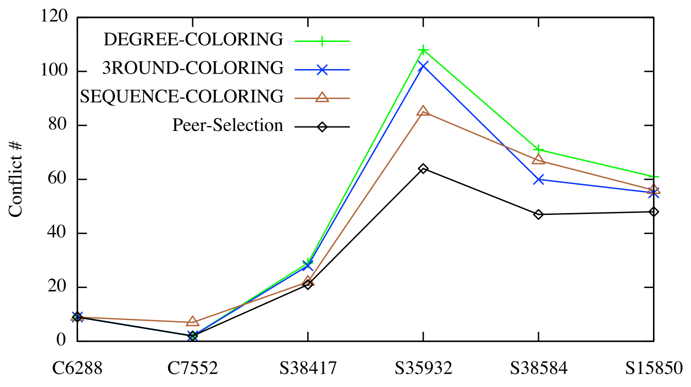


Figure 2.48: Comparison on conflict number.

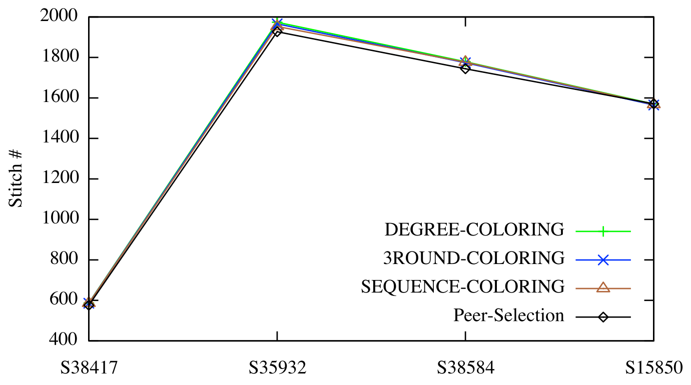


Figure 2.49: Comparison on stitch number.

2.5.4 Summary

In this section we have proposed the layout decomposition framework for quadruple patterning and beyond. Experimental evaluations have demonstrated that our algorithm is effective and efficient to obtain high quality solution. As continuing scaling of technology node to sub-10nm, MPL may be a promising manufacturing solution. We believe this work will stimulate more future research into this field, thereby facilitating the advancement of MPL technology.

Chapter 3

Standard Cell Compliance and Placement Co-Optimization with Triple Patterning

3.1 Introduction

The TPL layout decomposition problem with conflict and stitch minimization has been well studied in the past few years [23, 32, 38, 56, 92, 104, 105, 109, 115], including the early work presented in Chapter 2. However, most existing work suffers from one or more of the following drawbacks. (1) Because TPL layout decomposition problem is NP-hard [109], most of the decomposers are based on approximation or heuristic methods. Thus some extra conflicts may be reported. (2) For each design, since the library only contains fixed number of standard cells, layout decomposition would contain lots of redundant works. For example, if one cell is applied hundreds of times in a single design, it would be decomposed hundreds of times during layout decomposition. (3) Successfully carrying out these decomposition techniques requires the input layouts to be TPL friendly. However, since all these decomposition techniques are applied at post-place/route stage, where all the design patterns are already fixed, they lack the ability to resolve some native TPL conflict patterns, e.g., four-clique conflicts.

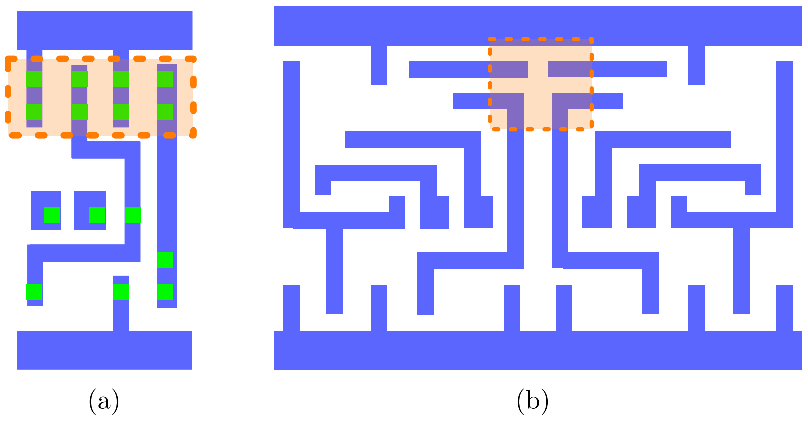


Figure 3.1: Two native conflicts (in read boxes) from (a) contact layer within a standard cell; (b) M1 layer between adjacent standard cells.

It is observed that the most hard-to-decompose patterns originate from contact and M1 layers. Fig. 3.1 shows two common native TPL conflicts in contact layer and M1 layer, respectively. As shown in Fig. 3.1(a), contact layout within the standard cell may generate some four-clique patterns, which is indecomposable. Meanwhile, if placement techniques are not TPL friendly, some boundary metals may introduce native conflicts (see Fig. 3.1(b)). Since redesigning indecomposable patterns in the final layout requires high ECO efforts, generating TPL-friendly layouts, especially in the early design stage, becomes urgent and pivotal. Through these two examples, we can see that TPL constraints should be considered in both standard cell design and placement stages, so that we can avoid indecomposable patterns in final layout.

There exist several placement studies toward different manufacturing process targets [20, 36, 42, 45, 59, 87]. Liebmann et al. [59] proposed some guidelines to enable DPL friendly standard cell design and placement, Taghavi et al. [87] presented a set of algorithms to disperse local congestion. Recently, [62, 65] proposed TPL aware detailed routing schemes. However, to our best knowledge, no previous work has addressed TPL compliance at standard cell or placement level.

In this section, we present a systematic framework to seamlessly integrate TPL constraints in early design stages, comprehending standard cell conflict removal, standard cell pre-coloring, and detailed placement together. Note that our framework is layout decomposition free, i.e., the TPL aware detailed placement can generate optimized positions and color assignment solutions for all cells simultaneously. Therefore, our framework does not require conventional and time consuming chip level layout decomposition. Our main contributions are summarized as follows:

- We propose systematic standard cell compliance techniques for TPL and coloring solution generation.
- We study the standard cell pre-coloring problem, and propose effective methods.
- We present the first systematic study for the TPL aware ordered single row placement, where cell placement and color assignment can be solved simultaneously.
- We propose linear dynamic programming algorithm to solve TPL aware single row placement with maximum displacement, and achieve a good trade-off in terms of runtime and solution quality.

- Our framework seamlessly integrates decomposition in each key step, therefore no additional layout decomposition is required.

- Experimental results show that our framework can achieve zero conflict, meanwhile can effectively reduce the stitch number.

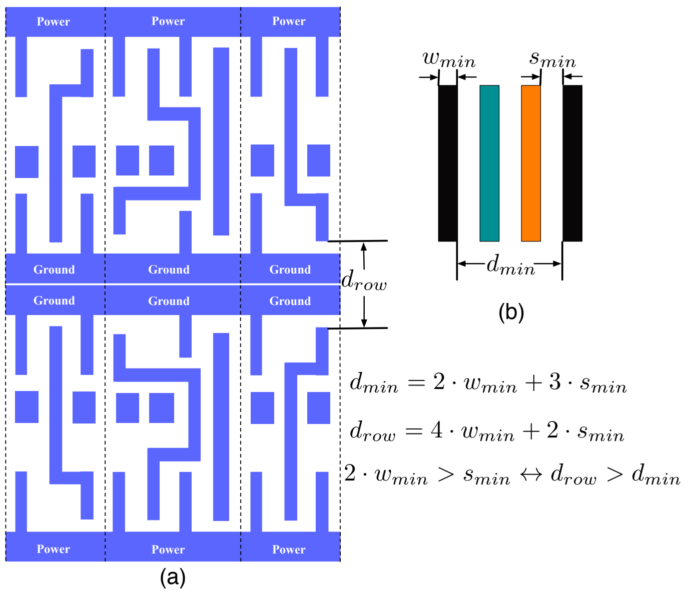
3.2 Preliminaries

3.2.1 Row Structure Layout

Our framework assumes a row-structure layout where cells in each row are with the same height, and power/ground rails are going from the very left to the very right (see Fig. 3.2(a)). A similar assumption was applied in row based TPL layout decomposition [92] as well. The minimum width of metal feature and the minimum spacing between neighboring metal features are denoted as w_{min} and s_{min}, respectively. We define the minimum spacing between metal features among different rows to be d_{row}. If we further analyze layout patterns in the library, it can be observed the width of a power/ground rail is twice the width of a metal wire within standard cells [3]. Under the row structure layout, we have the following lemma.

Lemma 6. *There is no coloring conflict between two M1 wires or contacts that are from different rows.*

Proof. For TPL, the layout will be decomposed into three masks, which means layout features within minimum coloring distance will be assigned three colors to increase the pitch between neighboring features. Then, we can see from Fig.



Power
Ground
w_{min}
s_{min}
d_{row}
d_{min}
$d_{min} = 2 \cdot w_{min} + 3 \cdot s_{min}$
$d_{row} = 4 \cdot w_{min} + 2 \cdot s_{min}$
$2 \cdot w_{min} > s_{min} \leftrightarrow d_{row} > d_{min}$
(a)
(b)

Figure 3.2: (a) Minimum spacing between M1 wires among different rows. (b) Minimum spacing between M1 wires with the same color.

3.2, the minimum spacing between M1 features with the same color in TPL is $d_{min} = 2 \cdot w_{min} + 3 \cdot s_{min}$. We assume the worst case for d_{row}, which means the standard cell rows are placed as mirrored cells and allow for no routing channel. Thus, $d_{row} = 4 \cdot w_{min} + 2 \cdot s_{min}$. We should have $d_{row} > d_{min}$, which equals $2 \cdot w_{min} > s_{min}$. This condition can easily be satisfied for M1 layer. For the same reason, we can achieve a similar conclusion for the contact layer. $\square$

Based on the row-structure assumption, the whole layout can be divided into rows, and layout decomposition or coloring assignment can be carried out for each row separately. Without loss of generality, for each row the power/ground rails are assigned the color 1 (*default color*). Then the decomposed results for each row will not induce coloring conflicts among different rows. In other words, the coloring assignment results in each row being able to be merged together, without losing optimality.

3.2.2 Overall Design Flow

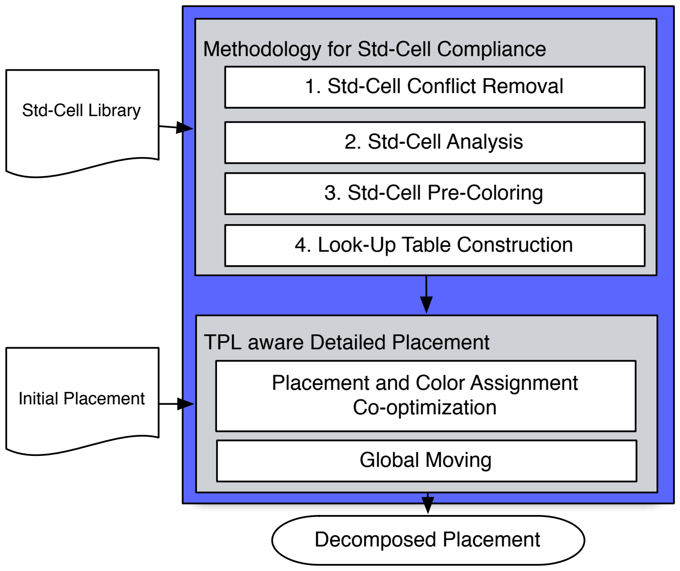

Std-Cell Library
Methodology for Std-Cell Compliance
1. Std-Cell Conflict Removal
2. Std-Cell Analysis
3. Std-Cell Pre-Coloring
4. Look-Up Table Construction
Initial Placement
TPL aware Detailed Placement
Placement and Color Assignment Co-optimization
Global Moving
Decomposed Placement

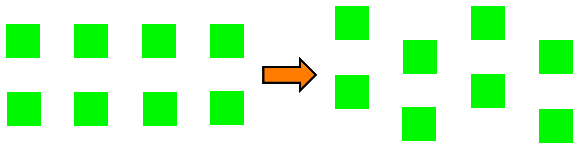
Figure 3.3: Overall flow of the methodologies for standard cell compliance and detailed placement.

The overall flow of our proposed framework is illustrated in Fig. 3.3, which consists of two stages: (1) Methodologies for standard cell compliance, and (2) TPL aware detailed placement. In the first stage, standard cell compliance, we carry out standard cell conflict removal, timing analysis, standard cell pre-coloring, and lookup table generation. After the first stage we can ensure that, for each cell, TPL friendly cell layout and a set of pre-coloring solutions will be provided. In the second stage, TPL aware detailed placement, we will discuss how to consider TPL constraints in the single row placement problem (see Section 3.4.1 and Section 3.4.2) and global moving (see Section 3.4.3).

Note that since triple patterning lithography constraints are seamlessly integrated into our coherent design flow, we do not need a separate additional step of layout decomposition. In other words, the output of our framework is decomposed layouts that have resolved cell placement and color assignment simultaneously.

3.3 Standard Cell Compliance

It is observed that without considering TPL in standard cell design, the cell library may involve several cells with native TPL conflict (see Fig. 3.1 (a) for one example). The inner native TPL conflict cannot be resolved through either cell shift or layout decomposition. In addition, one cell may be applied many times in one single design, thus each inner native conflict may cause hundreds of coloring conflicts in the final layout. To achieve TPL friendly layout after the physical design flow, we should first ensure the standard cell layout compliance for TPL. Specifically, we will manually remove all four-clique conflicts through standard cell modification. Then, parasitic extraction and SPICE simulation are applied to analyze the timing impact for the cell modification.


(a)

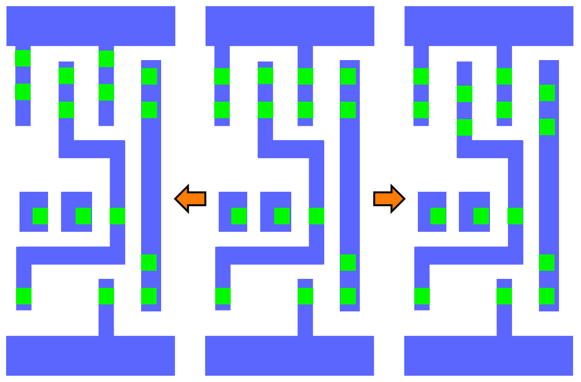

(b)

Figure 3.4: Contact layout modification to hexagonal packing. (a) The principle for contact shifting; (b) Demonstration of two options for contact shifting, with original layout in the middle, case 1 on the left and case 2 on the right.

3.3.1 Native TPL Conflict Removal

An example of native TPL conflict is illustrated in Fig. 3.4, where four contacts introduce an indecomposable four-clique conflict structure. For such cases we modify the contact layout into hexagonal close packing, which allows for the most aggressive cell area shrinkage for TPL friendly layout [64]. Note that after modification, the layout still needs to satisfy the design rules. From the layout analysis of different cells, we have various ways to remove such four-clique conflict. As shown in Fig. 3.4, with slight modification to the original layout, we can either choose to move contacts connected with power or ground rails or shift contacts on the signal paths of the cell. We call these

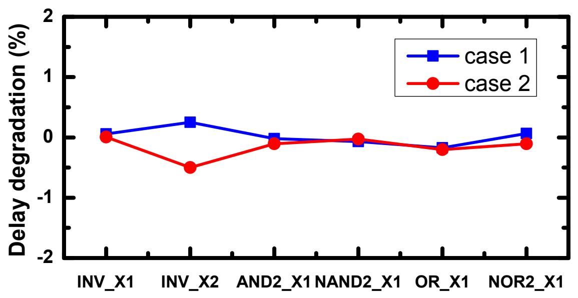


Figure 3.5: The timing impact from layout modification for different types of gates, including case 1 and case 2

two options case 1 and case 2, respectively, both of which will lead to TPL friendly standard cell layout. It shall be noted that although conventional cell migration techniques [30, 39, 110] might be able to automatically shift layout patterns to avoid four-clique patterns, it is hard to guarantee that the modified layout can maintain good timing performance. Therefore, in this work we manually modify the standard cell layout and verify timing after each shift operation.

3.3.2 Timing Characterization

Generally, the cell layout design flexibility is beneficial for resolving conflicts between cells when they are placed next to each other. However, from a circuit designer's perspective, we want to achieve little timing variation among various layout styles of a single cell. Therefore, we need simulation results to demonstrate negligible timing impact from layout modification.

A Nangate 45nm Open Cell Library [3] has been scaled to 16nm technology node. After native TPL conflict detection and layout modification, we carry out the standard cell level timing analysis. Calibre xRC [2] is used to extract parasitic information of the cell layout. For each cell, we have original and modified layouts with case 1 and case 2 options. From the extraction results, we can see that the source/drain parasitic resistance of transistors varies with the position of contacts, which is the direct impact from layout modification. We use SPICE simulation to characterize different types of gates, which is based on the 16nm PTM model [5]. Then, we can get the propagation delay of each gate, which is the average of rising and falling delay. We pick up six most commonly used cells to measure the relative changes of propagation delay due to layout modification (see Fig. 3.5). It is clearly observed that, for both case 1 and case 2, the timing impact will be within 0.5% of the original propagation delay of gates, which is assumed to be insignificant timing variation. Based on case 1 or case 2 options, we will remove all conflicts among cells of the library with negligible timing impact. Then, we can ensure the standard cell compliance for triple patterning lithography.

3.3.3 Standard Cell Pre-Coloring

For each type of standard cell, after removing the native TPL conflicts, we seek a set of pre-coloring solutions. These cell solutions are prepared as a supplement to the library. In this section, we first describe the cell pre-coloring problem formulation; then, we introduce our algorithms to solve this problem.

3.3.3.1 Problem Formulation

Given the input standard cell layout, all the stitch candidates are captured through wire projection [56]. One feature in the layout is divided into two touching parts, if one stitch candidate is introduced. Then a *constraint graph* (**CG**) is constructed to represent all input features and all the stitch candidates. A CG is an undirected graph, where each vertex is associated with one input layout feature. In a CG, there is a conflict edge iff the two corresponding touching vertices are connected through one stitch candidate, while there is a stitch edge iff two untouched vertices are within minimum coloring distance d_{min}. For example, given an input layout shown in Fig. 3.6 (a), five stitch candidates are generated through wire projection. The constraint graph is illustrated in Fig. 3.6 (b), where the conflict edges and the stitch edges are shown as solid edges and dash edges, respectively. Note that we forbid stitch on small features, e.g., contact, due to printability issue. Different from previous stitch candidate generation, we forbid the stitch on boundary metal wires due to the observation that boundary stitches tend to cause indecomposable patterns between two cells.

Based on the constraint graph, the standard cell pre-coloring problem is to search all possible coloring solutions. At first glance, this problem is similar to cell level layout decomposition. However, different from the conventional layout decomposition, for each cell pre-coloring could have more than one solution. It is observed that for some complex cell structures, if we exhaustively enumerate all possible colorings, it would have thousands of solutions. Large

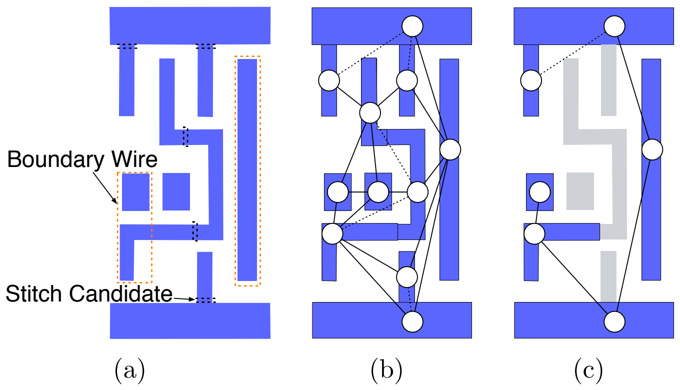


Figure 3.6: Constraint graph construction and simplification. (a) Input layout and all stitch candidates. (b) Constraint graph (CG) where solid edges are conflict edges and dash edges are stitch edges. (c) The simplified constraint graph (SCG) after removing immune features.

solution size would impact the performance of our whole flow. Therefore, to provide high quality pre-coloring solutions, meanwhile keeping the solution size as small as possible, we define *immune feature* and *redundant coloring solutions* as follows.

Definition 6 (Immune feature)**.** *In one standard cell, an inside feature that does not conflict with any outside feature is defined as an immune feature.*

It is easy to see that for one feature, if its distances to both vertical boundaries are larger than d_{min}, its color would not conflict with any other cells. Then, this feature is an immune feature.

Definition 7 (Redundant coloring solutions)**.** *If two coloring solutions are only different at the immune features, these two solutions are redundant to each other.*

Problem 6 (**Standard Cell Pre-Coloring**). *Given the input standard cell layout and the maximum allowed stitch number, $maxS$, the CG is constructed. Standard cell pre-coloring problem searches all coloring solutions on CG such that the stitch number is no more than $maxS$. Meanwhile, no two solutions are redundant with each other.*

Since in CG some vertices represent the immune features, to avoid redundant coloring solutions, these features are temporarily removed. We denote the remaining graph as a *simplified constraint graph* (**SCG**). For example, for the constraint graph in Fig. 3.6 (b), the corresponding simplified constraint graph is shown in Fig. 3.6 (c). Our standard cell pre-coloring algorithm consist of two stages: coloring solution enumeration on SCG, and solution verification on CG.

3.3.3.2 SCG Solution Enumeration

In the first step, given a SCG, we enumerate all possible coloring solutions. Our enumeration is based on backtracking algorithm [75], which usually explores implicit directed graphs to carry out a systematic search of all solutions.

The details of SCG solution enumeration are shown in Algorithm 11. Given a SCG, G, a backtracking function, BACKTRACK(0, G) is called to search the whole graph (line 1). The backtracking is a modified depth-first search of the solution space (lines 3-13). In line 7, a color c is denoted as legal, if when vertex $G[t]$ is assigned color c, no conflict is introduced, and the

Algorithm 11 SCG Solution Enumeration

```
Require: SCG $G = \{V, CE, SE\}$;
 1: BACKTRACK($0, G$);
 2: return All color solutions in $G$;

 3: function BACKTRACK($t, G$)
 4:     if t $\geq$ size[$G$] then
 5:         Store current color solution;
 6:     else
 7:         for all legal color $c$ do;
 8:             $G[t] \leftarrow c$;
 9:             BACKTRACK($t + 1, G$);
10:             $G[t] \leftarrow -1$;
11:         end for
12:     end if
13: end function
```

total stitch number does not exceed $maxS$. It should be mentioned that since all power/ground rails are assigned default color, the colors of corresponding vertices are assigned before the backtracking process. For example, given the SCG shown in Fig. 3.6(c), if no stitch is allowed, there are 8 solutions (see Fig. 3.7).

3.3.3.3 CG Solution Verification

Until now we have enumerated all coloring solutions for simplified constraint graph (SCG). However, not all the SCG solutions can achieve legal layout decomposition in the initial constraint graph (CG). Therefore, in the second step, CG solution verification is proposed to each generated solution. Since SCG is a sub-set of CG, the verification can be viewed as layout de-

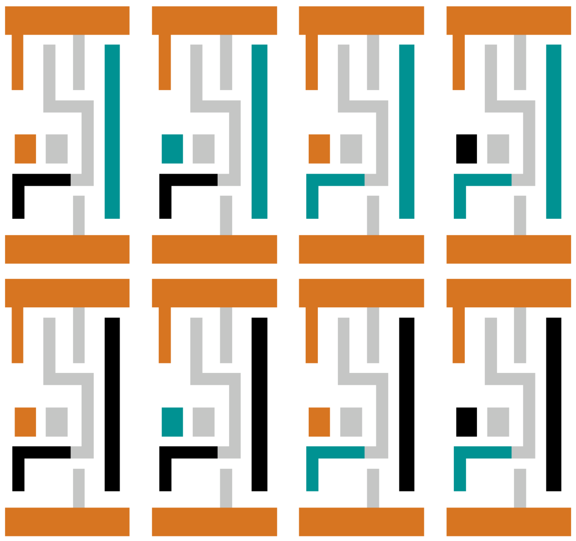


Figure 3.7: AND2X1 cell example: 8 enumerated solutions for SCG.

composition with pre-colored features on SCG. If a coloring solution for whole CG can be found with stitch number less than $maxS$, it would be stored as one pre-coloring solution. The CG solution verification is based on the **branch-and-bound** algorithm [75], which is very similar to backtracking in that a state space tree is used to solve a problem. However, the differences are twofold. (1) The branch-and-bound method is used only for optimization problem, i.e., only one solution is generated. (2) The branch-and-bound algorithm introduces bounding function to prune sub-optimal nodes in search space. That is, at each node of search space, we calculate a bound on the possible solution. If the bound is worse than the best solution we have found so far, then we do not need to go to the sub-space.

The details of the CG solution verification are shown in Algorithm

Algorithm 12 CG Solution Verification

```
Require: Set of initial coloring solutions $S'$ for SCG;
1: Generate corresponding coloring solutions $S$ for CG;
2: for each coloring solution $s_i \in S$ do
3:     $minCost \leftarrow \infty$;
4:     BRANCH-AND-BOUND$(0, s_i)$;
5:     if $minCost < maxS$ then
6:         Output $s_i$ as legal pre-coloring solution;
7:     end if
8: end for

9: function BRANCH-AND-BOUND$(t, s_i)$
10:     if t $\geq$ size$[s_i]$ then
11:         if GET-COST( ) $< minCost$ then
12:             $minCost \leftarrow$ GET-COST();
13:         end if
14:     else if LOWER-BOUND( ) $> minCost$ then
15:         Return;
16:     else if $s_i[t] \neq -1$ then
17:         BRANCH-AND-BOUND$(t + 1, s_i)$;
18:     else                                        $\triangleright$ $s_i[t] = -1$
19:         for each available color $c$ do;
20:             $s_i[t] \leftarrow c$;
21:             BRANCH-AND-BOUND$(t + 1, s_i)$;
22:             $s_i[t] \leftarrow -1$;
23:         end for
24:     end if
25: end function
```

12. Given a SCG coloring solutions $S' = \{s'_1, s'_2 \dots s'_n\}$, at the beginning the corresponding CG coloring solutions $S = \{s_1, s_2, \dots, s_n\}$ are generated (line 1). Then we iteratively check each coloring solution s_i (lines $2 - 6$). For one coloring solution s_i, if vertex t belongs to SCG, $s_i[t]$ should be already assigned one legal color. If t does not belong to SCG, $s_i[t] \leftarrow -1$. The BRANCH-AND-

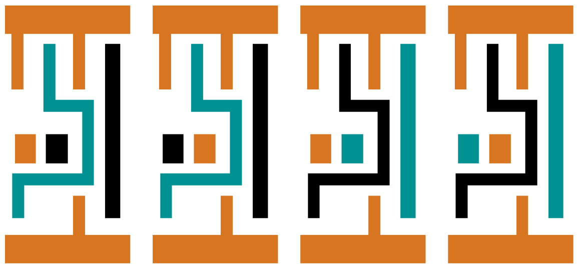


Figure 3.8: AND2X1 cell example: In CG 4 verified solutions are stored as final coloring solutions.

BOUND() algorithm traverses the decision tree with a depth first search (DFS) method (lines $7-19$). For each vertex t, if $s_i[t]$ has been assigned one legal color in SCG, we skip t and travel to the next vertex. Otherwise, every legal color would be assigned to t before traveling to the next vertex. Different from exhaustive search, search space can be effectively reduced through the pruning process (lines $11-12$). The function LOWER-BOUND() is to get the lower bound by calculating the current stitch number. Note that if one conflict is found, then the function returns a large value. Before checking any legal color of vertex t, we calculate its lower bound first. If LOWER-BOUND() is larger than $minCost$, we shall not branch from t, since all the children solutions will be of higher cost than $minCost$. Through the travel, all vertices have been assigned legal colors, stored in s_i. After the travel, if $minCost \leq maxS$, then s_i is one of the pre-coloring solutions (lines $5-6$).

It shall be noted that although other layout decomposition techniques, like integer linear programming (ILP), may be modified as the verification engine, our branch-and-bound based method is easy to implement and effective for standard cell level problem size. Even for the most complex cell, SCG solution enumeration and CG solution verification can be finished in 5 seconds. For the SCG solutions in Fig. 3.7, four solutions are verified and assigned final colors (see Fig. 3.8). These four solutions would be the final coloring solutions for this standard cell, and are provided as supplement to the library.

3.3.4 Look-Up Table Construction

For each cell c_i in the library, we have generated a set of pre-coloring solutions $S_i = \{s_{i1}, s_{i2}, \ldots, s_{iv}\}$. We further pre-compute the decomposability of each cell pair and store them in a lookup table. For example, if two cells, c_i and c_j, are assigned with the $p-$th and $q-$th coloring solutions, respectively, then in lookup table a value $\text{LUT}(i, p, j, q)$ would be stored, which is the minimum distance required when c_i is to the left of c_j. If two colored cells can be legally abutted to each other, the corresponding value would be 0. Otherwise, the value would be the site number required to keep two cells decomposable. Meanwhile, for each cell, the stitch numbers in different coloring solutions are also stored. It shall be noted that during the lookup table construction, the cell flipping is considered and related values are stored as well.

3.4 TPL aware Detailed Placement

3.4.1 TPL aware Ordered Single Row Placement

We first solve a single row placement, where the orders of all cells on the row are determined. When the TPL process is not considered this row

Table 3.1: Notations used in TPL-OSR problem

m	site number
n	cell number
C	a set of cells $\{c_1, c_2, \ldots, c_n\}$
v_i	pre-coloring solution number for cell c_i
K	$\max\{v_1, v_2, \ldots, v_n\}$
(i, p)	cell c_i is assigned to p^{th} color solution
LUT(i, p, j, q)	min distance required between (i, p) & (j, q)
$s(i, p)$	stitch number for (i, p)
$x(i)$	horizontal position of c_i
$w(i)$	width of c_i
$a(i)$	assigned color for c_i

based design problem is called *Ordered Single Row* (OSR) problem, which has been well studied [17, 50, 51, 94]. Here we revisit the OSR problem with the TPL process consideration. For convenience, Table 3.1 lists the notations used in this section.

3.4.1.1 Problem Formulation

We consider an input single row as m ordered sites $R = \{r_1, r_2, \ldots, r_m\}$, and an input n movable cells $C = \{c_1, c_2, \ldots, c_n\}$ whose order is determined. That is, c_i is to the left of c_j, if $i < j$. Each cell c_i has v_i different coloring solutions. A *cell-color pair* (i, p) denotes that cell c_i is assigned to the p^{th} color solution, where $p \in [1, v_i]$. Meanwhile, $s(i, p)$ gives the corresponding stitch number for (i, p). The horizontal position of cell c_i is given by $x(i)$, and the cell width is given by $w(i)$. All the cells in other rows are with fixed positions. A single row placement is *legal* if and only if any two cells, c_i and c_j, meet the following non-overlap constraint:

$$x(i) + w(i) + \text{LUT}(i, p, j, q) \leq x(c_j), \quad \text{if } (i, p)\&(j, q)$$

where $\text{LUT}(i, p, j, q)$ is the minimum distance required between (i, p) & (j, q). Based on all these notations, we define the TPL aware Ordered Single Row (*TPL-OSR*) problem as follows.

Problem 7 (TPL aware Ordered Single Row Problem). *Given a single row placement, we seek a legal placement and cell color assignment, so that the half-perimeter wire-length (HPWL) of all nets and the total stitch number are minimized.*

Compared with the traditional OSR problem, the TPL-OSR problem faces two special challenges: (1) TPL-OSR not only needs to solve cell placement, but also needs to assign appropriate coloring solutions for cells to minimize the stitch number. In other words, cell placement and color assignment should be solved simultaneously. (2) In conventional OSR problems, if the sum of all cell widths is less than row capacity, it is guaranteed that there would be one legal placement solution. However, for TPL-OSR problems, since some extra sites may be spared to resolve coloring conflicts, before coloring assignment we cannot calculate the required site number.

In addition, it shall be noted that compared with the conventional color assignment problem, in TPL-OSR the solution space is much larger. That is, to resolve the coloring conflict between two abutted cells, c_i and c_j, apart from

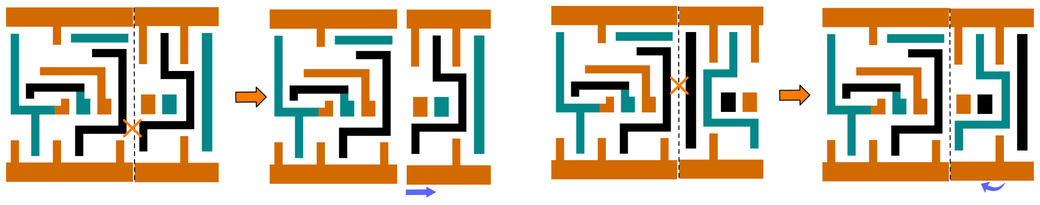

(a) (b)

Figure 3.9: Two techniques for removing conflicts during placement. (a) Flip the cell; (b) Shift the cell.

picking up compatible coloring solutions, TPL-OSR can seek to flip cells (see Fig. 3.9 (a)) or shift cells (see Fig. 3.9 (b)).

3.4.1.2 Unified Graph Model

We propose a graph model that correctly captures the cost of HPWL and the stitch number. Furthermore, we will show that performing a shortest path algorithm on the graph model can optimally solve the TPL-OSR problem.

To consider cell placement and cell color assignment simultaneously, a directed acyclic graph $G = (V, E)$ is constructed. The graph G is with vertex set V and edge set E. $V = \{\{0, \dots, m\} \times \{0, \dots, N\}, t\}$, where $N = \sum_{i=1}^{n} v_i$. The vertex in the first row and the first column is defined as vertex s. We can see that each column corresponds to one site's start point, and each row is related to one specified color assignment of one cell. Without loss of generality, we label each row as $r(i, p)$, if it is related to cell c_i with p^{th} coloring solution. The edge set E is composed of three sets of edges: horizontal edges E_h, ending

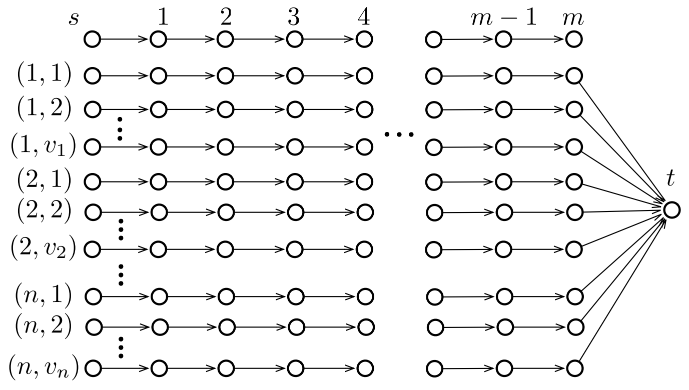


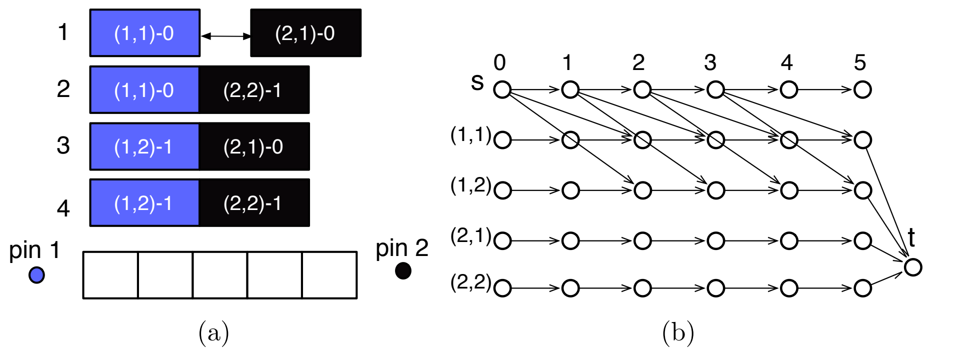
Figure 3.10: Graph model for the TPL-OSR problem (only the horizontal edges and ending edges are showed).

edges E_e, and diagonal edges E_d.

$$\begin{aligned} E_h =& \{(i, j-1) \rightarrow (i, j) | 0 \leq i \leq N, 1 \leq j \leq m\} \\ E_e =& \{(i, m) \rightarrow t | i \in [1, N]\} \\ E_d =& \{(r(i-1, p), k) \rightarrow (r(i, q), k + w(i) + \\ & LUT(i-1, p, i, q)) | i \in [2, n], p \in [1, v_{i-1}], q \in [1, v_i]\} \end{aligned}$$

We denote each edge by its start and end point. A legal TPL-OSR solution corresponds to finding a directed path from the vertex s to vertex t. Sometimes one row cannot insert all the cells, therefore ending edges are introduced. With these ending edges, the graph model can guarantee to find out one path from s to t.

To simultaneously minimize the HPWL and stitch number, we define the cost on edges as follows. (1) All horizontal edges are with zero cost. (2) For ending edge $\{(r(i, p), m) \rightarrow t\}$, it is labeled by the cost $(n - i) \cdot M$, where



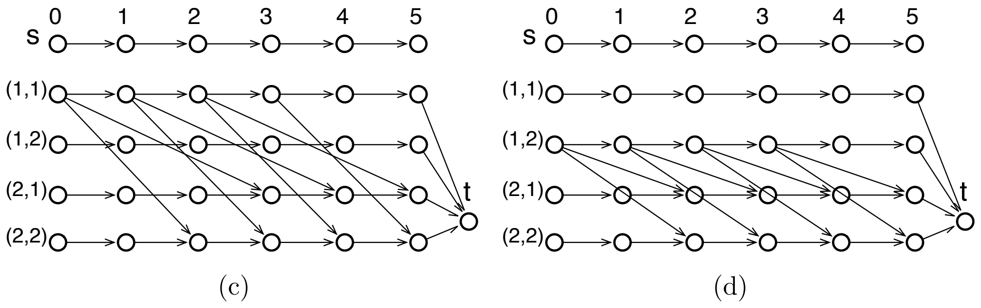


Figure 3.11: Example for the TPL-OSR problem. (a) two cells with different coloring solutions to be placed into a 5 sites row; Graph models with diagonal edges (b) from s vertex to first cell; (c) from c1_1 to second cell; (d) from c1_2 to second cell.

M is a large number. (3) For diagonal edge $\{(r(i,p),k) \rightarrow (r(j,q), k + w(c_j) + LUT(i,p,j,q))\}$, it is labelled by the cost as follows:

$$\Delta WL + \alpha \cdot s(i,p) + \alpha \cdot s(j,q)$$

where ΔWL is the HPWL increment of placing c_j in position $q-\text{LUT}(i,p,j,q)$. Here α is a user-defined parameter for assigning relative importance between the HPWL and the stitch number. In our framework, α is set to 10. The general structure of G is shown in Fig. 3.10. Note that for clarity, here we do not show the diagonal edges.

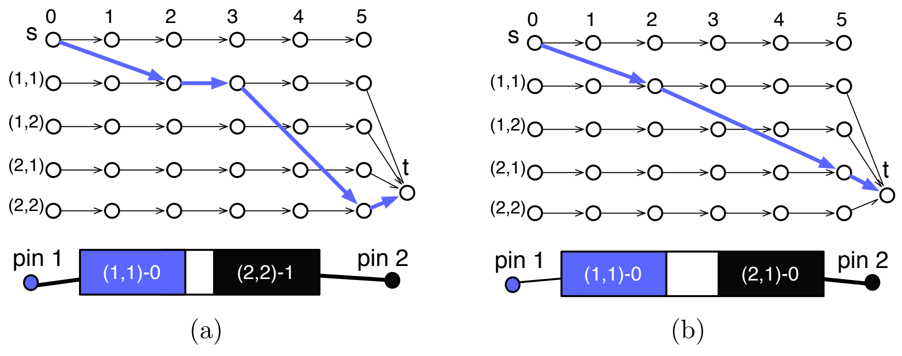


Figure 3.12: Shortest path solutions on the graph model with (a) 1 stitch; (b) 0 stitch.

One example of the graph model is illustrated in Fig. 3.11, where two cells, c_1 and c_2, are to be placed in a row with 5 sites. Each cell has two different coloring solutions and corresponding required stitch number. For example, the label (2,1)-0 means c_2 is assigned to the first coloring solution, with no stitch. The graph model is shown in Fig. 3.11(b)(c)(d), where each figure shows different part of diagonal edges. Cells c_1 and c_2 are connected with pin 1 and pin 2, respectively. Therefore, c_1 tends to be on the left side of the row, while c_2 tends to be on the right side. Fig. 3.12 gives two shortest path solutions with the same HPWL. Because the second has a lower stitch number, it would be selected as the solution for the TPL-OSR problem.

Since G is a directed acyclic graph, the shortest path can be calculated using topological traversal of G in $O(mnK)$ steps, where K is the maximal pre-coloring solution number for each cell. To apply topological traversal, a dynamic programming algorithm is proposed to find the shortest path from the s vertex to the t vertex.

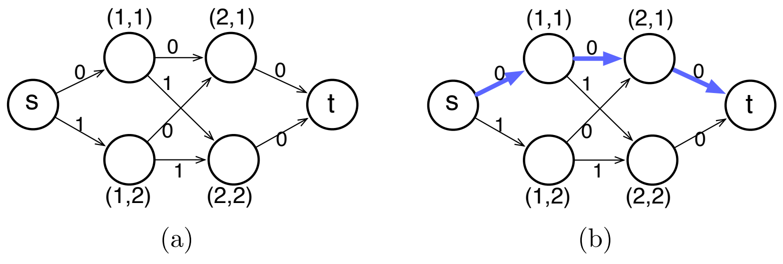


Figure 3.13: (a) The first stage to solve color assignment. In this example edge cost only considers the stitch number minimization. (b) One shortest path solution corresponds to a color assignment solution.

3.4.1.3 Two Stage Graph Model

Although the unified graph model can be optimally solved through a shortest path method in $O(mnK)$, for practical design when each cell could allow many pre-coloring solutions, the proposed graph model may still suffer from long runtime penalty. Here we present a new two-stage graph model for the TPL-OSR problem. The main idea is that the previous unified graph model is decomposed into two smaller graphs, one for color assignment and another for cell placement. Therefore, solving the new model can provide a fast solution to the TPL-OSR problem.

To solve the example in Fig. 3.11, the first stage graph model is illustrated in Fig. 3.13 (a), where the cost of each edge corresponds to the stitch number required for each cell-color pair (i, p). Note that in our framework, relative positions among cells are also considered in the edge cost. A shortest path on the graph corresponds to a color assignment with minimum stitch number.

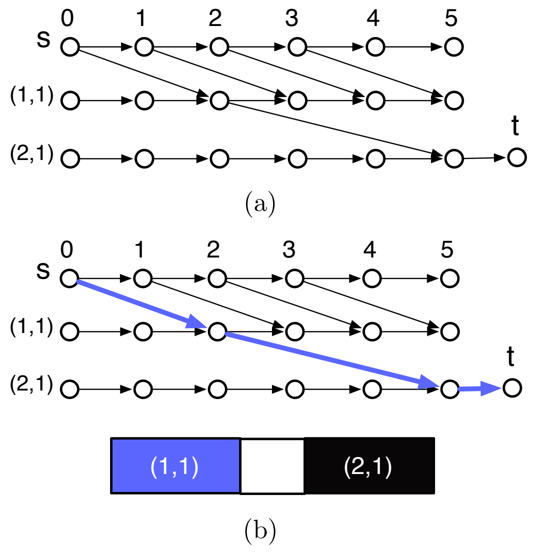


Figure 3.14: (a) The second stage to solve detailed placement. (b) One shortest path solution corresponds to a cell placement.

Our second stage is for cell placement and the previous color assignment solutions are considered here. That is, if in previous color assignment cells, c_{i-1} and c_i, are assigned its $p-$th and $q-$th coloring solutions, then the width of cell c_i is changed from $w(i)$ to $w(i) + \text{LUT}(i-1, p, i, q)$. This way, the extra sites to resolve coloring conflicts are prepared for cell placement. Based on the updated cell widths, the graph model in [50] can be directly applied here. For instance, the second stage graph model for the example in Fig. 3.11 is illustrated in Fig. 3.14. It shall be noted that all cells have been assigned a coloring solution, thus the graph size is much smaller than that in Fig. 3.11. As shown in Fig. 3.14 (b), the shortest path on the graph corresponds to a cell placement.

The first graph model can be solved in $O(nK)$, while the second graph model can be resolved in $O(mn)$. Therefore, although the speed-up technique can not achieve an optimal solution of the TPL-OSR problem, applying the two-stage graph model can reduce the complexity from $O(mnK)$ to $O(nK + mn)$.

3.4.2 TPL-OSR with Maximum Displacement

Here we consider another single row placement problem, which is similar to the TPL-OSR, where the initial cell orders are determined. The slight difference here is that each cell is forbidden from moving more than distance M from its original location. The new problem is called **TPL-OSR with Maximum Displacement**. The motivation to study this new problem is twofold. **First**, in the previous TPL-OSR problem, although the two stage graph model can provide fast solutions due to the nature that the color assignment and cell placement are solved separately, its solution qualities may not be good enough. For the new problem, we are able to propose a fast but high performance optimization algorithm. **Second**, from the design perspective, a detailed placement technique with maximum displacement constraints is robust and important in practical situations. For example, if the initial placement is optimized toward other design metrics, e.g., pin density or routability, limiting cell displacements can help to maintain these metrics.

3.4.2.1 Problem Formulation

The problem we solve is finding new locations for all cells that preserve their relative order. Meanwhile, each cell has maximum moving distance, M, from its original location. In other words, each cell c_i has $2M + 1$ possible locations between $[x(i) - M, x(i) + M]$. Here $x(i)$ is the original position of cell c_i, while M is a user-defined parameter. Based on these notations, the TPL-OSR with maximum displacement problem is defined as follows:

Problem 8 (TPL-OSR with Maximum Displacement). *Given a single row placement, we seek cell displacement values $d(i)$, with $|d(i)| < M$, and color assignments so that the half-perimeter wire-length (HPWL) of all nets and the total stitch number are minimized.*

3.4.2.2 Linear Dynamic Programming Algorithm

Inspired by [87], our algorithm is based on linear dynamic programming, which means the optimal solution can be searched in linear time. The main idea is that we process cells starting from c_1 and explore cell pair locations for (c_1, c_2), followed by (c_2, c_3), etc. Once the optimal placements and color assignments for $c_1, \ldots, c_{i-1}$ are computed we search the optimal placement and color assignment simultaneously for c_i. For convenience, Table 3.2 lists some additional notations used in the linear dynamic programming.

The details of the linear dynamic programming are shown in Algorithm 13. Line 1 initializes the solution costs. The main algorithmic computation takes place in the loops (lines 2–17). We iteratively explore all cell pairs

Table 3.2: Notations used in linear dynamic programming

$a(i)$	Color assignment value for c_i.
$d(i)$	Displacement value for c_i.
$t[i][d][a]$	Best cost for $c_1, \ldots, c_i$ with $d(i) = d$ and $a(i) = a$.
$d[i][d][a]$	Displacement of c_{i-1} in an optimal sub-solution of $\{c_1, \ldots, c_i\}$ with $d(i) = d$ and $a(i) = a$.
$a[i][d][a]$	Color assignment of c_{i-1} in an optimal sub-solution of $\{c_1, \ldots, c_i\}$ with $d(i) = d$ and $a(i) = a$.
$r[i][d][a]$	whether $t[i][d][a]$ is inferior.

(c_{i-1}, c_i), with different displacement values and color assignment solutions (lines 2–4). For cell pair (c_{i-1}, c_i) and different combinations of $(d1, a1, d2, a2)$, the best cost is stored in $t[i][d2][a2]$, while $d1$ and $a1$ are stored in $d[i][d2][a2]$ and $a[i][d2][a2]$, respectively (lines 9–13). $F_{i-1}(d1, a1, d2, a2)$ is the cost considering wire-length impact and stitch number, defined as follows:

$$\Delta WL + \alpha \cdot s(i-1, a1) + \alpha \cdot s(i, a2)$$

where ΔWL is the HPWL improvement of placing c_{i-1} and c_i in $x(i-1) + d1$ and $x(i) + d2$, respectively. $s(i-1, a1)$ and $s(i, a2)$ are used to calculate the stitch numbers. Here α is a user-defined parameter for assigning relative importance between the HPWL and the stitch number.

Different from the method in [87], we propose pruning techniques to speed-up the dynamic programming process. For any two solutions $t[i][d1][a]$ and $t[i][d2][a]$, if $t[i][d1][a] >= t[i][d2][a]$ and $d1 >= d2$, we can say $t[i][d1][a]$ is **inferior** to $t[i][d2][a]$. Then $r[i][d1][a]$ is assigned 1 to label the inferiority (line 16). Therefore, one can exit early when checking the r value (lines 5-7).

Algorithm 13 Linear Dynamic Programming

```
Require: Cells $C$ in row sites $R$;
 1: Initialize matrices $r, t, d$, and $a$; $F \leftarrow \infty$;
 2: for all $i = 2$ to $n$ do
 3:     for all $a1 = 1$ to $v_{i-1}, a2 = 1$ to $v_i$ do
 4:         for all $d1 = -M$ to $M, d2 = -M$ to $M$ do
 5:             if $r[i-1][d1][a1] = 1$ then
 6:                 continue;
 7:             end if
 8:             $y = t[i-1][d1][a1] + F_{i-1}(d1, a1, d2, a2)$;
 9:             if $y < t[i][d2][a2]$ then
10:                 $t[i][d2][a2] \leftarrow y$;
11:                 $d[i][d2][a2] \leftarrow d1$;
12:                 $a[i][d2][a2] \leftarrow a1$;
13:             end if
14:         end for
15:     end for
16:     Mark inferior ones with $r[i][d][a] \leftarrow 1$;
17: end for
18: for $dn = -M$ to $M, an = 1$ to $v_n$ do
19:     if $t[n][dn][an] < F$ then
20:         $F \leftarrow t[n][dn][an]$;
21:         $d(n) \leftarrow dn$;
22:         $a(n) \leftarrow an$;
23:     end if
24: end for
25: for $i = n$ downto 2 do
26:     $d(i-1) \leftarrow d[i][d(i)][a(i)]$;
27:     $a(i-1) \leftarrow a[i][d(i)][a(i)]$;
28: end for
```

Lines 18–24 compute the end case of the last cell in the row, and the solution is recovered at last (lines 25–28).

Theorem 5. *The linear dynamic programming runs in* $O(nK^2M^2)$ *time to* *optimally solve the problem,*

The complexity analysis results from the **for** loops (lines 2–17). Since both K and M are constants, the runtime complexity of Algorithm 13 is linear. Optimality stems from the fact that $t[i][\,][\,]$ explores all possible displacement values and color assignment solutions. It shall be noted that the unified graph model in Section 3.4.1.2 can be also modified to solve the problem here. However, the runtime complexity using unified graph model is $O(nmK)$. Since usually m is larger than n, the complexity of a unified graph model is quadratic and may be slower than linear dynamic programming.

3.4.3 Overall Placement Scheme

In this section, we present our overall scheme for the whole design level TPL aware detailed placement. Algorithm 14 summarizes the overall flow. At the beginning, all rows are labeled as $FREE$, which means additional cells can be inserted (line 3). In each main loop, rows are sorted such that the row with more cells occupied would be solved earlier. For each row row_i, we carry out single row TPL aware detailed placement as introduced in Section 3.4.1 and Section 3.4.2, to solve color assignment and cell placement simultaneously. Note that sometimes in one row we cannot assign all cells legal positions, due to extra sites required to resolve coloring conflicts.

If single row problem ends with unsolved cells, *Global Moving* is applied to move some cells to other rows (line 7). The basic idea behind the Global Moving is to find the "optimal row and site" for a cell in the placement region

Algorithm 14 TPL aware Detailed Placement

```
Require: Cells to be placed;
 1: repeat
 2:     Sort all rows;
 3:     Label all rows as $FREE$;
 4:     for each row $row_i$ do
 5:         Solve single row problem for $row_i$;
 6:         if exist unsolved cells then
 7:             Global Moving;
 8:             Update cell widths considering assigned colors;
 9:             Solve OSR problem for $row_i$;
10:         end if
11:         Label $row_i$ as $BUSY$;
12:     end for
13: until no significant improvement;
```

and remove some local triple patterning conflicts. For each cell we define its "optimal region" as the site to place where the HPWL is optimal [41]. Note that one cell can be only moved to $FREE$ rows. Since some cells in the middle of a row may be moved, we need to solve OSR problem to rearrange the cell positions [50]. Note that since all cells on the row have been assigned colors, cell widths should be updated to preserve extra space for coloring conflict (line $8-9$) . After solving one row_i, it is labeled as $BUSY$ (line 10).

Since the rows are placed and colored one by one sequentially, the solution obtained within one single row may not be good enough. Therefore, our scheme is able to repeatedly call the main loop until no significant improvement is achieved (line 13).

3.5 Experimental Results

We implement our standard cell pre-coloring and TPL aware detailed placement in C++, and all the experiments are performed on a Linux machine with 3.0GHz CPU. Nangate 45nm library [3] is scaled down to 16nm technology node, as our initial standard cell library. We apply standard cell compliance and pre-coloring on the scaled the library. During standard cell pre-coloring, each cell's maximum allowed stitch number, $maxS$, is set to 2. We use Design Compiler [6] to synthesize OpenSPARC T1 designs based on the modified cell library. For each benchmark, we perform placement with Cadence SOC Encounter [1] to generate initial placement results. To better compare the performance of detailed placement under different placement densities, for each circuit, we choose three different core utilization rates 0.7, 0.8, and 0.85. Generally speaking, the higher utilization rate, the more difficult of the detailed placement.

The benchmark statistics are listed in Table 3.3. Column "K" is the maximum cell pre-coloring solution number among all standard cell types, which is related to the lookup table size. Columns "cell #" and "row #" are the total cell module number and the total row number for each placement test case, respectively. Both "cell #" and "row #" reflect the placement problem size. To demonstrate the problem size of each single row placement, columns "max cell # per row" and "max m per row" are used. These columns represent maximum cell module number in one row, and maximum site number in one row, respectively.

Table 3.3: Benchmark Statistics

bench	K	cell #	row#	max cell # per row	max m per row
alu-70	74	2110	43	67	330
alu-80	74	2110	41	68	302
alu-85	74	2110	39	67	299
byp-70	32	4416	81	75	597
byp-80	32	4416	75	84	564
byp-85	32	4416	73	89	545
div-70	74	3758	65	92	489
div-80	74	3758	61	94	456
div-85	74	3758	59	109	444
ecc-70	32	1322	42	42	322
ecc-80	32	1322	40	47	296
ecc-85	32	1322	38	45	293
efc-70	74	1183	40	45	300
efc-80	74	1183	37	43	283
efc-85	74	1183	36	44	274
ctl-70	74	1694	48	45	363
ctl-80	74	1694	45	54	339
ctl-85	74	1694	44	53	326
top-70	74	14793	123	146	919
top-80	74	14793	115	157	860
top-85	74	14793	112	158	832

In the first experiment we demonstrate the effectiveness of our overall TPL aware design compared to conventional TPL aware design. Conventional TPL aware design flow consists of standard cell synthesis, placement, and TPL layout decomposition at post-stage. Our TPL aware design flow integrates TPL constraints into standard cell synthesis and detailed placement, and no layout decomposition is required on the whole chip layout. Table 3.4 compares both flows for the M1 layer of all the benchmarks. Column "**Conventional**

flow” is the conventional TPL design flow. Encounter is chosen as the placer, and an academic decomposer [32] is used as our layout decomposer. Column “**Our flow**” is the proposed TPL aware detailed placement. Layout modification and pre-coloring are carried out for each standard cell, and the optimal graph model is utilized to solve cell placement and color assignment simultaneously. Note here for each flow, the standard cell inner native conflicts have been removed through our compliance techniques (see Section 3.3). In other words, theoretically the conflicts can only happen on the boundaries between standard cells.

On the one hand, we can see that in the conventional design flow, even each standard cell itself is TPL-friendly on average more than 1,000 conflicts are reported in final decomposed layout. Meanwhile, on average over 20,000 stitches are introduced for each case. Due to the large number of conflicts and stitches, a lot of efforts may be required to manually modify or migrate the layout to resolve the conflicts. On the other hand, through considering TPL constraints in early design stages, our proposed TPL aware design flow can guarantee **zero** conflict. Since stitch number optimization is considered in both cell pre-coloring and TPL aware detailed placement, the stitch number can be reduced by 92.7% comparing with traditional flow.

In Section 3.4.1 and Section 3.4.2 we have proposed several algorithms to solve TPL aware single row detailed placement. In the second experiment, we analyze the performances of the proposed algorithms and related speed-up techniques in Table 3.5. Column “**GREEDY**” is a greedy detailed placement

Table 3.4: Comparisons with Traditional Flow

bench	Conventional flow		Our flow		
	CN#	ST#	CN#	ST#	ΔST#
alu-70	461	8476	0	1013	-89.5%
alu-80	428	10862	0	1011	-91.2%
alu-85	493	5559	0	1006	-82.2%
byp-70	1168	43730	0	2743	-96.7%
byp-80	1251	35973	0	2889	-95.6%
byp-85	1342	38576	0	3136	-95.6%
div-70	821	19492	0	2119	-91.9%
div-80	860	18928	0	2090	-90.8%
div-85	982	22070	0	2080	-91.9%
ecc-70	368	7801	0	247	-96.4%
ecc-80	362	10083	0	274	-97.1%
ecc-85	372	9990	0	369	-96.8%
efc-70	222	7589	0	1005	-90.0%
efc-80	225	13006	0	1008	-93.5%
efc-85	258	5260	0	1005	-83.7%
ctl-70	354	15356	0	573	-97.8%
ctl-80	380	15158	0	561	-97.4%
ctl-85	414	14873	0	556	-97.1%
top-70	3708	58953	0	8069	-90.5%
top-80	3976	60736	0	8120	-89.8%
top-85	4265	70316	0	8710	-90.9%
Average	1081	23466	**0**	1672.5	**-92.7%**

algorithm [36], which is implemented as our baseline. Although the work in [36] is targeting the self-aligned double patterning (SADP), the proposed detailed placement algorithm can be modified to be integrated into our framework. Columns "**TPLPlacer**" and "**TPLPlacer-2Stage**" are detailed placement algorithms with different TPL-OSR engines. TPLPlacer utilizes the optimal unified graph model, while TPLPlacer-2Stage uses fast two-stage graph models to solve color assignment and cell placement iteratively. In addition, column "**TPLPlacer-MDP**" is to apply the linear dynamic programming method (see Section 3.4.2) to solve the TPL-OSR with maximum displacement problem. Here the maximum displacement value, M, is set to 8. For each algorithm we list several metrics "ST#", "ΔWL", and "CPU(s)". "ST#" is the stitch number on the final decomposed layout. "ΔWL" is the total wire-length difference before and after our TPL aware placement, where half-perimeter wire-length (HPWL) is applied to calculate the total wire-length. Column "CPU(s)" gives the detailed placement process runtime in seconds.

From column "GREEDY" we can see that the greedy method is very fast, i.e., if a legal solution is found it can be finished in less than 0.01 seconds. However, in 9 out of 21 cases it cannot find legal placement solutions. For each illegal result "N/A" is labeled in the table. The main reason for these illegal solutions is that GREEDY only shifts the cells right. Therefore, due to the greedy nature, for a benchmark case with high cell utilization it may cause final placement violation. Meanwhile, since the color assignment is solved through greedy method as well, it loses the global view to minimize the stitch number. We can observe that more stitches are reported for those cases where it finds out legal results.

We further compare two TPL-OSR algorithms: "TPLPlacer" and "TPLPlacer-2Stage". Comparing these two columns we can see that both of them can yield very similar wire-length improvement (around 1% wire-length reduction). In "TPLPlacer-2Stage" the unified graph is divided into two independent graphs, so the graph size can be reduced. Due to the smaller graph size, "TPLPlacer-2Stage" can get 100x speed-up against "TPLPlacer". However, "TPLPlacer-2Stage" introduces 19% more stitch numbers. The possible reason is that under the 2-stage graph model, placement and color assignment are optimized separately, and then this speed-up technique may lose some optimality in terms of stitch number.

From column "TPLPlacer-MDP" we can see that the linear dynamic programming technique has a better trade-off to optimize wire-length and stitch number together. That is, "TPLPlacer-MDP" achieves nearly the same wire-length and stitch number results comparing with "TPLPlacer". Meanwhile, "TPLPlacer-MDP" is 14× faster than the unified graph model in "TPLPlacer". The reason is that "TPLPlacer-MDP" is a linear runtime algorithm, while "TPLPlacer" has nearly a quadratic runtime complexity.

For test case ctl-70, Fig. 3.15 demonstrates three stitch density maps through different detailed placement algorithms: TPLPlacer, TPLPlacer-MDP, and TPLPlacer-2Stage. The final stitch numbers for these three detailed placement techniques are 335, 330 and 491, respectively. We can see that the density maps in Fig. 3.15 (a) and Fig. 3.15 (b) are very similar, which means the speed-up technique TPLPlacer-MDP can achieve very comparable result with TPLPlacer. However, another speed-up technique, TPLPlacer-2Stage, may involve more final stitches (see Fig. 3.15 (c)).

The "TPLPlacer-MDP" is implemented with $M = 8$. In other words, each cell c_i has $2M + 1$ possible new positions between $[x(i) - M, x(i) + M]$.

Since the M value determines the placement solution space, it impacts the performance of detailed placement a lot. Therefore, to demonstrate the robustness of "TPLPlacer-MDP", it would be interesting to analyze the performance with different M settings. Fig. 3.16 gives such analysis for test cases *alu_70, alu_80* and *alu_85.* From Fig. 3.16 (a) and Fig. 3.16 (b) we can see that with different M values, "TPLPlacer-MDP" can achieve similar stitch number and wire-length improvement. It is not hard to see from Fig. 3.16 (c) that the runtime is related to the M value, i.e., for each test case the runtime is nearly a linear function of M. Therefore, we can conclude that "TPLplacer-MDP" is very robust and insensitive to the M value. In our implementation, M is set as a small value 8, to maintain both good speed-up and good performance.

3.6 Summary

In this chapter, we propose a coherent framework to seamlessly integrate the TPL aware optimizations into early design stages. To our best knowledge, this is the first work for TPL compliance at both standard cell and placement levels. An optimal graph model to simultaneously solve cell placement and color assignment is proposed, and then a two-stage graph model is presented to achieve speedup. Our framework is compared with traditional layout decomposition. The results show that considering TPL constraints in early design stages can dramatically reduce the conflict number and stitch number in final layout. As continuing growth of technology node to sub-16 nm, TPL turns out to be a definitely promising lithography solution. A dedicated design flow integrating TPL constraints is necessary to assist in the whole process. We believe this work will stimulate more research on TPL aware design.

Table 3.5: Comparisons of Detailed Placement Algorithms

bench	GREEDY [36]			TPLPlacer			TPLPlacer-2Stage			TPLPlacer-MDP		
	ΔWL	ST#	CPU(s)	ΔWL	ST#	CPU(s)	ΔWL	ST#	CPU(s)	ΔWL	ST#	CPU(s)
alu-70	N/A	N/A	N/A	-1.67%	882	27.3	-1.69%	1105	0.48	-1.64%	883	4.8
alu-80	N/A	N/A	N/A	-0.6%	948	24.8	-0.62%	1099	0.43	-0.67%	943	4.6
alu-85	N/A	N/A	N/A	+2.0%	988	26.9	+1.98%	1085	0.43	+1.69%	991	3.8
byp-70	+0.002%	1764	0.001	-1.26%	1411	44.3	-1.27%	1979	0.88	-1.17%	1369	5.4
byp-80	+0.196%	1798	0.001	-0.7%	1568	43.7	-0.72%	1984	0.81	-0.76%	1557	5.0
byp-85	N/A	N/A	N/A	-0.6%	1676	82.8	-0.61%	1979	1.58	-0.62%	1679	9.6
div-70	+0.04%	1843	0.001	-1.65%	1566	22.9	-1.67%	1927	0.59	-1.62%	1558	3.1
div-80	N/A	N/A	N/A	-1.0%	1728	42.9	-1.04%	1957	1.06	-1.04%	1732	5.8
div-85	+0.8%	1846	0.001	-0.76%	1774	21.8	-0.77%	1961	0.53	-0.81%	1775	2.2
ecc-70	+0.05%	355	0.001	-1.52%	278	6.1	-1.52%	339	0.14	-1.50%	275	1.24
ecc-80	+0.24%	359	0.001	-0.8%	290	5.8	-0.82%	344	0.12	-0.81%	291	1.01
ecc-85	N/A	N/A	N/A	-0.49%	312	5.79	-0.48%	344	0.13	-0.55%	314	0.83
efc-70	+0.08%	890	0.001	-4.2%	755	5	-4.30%	901	0.11	-3.84%	747	1.04
efc-80	+0.85%	877	0.001	-2.35%	834	4.7	-2.39%	922	0.12	-2.24%	825	0.91
efc-85	N/A	N/A	N/A	-0.96%	856	5.1	-0.98%	927	0.11	-0.75%	852	0.79
ctl-70	-0.03%	487	0.001	-2.4%	335	8.6	-2.43%	491	0.20	-2.26%	330	1.42
ctl-80	+0.08%	468	0.001	-1.44%	385	8.1	-1.45%	492	0.18	-1.43%	383	1.27
ctl-85	N/A	N/A	N/A	-0.29%	423	16.1	-0.30%	489	0.35	-0.42%	418	2.42
top-70	+0.15%	6525	0.003	-1.5%	5597	196.9	-1.52%	7105	4.3	-1.46%	5565	12.5
top-80	+0.63%	6507	0.003	-0.7%	6166	193.3	-0.73%	7193	4.5	-0.75%	6143	11.0
top-85	N/A	N/A	N/A	-0.17%	6350	185.7	-0.18%	7075	3.9	-0.29%	6350	9.3
Average	N/A	N/A	N/A	-1.10%	1672.5	46.6	-1.12%	1986	0.50	-1.09%	1665.7	3.34
Ratio				**-1.0**	**1.0**	**1.0**	**-1.02**	**1.19**	**0.01**	**-0.99**	**1.00**	**0.07**

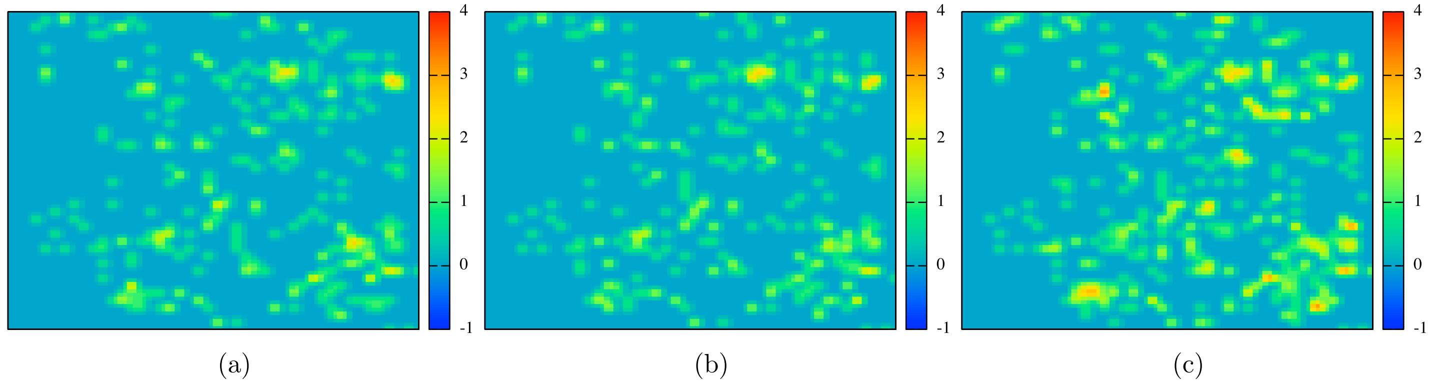
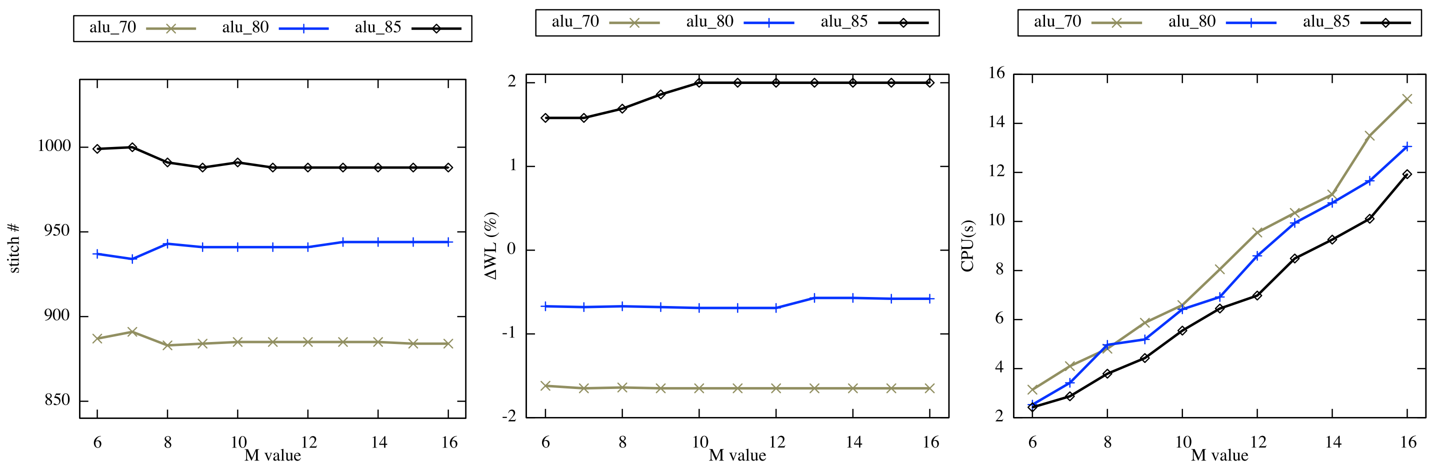


Figure 3.15: For test case ctl-70, three different stitch density maps through different detailed placement algorithms. (a) 335 stitches through TPLPlacer; (b) 330 stitches through TPLPlacer-MDP; (c) 491 stitches through TPLPlacer-2Stage.



(a) (b) (c)

Figure 3.16: TPLPlacer-MDP performance analyses with different M values for alu design cases. (a) Impact on stitch numbers. (b) Impact on wire-length improvements (ΔWL). (c) Impact on runtimes.

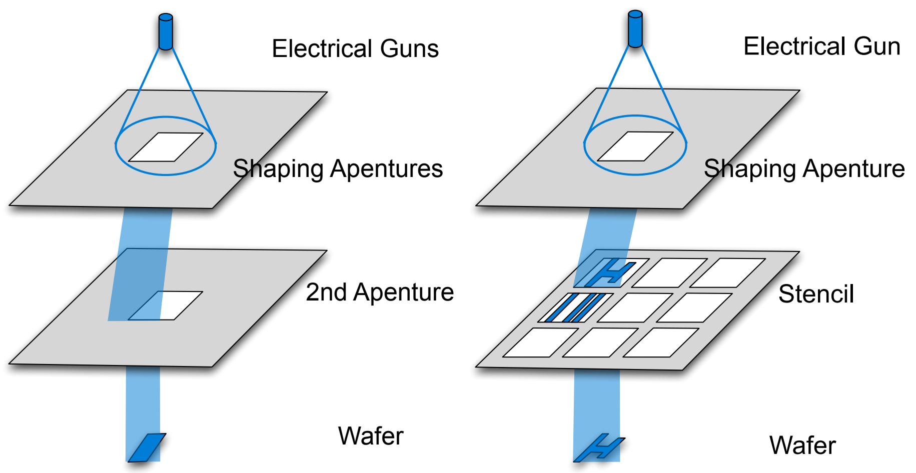
Chapter 4

Design for Manufacturing with E-Beam Lithography

4.1 Introduction

Conventional EBL system applies variable shaped beam (VSB) technique. In this mode, the entire layout is decomposed into a set of rectangles, each being shot into resist by one electron beam. The printing process of VSB mode is illustrated in Fig. 4.1 (a). At first the electrical gun generates an initial beam, which becomes uniform through the shaping aperture. Then the second aperture finalizes the target shape with a limited maximum size. Since each pattern needs to be fractured into pieces of rectangles and printed one by one, the VSB mode may suffer from serious throughput problem.

One improved technique is called character projection (CP) [34]. As shown in Fig. 4.1 (b), in CP mode the second aperture is replaced by a *stencil*. Some complex shapes, called *characters*, are prepared on the stencil. The key idea is that if a pattern is pre-designed on the stencil, it can be printed in one electronic shot, otherwise it needs to be fractured into a set of rectangles and printed one by one through VSB mode. By this way the CP mode can improve the throughput significantly. In addition, CP exposure has a good



(a) VSB (b) CP

Figure 4.1: Printing process of conventional EBL system. (a) VSB mode. (b) CP mode.

CD control stability compared with VSB [71]. However, the area constraint of stencil is the bottleneck. For modern design, due to the numerous distinct circuit patterns, only limited number of patterns can be employed on stencil. Those patterns not contained by stencil are still required to be written by VSB. Thus one emerging challenge in CP mode is how to pack the characters into stencil to effectively improve the throughput.

Even with decades of development, the key limitation of the EBL system has been and still is the low throughput. This chapter presents two methodologies to deal with the throughput problem in EBL system.

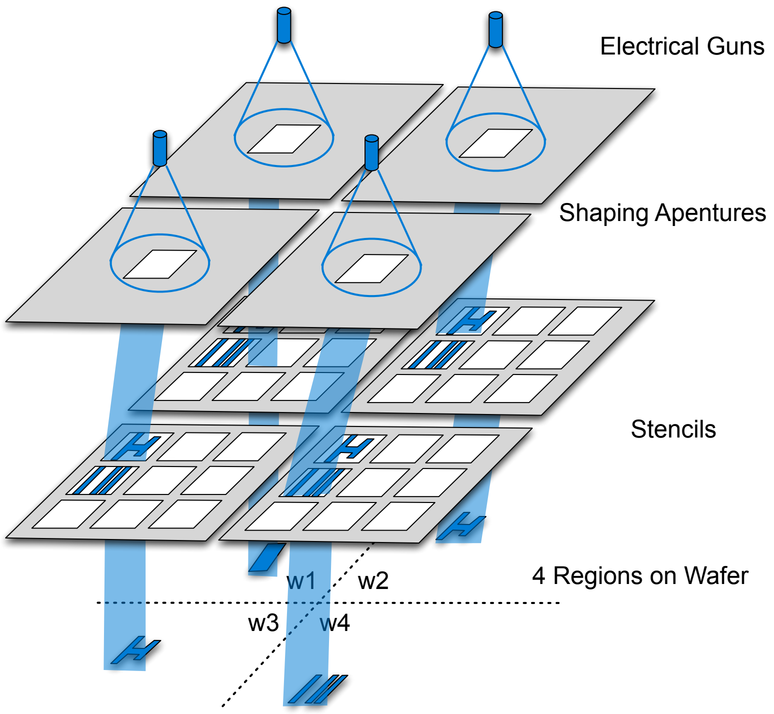


Figure 4.2: Printing process of MCC system, where four CPs are bundled.

4.2 E-BLOW: Overlapping aware Stencil Planning for MCC System

Even with decades of development, the key limitation of the EBL system has been and still is the low throughput. Recently, multi-column cell (MCC) system is proposed as an extension to CP technique [101] [72]. In MCC system, several independent character projections (CP) are used to further speed-up the writing process. Each CP is applied on one section of wafer, and all CPs can work parallelly to achieve better throughput. Due to the design complexity and cost consideration, different CPs share one stencil design [83]. One example of MCC printing process is illustrated in Fig. 4.2,

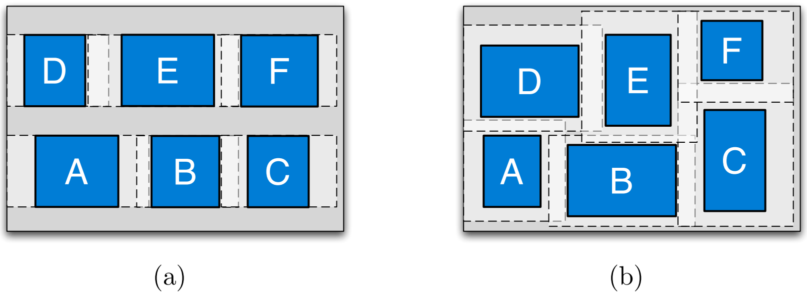


Figure 4.3: Two types of OSP problem. (a) 1D-OSP. (b) 2D-OSP.

where four CPs are bundled to generate an MCC system. In this example, the whole wafer is divided into four regions, w_1, w_2, w_3 and w_4, and each region is printed through one CP. Note that the whole writing time of the MCC system is determined by the maximum one of the four regions. For modern design, because of the numerous distinct circuit patterns, only limited number of patterns can be employed on stencil. Since the area constraint of stencil is the bottleneck, the stencil should be carefully designed/manufactured to contain the most repeated cells or patterns.

As one of the most challenges in CP mode, stencil planning has earned much attentions [22,57,68,84,113]. When blank overlapping is not considered, the stencil planning equals to a character selection problem. [84] proposed an integer linear programming (ILP) formulation to select a group of characters for throughput maximization. When the characters can be overlapped to save more stencil space, the corresponding stencil planning is referred as *overlapping-aware stencil planning* (OSP). [113] [57] investigated on OSP problem to place more characters onto stencil, Recently, [22] [68] assumed that the pattern position in each character can be shifted, and integrated the character re-design into OSP problem. As suggested in [113], the OSP problem can be divided into two types: *1D-OSP* and *2D-OSP*. In 1D-OSP, the standard cells with same height are selected into stencil. As shown in Fig. 4.3(a), each character implements one standard cell, and the enclosed circuit patterns of all the characters have the same height. Note that here we only show the horizontal blanks, and the vertical blanks are not represented because they are identical. In 2D-OSP, the blanking spaces of characters are non-uniform along both horizontal and vertical directions. By this way, stencil can contain both complex via patterns and regular wires. Fig. 4.3(b) illustrates a stencil design example for 2D-OSP.

Compared with conventional EBL system, MCC system introduces two main challenges in OSP problem. First, the objective is new: in MCC system the wafer is divided into several regions, and each region is written by one CP. Therefore the new OSP should minimize the maximal writing times of all regions. However, in conventional EBL system the objective is simply minimize the wafer writing time. Besides, the stencil for an MCC system can contain more than 4000 characters, previous methodologies for EBL system may suffer from runtime penalty. However, no existing stencil planning work has been done toward the MCC system.

This section presents E-BLOW, the first study for OSP problem in MCC system. Our main contributions are summarized as follows.

- We show that both 1D-OSP and 2D-OSP problems are NP-hard.
- We formulate integer linear programming (ILP) to co-optimizing characters selection and physical placements on stencil. To our best knowledge, this is the first mathematical formulation for both 1D-OSP and 2D-OSP.
- We proposes a simplified formulation for 1D-OSP, and proves its rounding lower bound theoretically.
- We present a successive relaxation algorithm to find a near optimal solution.
- We design a KD-Tree based clustering algorithm to speedup 2D-OSP solution.

4.2.1 Preliminaries

During character design, blanking area is usually reserved around its boundaries. Note that the blanking space here refers to the blank around the character boundaries. The term "overlapping" means sharing blanks between adjacent characters. By this way, more characters can be placed on the stencil [113]. In the beginning of this section the problem formulation will be provided, then we will show that both 1D-OSP and 2D-OSP are NP-hard.

4.2.1.1 Problem Formulation

In an MCC system with P CPs, the whole wafer is divided into P regions $\{r_1, r_2, \ldots, r_P\}$, and each region is written by one particular CP. We assume cell extraction [69] has been resolved first. In other words, a set of character candidates $C^C = \{c_1, \cdots, c_n\}$ has already been given to the MCC system. For each character candidate $c_i \in C^C$, its writing time through VSB mode is denoted as n_i, while its writing time through CP mode is 1.

The regions of wafer have different layout patterns, and the throughputs would be also different. Suppose character candidate c_i repeats t_{ic} times with MCC system on region r_c. Let a_i indicate the selection of character candidate c_i as follows.

$$a_i = \begin{cases} 1, & \text{candidate } c_i \text{ is selected on stencil} \\ 0, & \text{otherwise} \end{cases}$$

If c_i is prepared on stencil, the total writing time of pattern c_i on region r_c is $t_{ic} \cdot 1$. Otherwise, c_i should be printed through VSB. Since region r_c comprises t_{ic} candidate c_i, the writing time would be $t_{ic} \cdot n_i$. Therefore, for region r_c the total writing time T_c is as follows:

$$\begin{aligned} T_c &= \sum_{i=1}^{n} a_i \cdot (t_{ic} \cdot 1) + \sum_{i=1}^{n} (1 - a_i) \cdot (t_{ic} \cdot n_i) \\ &= \sum_{i=1}^{n} t_{ic} \cdot n_i - \sum_{i=1}^{n} t_{ic} \cdot (n_i - 1) \cdot a_i \\ &= T_c^{VSB} - \sum_{i=1}^{n} R_{ic} \cdot a_i \end{aligned}$$

where we denote $T_c^{VSB} = \sum_{i=1}^{n} t_{ic} \cdot n_i$, and $R_{ic} = t_{ic} \cdot (n_i - 1)$. T_c^{VSB} shows the writing time on r_c when only VSB is applied, and R_{ic} represents the writing time reduction of candidate c_i on region r_c. In MCC system, for each region r_c both T_c^{VSB} and R_{ic} are constants. Therefore, the total writing time of the MCC system is formulated as follows:

$$
\begin{aligned}
T_{total} &= \max\{T_c\} \\
&= \max\{T_c^{VSB} - \sum_{i=1}^{n} R_{ic} \cdot a_i\}, \forall c \in P
\end{aligned}
\tag{4.1}
$$

Problem 9. ***Overlapping aware Stencil Planning (OSP) for MCC System**: Given a set of character candidate C^C, select a subset C^{CP} out of C^C as characters, and place them on the stencil. The objective is to minimize the system writing time T_{total} expressed by Eqn. (4.1), while the placement of C^{CP} is bounded by the outline of stencil. The width and height of stencil is W and H, respectively.*

For convenience, we use the term OSP to refer OSP for MCC system in the rest of this chapter.

4.2.1.2 NP-Hardness

Here we discuss the computational complexity of the OSP problem. Before the details of proof, we define a Knapsack problem as follows.

Problem 10 (Knapsack). *Given a knapsack with capacity C and a set of items $S = \{1, \ldots, n\}$, where item i has size s_i and value v_i. We are searching a subset $S' \in S$ that maximizes the value of $\sum_{i \in S'} v_i$, such that $\sum_{i \in S'} s_i \leq C$. That is, all the items fit in a knapsack of size C.*

Knapsack problem is a well known NP-hard problem [70].

Lemma 7. *1D-OSP problem is NP-hard.*

Proof. We prove this lemma by showing Knapsack $\leq_P$ 1D-OSP, i.e., Knapsack can be reduced to 1D-OSP. Given an instance of Knapsack, its items $S = \{1, \ldots, n\}$, we construct a special 1D-OSP instance with: (a) Each item i in Knapsack is transferred to a character candidate c_i with size $s_i \times 1$. (b) Each character has "zero blank space", which means there is no blank around the character boundaries, and there is no overlapping between adjacent characters. (c) For character c_i, its writing time through VSB is set as $v_i + 1$, while the writing time through CP is set as 0. (d) The stencil area is set as $C \times 1$. (e) There is only one region, and each character c_i repeats one time in the region. After solving the Knapsack problem, if item i is selected into the knapsack, character c_i is also selected into the stencil. If the maximum value of the Knapsack problem is $V = \sum_{i \in S'} v_i$, the minimum system writing time in the 1D-OSP is $\sum v_i - V$. Thus maximizing the knapsack value in the Knapsack problem instance is equal to minimizing system writing time in the 1D-OSP instance, which completes the proof. □

Lemma 8. *2D-OSP problem is NP-hard.*

Since 1D-OSP is a special case of 2D-OSP. Due to the NP-hardness of 1D-OSP, the 2D-OSP problem is NP-hard as well.

Combining Lemma 7 and Lemma 8, we can achieve the conclusion that OSP problem, even for conventional EBL system, is NP-hard.

4.2.2 E-BLOW for 1D-OSP

When each character implements one standard cell, the enclosed circuit patterns of all the characters have the same height. The corresponding OSP problem is called 1D-OSP, which can be viewed as a combination of character selection and single row ordering problems [113]. Different from two-step heuristic proposed in [113], we show that these two problems can be solved simultaneously through a unified ILP formulation (4.2). For convenience, Table 4.1 lists the notations used in 1D-OSP problem.

Table 4.1: Notations used in 1D-ILP Formulation

W	width constraint of stencil or row
n	number of characters
m	number of rows
x_i	x-position of character c_i
w_i	width of character c_i
o^h_{ij}	horizontal overlap between c_i and c_j
p_{ij}	0-1 variable, $p_{ij} = 0$ if c_i is left of c_j
a_{ij}	0-1 variable, $a_{ij} = 1$ if c_i is on jth row

In formulation (4.2), W is the stencil width, m is the number of rows. For each character c_i, w_i and x_i are the width and the x-position, respectively. If and only if c_i is assigned to kth row, $a_{ik} = 1$. Otherwise, $a_{ik} = 0$. Constraints (4.2d) (4.2e) are used to check position relationship between c_i and c_j. Here $w_{ij} = w_i - o^h_{ij}$ and $w_{ji} = w_j - o^h_{ji}$, where o^h_{ij} is the overlapping when candidates c_i and c_j are packed together. For kth row, it is easy to see that only when $a_{ik} = a_{jk} = 1$, i.e. both character i and character j are assigned to row k, one of the two constraints (4.2d) (4.2e) will be active. The number of variables for

$$
\begin{aligned}
\min \quad & T_{total} && (4.2)\\
\text{s.t} \quad & T_{total} \ge T_c^{VSB} - \sum_{i=1}^{n}(\sum_{k=1}^{m} R_{ic} \cdot a_{ik}), \ \forall c \in P && (4.2a)\\
& 0 \le x_i \le W - w_i, \quad \forall i && (4.2b)\\
& \sum_{k=1}^{m} a_{ik} \le 1, \qquad \forall i && (4.2c)\\
& x_i + w_{ij} - x_j \le W(2 + p_{ij} - a_{ik} - a_{jk}) && (4.2d)\\
& x_j + w_{ji} - x_i \le W(3 - p_{ij} - a_{ik} - a_{jk}) && (4.2e)\\
& a_{ik}, a_{jk}, p_{ij} : 0-1 \text{ variable } \ \forall i, j && (4.2f)
\end{aligned}
$$

(4.2) is $O(n^2)$, where n is the number of character candidates.

Since ILP is a well known NP-hard problem, directly solving it may suffer from long runtime penalty. One straightforward speedup method is to relax the ILP into the corresponding linear programming (LP) through replacing constraints $(4.2f)$ by the following:

$$0 \le a_{ik}, a_{jk}, p_{ij} \le 1$$

It is obvious that the LP solution provides a lower bound to the ILP solution. However, we observe that the solution of relaxed LP could be like this: for each i, $\sum_j a_{ij} = 1$ and all the p_{ij} are assigned 0.5. Although the objective function is minimized and all the constraints are satisfied, this LP relaxation provides no useful information to guide future rounding, i.e., all the character candidates are selected and no ordering relationship is determined.

To overcome the limitation of above rounding, E-BLOW proposes a

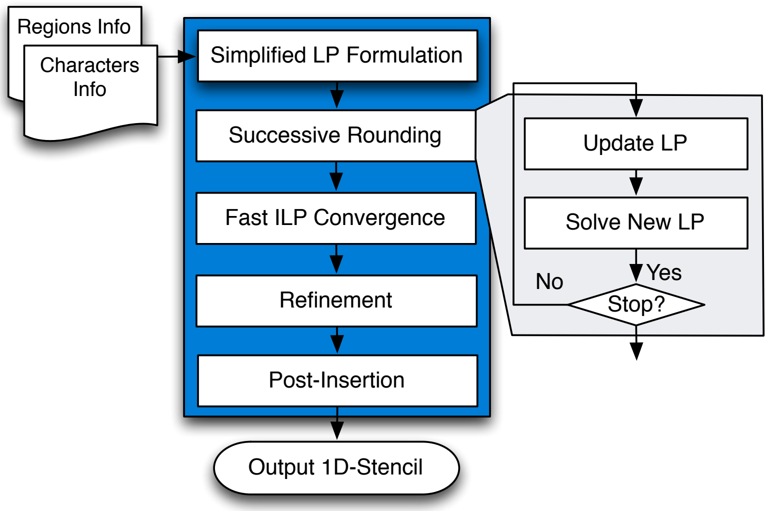


Figure 4.4: E-BLOW overall flow for 1D-OSP.

novel successive rounding framework to search near-optimal solution in reasonable runtime. The main idea is to modify the ILP formulation, so that the corresponding LP relaxation can provide good lower bound theoretically.

As shown in Fig. 4.4, the overall flow includes five parts: Simplified ILP formulation, Successive Rounding, Fast ILP Convergence, Refinement, and Post-Insertion. In section 4.2.2.1 the simplified formulation will be discussed, and its LP rounding lower bound will be proved. In section 4.2.2.2 the details of successive rounding would be introduced. At last, to further improve the performance, section 4.2.2.3 propose refinement and post-insertion.

4.2.2.1 Simplified ILP Formulation

The simplified formulation in E-BLOW is based on a *symmetrical blank* (S-Blank) assumption: the blanks of each character are symmetry, i.e., left blank equals to right blank. s_i is used to denote the blank of character c_i. Note that for different characters c_i and c_j, their blanks s_i and s_j can be different.

At first glance the S-Blank assumption may lose optimality. However, it provides several practical and theoretical benefits. (1) In [113] the single row ordering problem was transferred into Hamilton Cycle problem, which is a well known NP-hard problem and even particular solver is quite expensive. In our work, instead of relying on expensive solver, under this assumption the problem can be optimally solved in $O(n)$. (2) Under S-Blank assumption, the ILP formulation can be effectively simplified to provide a reasonable rounding bound theoretically. Compared with previous heuristic framework [113], the proved rounding bound provides a better guideline for a global view search. (3) To compensate the inaccuracy in the asymmetrical blank cases, E-BLOW provides a refinement (see section 4.2.2.3).

Given p character candidates, single row ordering problem adjusts the relative locations to minimize the total width. Under S-Blank assumption, this problem can be optimally solved through the following two-step greedy approach.

1. All characters are sorted decreasingly by blanks;
2. All characters are inserted one by one. Each one can be inserted at either left end or right end.

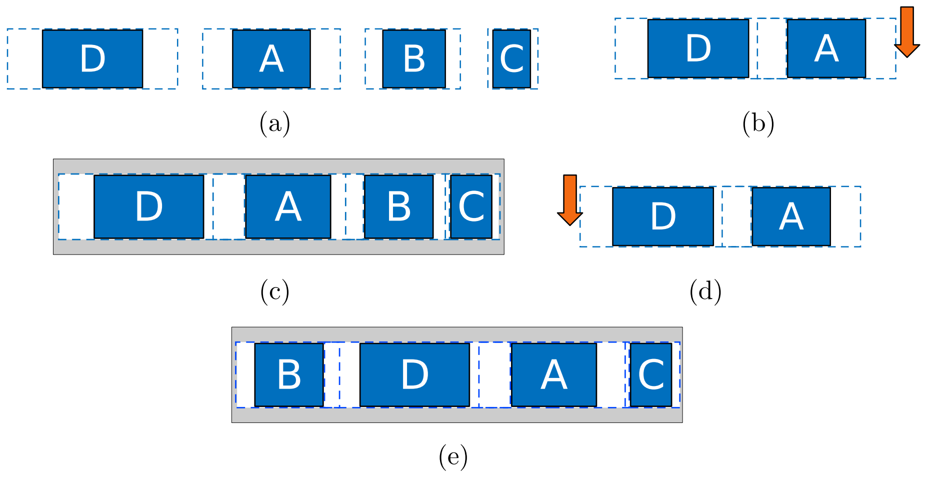


Figure 4.5: Greedy based Single Row Ordering. (a) At first all candidates are sorted by blanking space. (c) One possible ordering solution where each candidate chooses the right end position. (e) Another possible ordering solution.

One example of the greedy approach is illustrated in Fig. 4.5, where four character candidates A, B, C and D are to be ordered. In Fig. 4.5(a), they are sorted decreasingly by blanking space. Then all the candidates are inserted one by one. From the second candidate, each insertion has two options: left side or right side of the whole packed candidates. For example, if A is inserted at the right of D, B has two insertion options: one is at the right side of A (Fig. 4.5(b)), another is at the left side of A (Fig. 4.5(d)). Given different choices of candidate B, Fig. 4.5(c) and Fig. 4.5(e) give corresponding final solutions. Since from the second candidate each one has two choices, by this greedy approach n candidates will generate 2^{n-1} possible solutions. But following theorem shows that under the symmetry blank assumption, all these solutions have the same length.

Theorem 6. *Under S-Blank assumption, the greedy approach can get maximum overlapping space* $\sum_i s_i - max\{s_i\}$.

Proof. It can be proved by recursion. For p candidates, each one c_i is with blank space s_i. Without loss of generality we set $s_1 \geq s_2 \geq \cdots \geq s_p$. Then we try to prove the maximum overlapping space is $f(p) = \sum_{i=2}^{p} s_i$. If $p = 2$ the theorem is trivial and $f(2) = s_2$. We assume that when $p = n - 1$, the maximum overlapping space $f(n-1) = \sum_{i=2}^{n-1} s_i$. When $p = n$, since the last candidate can only be inserted at either the left end or the right end, and for any $i < n$, $s_i \geq s_n$, we can find the incremental overlapping space is s_n. Then $f(n) = f(n-1) + s_n = \sum_{i=2}^{n} s_i$. □

In practical, we set $s_i = \lceil (sl_i + sr_i)/2 \rceil$, where sl_i and sr_i are c_i's left blank and right blank, respectively.

In E-BLOW, the original ILP formulation (4.2) is relaxed into formulation (4.3).

$$\max \sum_i \sum_j a_{ij} \cdot profit_i \tag{4.3}$$

$$\text{s.t.} \quad \sum_i (w_i - s_i) \cdot a_{ij} \leq W - B_j, \forall j \tag{4.3a}$$

$$B_j \geq s_i \cdot a_{ij}, \forall i \tag{4.3b}$$

$$\sum_j a_{ij} \leq 1, \quad \forall c_i \in C^C \tag{4.3c}$$

$$a_{ij} = 0 \text{ or } 1 \quad \forall i, j \tag{4.3d}$$

In formulation (4.3), (4.3c) implies each character can be assigned into at most one row. It's easy to see that the number of variables is $O(nm)$, where n is the number of characters, and m is the number of rows. Generally speaking, single character number n is much larger than row number m. Thus compared with basic ILP formulation (4.2), the variable number in (4.3) can be reduced dramatically.

The difference between (4.3) and (4.2) is twofold. On the one hand, due to Theorem 6, constraint (4.3a) and constraint (4.3b) are for row width calculation, where (4.3b) is to linearize max operation. Here B_j can be viewed as the maximum blank space of all the characters on row r_j. On the other hand, through assigning each character c_i with one profit value $profit_i$, we can simplify the complex constraint (4.2a). Section 4.2.2.2 will discuss how to assign $profit_i$ to each character c_i. Based on these simplifications, we will show that the LP relaxation of (4.3) has reasonable lower bound. To explain this, let us first look at a similar formulation (3′) as follows:

$$\begin{aligned}
\max \ & \sum_i \sum_j a_{ij} \cdot profit_i && (3') \\
\text{s.t.} \ & \sum_i (w_i - s_i) \cdot a_{ij} \le W - max_s && (3'a) \\
& (3c) - (3d)
\end{aligned}$$

where max_s is the maximum horizontal blank length of every character, i.e. $max_s = \max\{s_i | i = 1, 2, \ldots, n\}$. Program (3′) is a multiple knapsack problem [70]. A multiple knapsack is similar to a knapsack problem, with the difference that there are multiple knapsacks. In formulation (3′), each $profit_i$ can be rephrased as $(w_i - s_i) \times ratio_i$. If all $ratio_i$ are the same, formulation $(3')$ can be approximated to a max-flow problem. We have the following lemma:

Lemma 9. *If each $ratio_i$ is the same, the multiple knapsack problem $(3')$ can find a $1/2-$approximation algorithm using LP rounding method.*

For brevity we omit the proof, detailed explanations can be found in [25]. The difference between (4.3) and $(3')$ is twofold. First, the right side values at $(4.3a)$ and $(3'a)$. Blank spacing is relatively small comparing with the row length, we can get that $W - max_s \approx W - B_j$. Second, the $ratio_i$ values in objective functions. Then we can conclude that program $(3')$ has a reasonable rounding bound.

4.2.2.2 Successive Rounding

We propose a successive rounding algorithm to solve program (4.3) iteratively. Successive rounding uses a simple iterative scheme in which fractional variables are rounded one after the other until an integral solution is found [47]. The ILP formulation (4.3) becomes an LP if we relax the discrete constraint to a continuous constraint as: $0 \leq a_{ij} \leq 1$.

The details of successive rounding is shown in Algorithm 15. At first we set all a_{ij} as *unsolved* since none of them is assigned to rows. The LP is updated and solved iteratively. For each new LP solution, we search the maximal a_{ij}, and store in a_{pq} (line 6). Then we find all a_{ij} that is closest to the maximum value a_{pq}, i.e., $a_{ij} \geq a_{pq} \times th_{inv}$. In our implementation, th_{inv}

Algorithm 15 SuccRounding (th_{inv})

```
Require: ILP Formulation (4.3)
 1: Set all $a_{ij}$ as unsolved;
 2: repeat
 3:     Update $profit_i$ for all unsolved $a_{ij}$;
 4:     Solve relaxed LP of (4.3);
 5:     repeat
 6:         $a_{pq} \leftarrow \max\{a_{ij}\}$;
 7:         for all $a_{ij} \geq a_{pq} \times th_{inv}$ do
 8:             if $c_i$ can be assigned to row $r_j$ then
 9:                 $a_{ij} = 1$ and set it as solved;
10:                 Update capacity of row $r_j$;
11:             end if
12:         end for
13:     until cannot find $a_{pq}$
14: until
```

is set to 0.9. For each selected variables a_{ij}, we try to pack c_i into row r_j, and set a_{ij} as *solved*. Note that when one character c_i is assigned to one row, all a_{ij} would be set as solved. Therefore, the variable number in updated LP formulation would continue to decrease. This procedure repeats until no appropriate a_{ij} can be found. One key step of Algorithm 15 is the $profit_i$ update (line 3). For each character c_i, we set its $profit_i$ as follows:

$$profit_i = \sum_c \frac{t_c}{t_{max}} \cdot (n_i - 1) \cdot t_{ic} \tag{4.4}$$

where t_c is current writing time of region r_c, and $t_{max} = \max\ \{t_c, \forall c \in P\}$. Through applying the $profit_i$, the region r_c with longer writing time would be considered more during the LP formulation. If c_i is not assigned to any row, $profit_i$ would continue to be updated, so that the total writing time of the whole MCC system can be minimized.

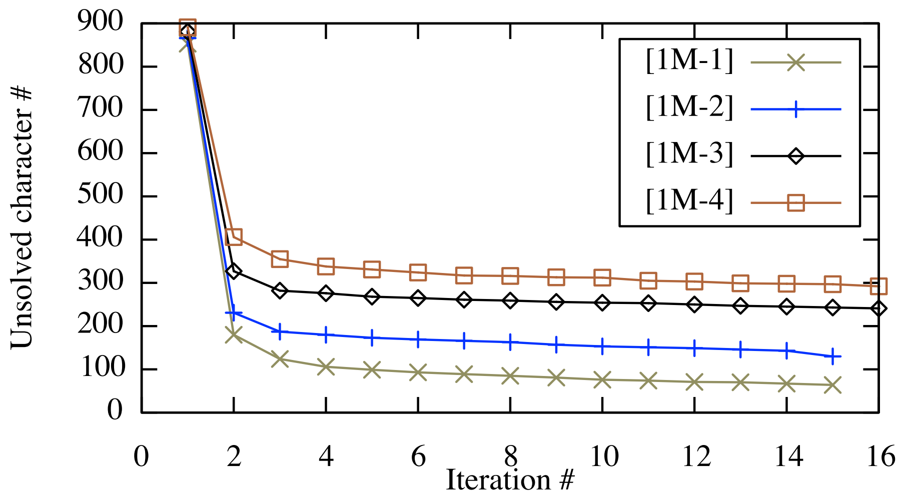


Figure 4.6: Unsolved character number along the LP iterations for testcases 1M-1, 1M-2, 1M-3, and 1M-4.

During successive rounding, for each LP iteration, we select some characters into rows, and set these characters as solved. In the next LP iteration, only unsolved characters would be considered in formulation. Thus the number of unsolved characters continues to decrease through the iterations. For four test cases (1M-1 to 1M-4), Fig. 4.6 illustrates the number of unsolved characters in each iteration. We observe that in early iterations, more characters would be assigned to rows. However, when the stencil is almost full, fewer of a_{ij} could be close to 1. Thus, in late iterations only few characters would be assigned into stencil, and the successive rounding requires more iterations.

To overcome this limitation so that the successive rounding iteration number can be reduced, we present a convergence technique based on fast ILP formulation. The basic idea is that when we observe only few characters are assigned into rows in one LP iteration, we stop successive rounding in advance,

Algorithm 16 Fast ILP Convergence (L_{th}, U_{th})

```
Require: Solutions of relaxed LP (4.3);
1: for all $a_{ij}$ in relaxed LP solutions do
2:     if $a_{ij} < L_{th}$ then
3:         Set $a_{ij}$ as solved;
4:     end if
5:     if $a_{ij} > U_{th}$ then
6:         Assign $c_i$ to row $r_j$;
7:         Set $a_{ij}$ as solved;
8:     end if
9: end for
10: Solve ILP formulation (4.3) for all unsolved $a_{ij}$
11: if $a_{ij} = 1$ then
12:     Assign $c_i$ to row $r_j$;
13: end if
```

and call fast ILP convergence to assign all left characters. Note that in [57] an ILP formulation with similar idea was also applied. The details of the ILP convergence is shown in Algorithm 16. The input are the solutions of last LP rounding, and two parameters L_{th} and U_{th}. First we check each a_{ij} (lines 1-9). If $a_{ij} < L_{th}$, then we assume character c_i would be not assigned to row r_j, and set a_{ij} as solved. Similarly, if $a_{ij} > U_{th}$, we assign c_i to row r_j and set a_{ij} as solved. For those unsolved a_{ij} we build up ILP formulation (4.3) to assign final rows (lines 10-13).

At first glance the ILP formulation may be expensive to solve. However, we observe that in our convergence Algorithm 16, typically the variable number is small. Fig. 4.7 illustrates the solution distribution in last LP formulation. We can see that most of the values are close to 0. In our implementation L_{th} and U_{th} are set to 0.1 and 0.9, respectively. For this case, although the

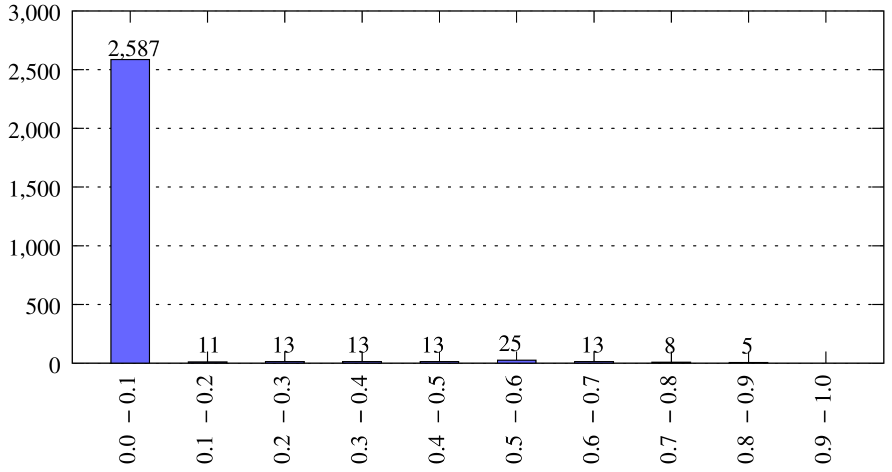


Figure 4.7: For test case 1M-1, solution distribution in last LP, where most of values are close to 0.

LP formulation contains more than 2500 variables, our fast ILP formulation results in only 101 binary variables.

4.2.2.3 Refinement

Simplified formulation, successive relaxation and fast convergence are all under the symmetrical blank assumption. Although the problem can be effectively solved, for asymmetrical cases it would lose optimality. Under symmetrical blank space assumption, all these orderings in simplified LP formulation get the same length. But for the asymmetrical cases, it does not hold anymore. To compensate the losing, E-BLOW consists of a refinement stage. For n characters $\{c_1, \ldots, c_n\}$, single row ordering can have $n!$ possible solutions. We avoid enumerating such huge solutions, and take advantage of the order in symmetrical blank assumption. That is, we pick up one best solution from the 2^{n-1} possible ones. Noted that although considering 2^{n-1} instead of $n!$ options cannot guarantee optimal single row packing, our preliminary results show that the solution quality loss is negligible in practice.

The refinement is based on dynamic programming, and the details are shown in Algorithm 17. Refine(k) generates all possible order solutions for the first k characters $\{c_1, \ldots, c_k\}$. Each order solution is represented as a set (w, l, r, O), where w is the total length of the order, l is the left blank of the left character, r is the right blank of the right character, and O is the character order. At the beginning, an empty solution set S is initialized (line 1). If $k = 1$, then an initial solution $(w_1, sl_1, sr_1, \{c_1\})$ would be generated (line 2). Here w_1, sl_1, and sr_1 are width of first character c_1, left blank of c_1, and right blank of c_1. If $k > 1$, then *Refine*(k) will recursively call *Refine*(k-1) to generate all old partial solutions. All these partial solutions will be updated by adding candidate c_k (lines 5-9).

We propose pruning techniques to speed-up the dynamic programming process. Let us introduce the concept of inferior solutions. For any two solutions $S_A = (w_a, l_a, r_a, O_a)$ and $S_B = (w_b, l_b, r_b, O_b)$, we say S_B is **inferior** to S_A if and only if $w_a \geq w_b$, $l_a \leq l_b$ and $r_a \leq r_b$. Those inferior solutions would be pruned during pruning section (lines 10-12). In our implementation, the *threshold* is set to 20.

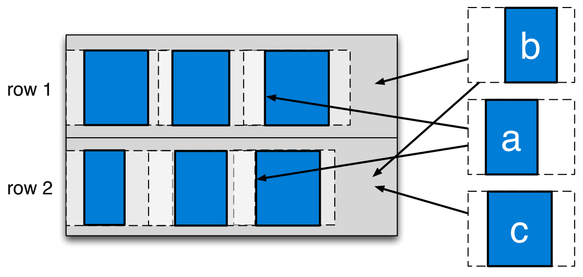
After refinement, a post-insertion stage is applied to further insert more characters into stencil. Different from the greedy insertion approach in [113] that new characters can be only inserted into one row's right end. We con-

Algorithm 17 Refine(k)

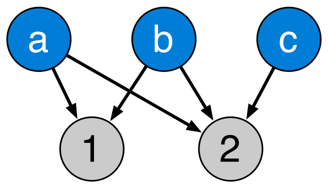
```
Require: k characters $\{c_1, \dots, c_k\}$;
 1: if k = 1 then
 2:     Add $(w_1, sl_1, sr_1, \{c_1\})$ into $S$;
 3: else
 4:     Refine(k-1);
 5:     for each partial solution $(w, l, r, O)$ do
 6:         Remove $(w, l, r, O)$ from $S$;
 7:         Add $(w + w_k - \min(sr_k, l), sl_k, r, \{c_k, O\})$ into $S$;
 8:         Add $(w + w_k - \min(sl_k, r), l, sr_k, \{O, c_k\})$ into $S$;
 9:     end for
10:     if $S$ size $\geq$ threshold then
11:         Prune inferior solutions in $S$;
12:     end if
13: end if
```

sider to insert characters into the middle part of rows. Generally speaking, the character with higher profit value (4.4) would have a higher priority to be inserted into rows. We assume that each row can introduce at most one additional character, and formulate the insertion as a maximum weighted matching problem [35].

Fig. 4.8 illustrates one example of the character insertion. As shown in Fig. 4.8 (a), there are two rows (row 1, row 2) and three additional characters (a, b, c). Characters a and b can be inserted into either row 1 or row 2, but character c can only be inserted into row 2. It shall be noted that the insertion position is labeled by arrows. For example, two arrows from character a mean that a can be inserted into the middle of each row. We build up a bipartite graph to represent the relationships among characters and rows (see Fig. 4.8 (b)). Each edge is associated with a cost as character's profit. By utilizing



(a)



(b)

Figure 4.8: Example of maximum weighted matching based post character insertion. (a) Three additional characters a, b, c and two rows. (b) Corresponding bipartite graph to represent the relationships among characters and rows.

the bipartite graph, the best character insertion can be solved by finding a maximum weighted matching.

Given n additional characters, we search the possible insertion position under each row. The total search time needs $O(nmC)$ time, where m is the total row number and C is the maximum character number on each row. We propose two heuristics to speed-up the search process. First, to reduce n, we only consider those additional characters with high profits. Second, to reduce m, we skip those rows with very little empty spaces.

4.2.3 E-BLOW for 2D-OSP

Now we consider a more general case: the blanking spaces of characters are non-uniform along both horizontal and vertical directions. This problem is referred to 2D-OSP problem. In [113] the 2D-OSP problem was transformed into a floorplanning problem. However, several key differences between traditional floorplanning and OSP were ignored. (1) In OSP there is no wirelength to be considered, while at floorplanning wirelength is a major optimization objective. (2) Compared with complex IP cores, lots of characters may have similar sizes. (3) Traditional floorplanner could not handle the problem size of modern MCC design.

4.2.3.1 ILP Formulation

Table 4.2: Notations used in 2D-ILP Formulation

$W(H)$	width (height) constraint of stencil
$w_i(h_i)$	width (height) of candidate c_i
$o^h_{ij}(o^v_{ij})$	horizontal (vertical) overlap between c_i and c_j
$w_{ij}(h_{ij})$	$w_{ij} = w_i - o^h_{ij}$, $h_{ij} = h_i - o^v_{ij}$
a_i	0-1 variable, $a_i = 1$ if c_i is on stencil

Here we will show that 2D-OSP can be formulated as integer linear programming (ILP) as well. Compared with 1D-OSP, 2D-OSP is more general: the blanking spaces of characters are non-uniform along both horizontal and vertical directions. The 2D-OSP problem can be also formulated as an ILP formulation (4.5). For convenience, Table 4.2 lists some notations used in the ILP formulation. The formulation is motivated by [86], but the difference is that our formulation can optimize both placement constraints and character selection, simultaneously. where a_i indicates whether candidate c_i is on the stencil, p_{ij} and q_{ij} represent the location relationships between c_i and c_j. The number of variables is $O(n^2)$, where n is number of characters. We can see that if $a_i = 0$, constraints (4.5b) - (4.5e) are not active. Besides, it is easy to

$$
\begin{aligned}
&\min\ T_{total} && (4.5)\\
&\text{s.t.}\quad T_{total} \geq T_c^{VSB} - \sum_{i=1}^{n} R_{ic}\cdot a_i, \quad \forall c \in P && (4.5a)\\
&\qquad x_i + w_{ij} \leq x_j + W(2 + p_{ij} + q_{ij} - a_i - a_j) && (4.5b)\\
&\qquad x_i - w_{ji} \geq x_j - W(3 + p_{ij} - q_{ij} - a_i - a_j) && (4.5c)\\
&\qquad y_i + h_{ij} \leq y_j + H(3 - p_{ij} + q_{ij} - a_i - a_j) && (4.5d)\\
&\qquad y_i - h_{ji} \geq y_j - H(4 - p_{ij} - q_{ij} - a_i - a_j) && (4.5e)\\
&\qquad 0 \leq x_i + w_i \leq W, \quad 0 \leq y_i + h_i \leq H && (4.5f)\\
&\qquad p_{ij}, q_{ij}, a_i : \text{0-1 variable } \forall i, j && (4.5g)
\end{aligned}
$$

see that when $a_i = a_j = 1$, for each of the four possible choices of $(p_{ij}, q_{ij}) = (0,0), (0,1), (1,0), (1,1)$, only one of the four inequalities $(4.5b)$ - $(4.5e)$ are active. For example, with $(a_i, a_j, p_{ij}, q_{ij})$ = (1,1,1,1), only the constraint $(4.5e)$ applies, which allows character c_i to be anywhere above character c_j. The other three constraints $(4.5b)$-$(4.5d)$ are always satisfied for any permitted values of (x_i, y_i) and (x_j, y_j).

Program (4.5) can be relaxed to linear programming (LP) by replacing constraint $(4.5g)$ as:

$$0 \leq p_{ij}, q_{ij}, a_i \leq 1$$

However, similar to the discussion in 1D-OSP, the relaxed LP solution provides no information or guideline to the packing, i.e., every a_i is set as 1, and every p_{ij} is set as 0.5. In other words, this LP relaxation provides no useful information to guide future rounding: all the character candidates are selected and no ordering relationship is determined. Therefore we can see that LP rounding method cannot be effectively applied to program (4.5).

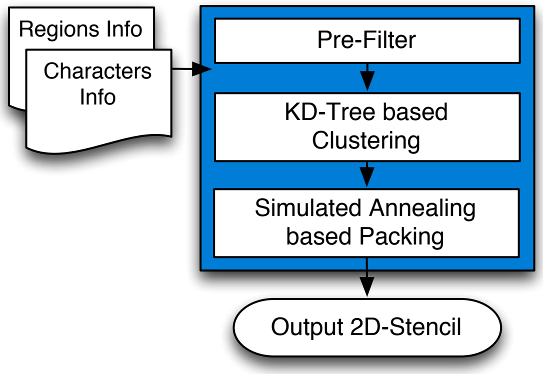


Figure 4.9: E-BLOW overall flow for 2D-OSP.

4.2.3.2 Clustering based Simulated Annealing

To deal with all these limitations of ILP formulation, an fast packing framework is proposed (see Fig. 4.9). Given the input character candidates, the pre-filter process is first applied to remove characters with bad profit (defined in (4.4)). Then the second step is a clustering algorithm to effectively speed-up the design process. Followed by the final floorplanner to pack all candidates.

Clustering is a well studied problem, and there are many of works and applications in VLSI [8] However, previous methodologies cannot be directly applied here. First, traditional clustering is based on netlist, which provides the all clustering options. Generally speaking, netlist is sparse, but in OSP the connection relationships are so complex that any two characters can be clustered, and totally there are $O(n^2)$ clustering options. Second, given two candidates c_i and c_j, there are several clustering options. For example, horizontal clustering and vertical clustering may have different overlapping space.

The details of our clustering procedure are shown in Algorithm 18. The clustering is repeated until no characters can be further merged. Initially all the candidates are sorted by $profit_i$, so those candidates with more shot number reduction are tend to be clustered. Then clustering (lines 3-8) is carried out, where we iteratively search all character pair (c_i, c_j) with similar blank spaces, profits, and sizes. Character c_i is said to be similar to c_j, if the following condition is satisfied:

$$\begin{cases} \max\{|w_i - w_j|/w_j, |h_i - h_j|/h_j\} \leq bound \\ \max\{|sh_i - sh_j|/sh_j, |sv_i - sv_j|/sv_j\} \leq bound \\ |profit_i - profit_j|/profit_j \leq bound \end{cases} \tag{4.6}$$

where w_i and h_i are the width and height of c_i. sh_i and sv_i are the horizontal space and vertical space of c_i, respectively. In our implementation, *bound* is set as 0.2. We can see that in clustering, all the size, blanks, and profits are considered.

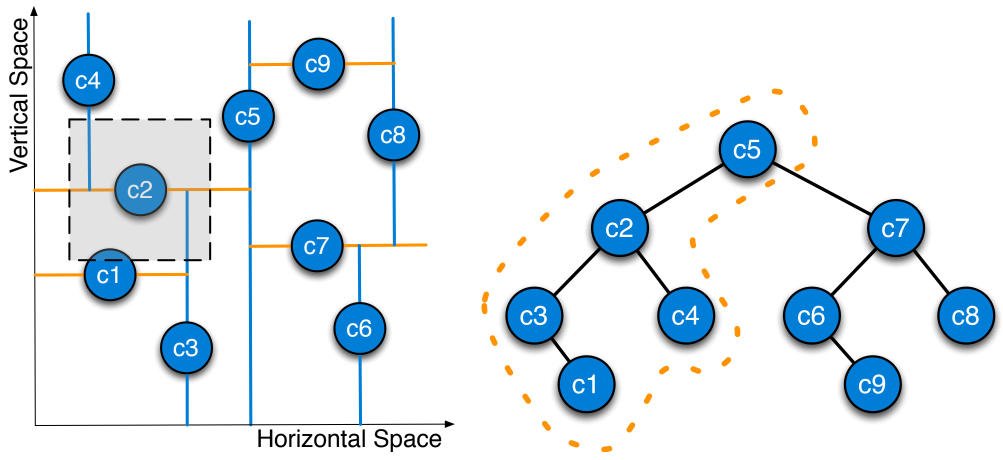
Algorithm 18 KD-Tree based Clustering

```
Require: set of candidates $C^C$.
1: repeat
2:     Sort all candidates by $profit_i$;
3:     Set each candidates $c_i$ to unclustered;
4:     for all unclustered candidate $c_i$ do
5:         Search all similar character pairs $(c_i, c_j)$;
6:         Cluster $(c_i, c_j)$, label them as clustered;
7:     end for
8:     Update candidate information;
9: until No characters can be merged
```

For each candidate c_i, finding available c_j may need $O(n)$, and complexity of the horizontal clustering and vertical clustering are both $O(n^2)$. Then the complexity of the whole procedure is $O(n^2)$, where n is the number of candidates.

A KD-Tree [14] is used to speed-up the process of finding available pair (c_i, c_j). It provides fast $O(logn)$ region searching operations which keeping the time for insertion and deletion small: insertion, $O(logn)$; deletion of the root, $O(n(k-1)/k)$; deletion of a random node, $O(logn)$. Using KD-Tree, the complexity of the Algorithm 18 can be reduced to $O(nlogn)$. For instance, given nine character candidates $\{c_1, \ldots, c_9\}$, the corresponding KD-Tree is shown in Fig. 4.10 (a). For the sake of convenience, here characters are distributed only based on horizontal and vertical spaces. The edges of KD-Tree are labelled as well. To search candidates with similar space with c_2 (see the shaded region of Fig. 4.10 (a)), it may need $O(n)$ time to scan all candidates, where n is the total candidate number. However, under the KD-Tree structure, this search procedure can be resolved in $O(logn)$. Particularly, all candidates scanned $(c_1 - c_5)$ are illustrated in Fig. 4.10 (b).

In [113], the 2D-OSP is transformed into a fixed-outline floorplanning problem. If a character candidate is outside the fixed-outline, then the character would not be prepared on stencil. Otherwise, the character candidate would be selected and packed on stencil. Parquet [7] was adopted as simulated annealing engine, and Sequence Pair [74] was used as a topology representation. In E-BLOW we apply a simulated annealing based framework similar to that in [113]. To demonstrate the effectiveness of our pre-filter and clustering



(a) (b)

Figure 4.10: KD-Tree based region searching.

methodologies, E-BLOW uses the same parameters.

4.2.4 Experimental Results

E-BLOW is implemented in C++ programming language and executed on a Linux machine with two 3.0GHz CPU and 32GB Memory. GUROBI [43] is used to solve ILP/LP. The benchmark suite from [113] are tested (1D-1, . . . , 1D-4, 2D-1, . . . , 2D-4). To evaluate the algorithms for MCC system, eight benchmarks (1M-x) are generated for 1D-OSP and the other eight (2M-x) are generated for the 2D-OSP problem. In these new benchmarks, character projection (CP) number are all set to 10. For each small case (1M-1, . . . , 1M-4, 2M-1, . . . , 2M-4) the character candidate number is 1000, and the stencil size is set to $1000\mu m \times 1000\mu m$. For each larger case (1M-5 , . . . , 1M-8, 2M-5, . . . , 2M-8) the character candidate number is 4000, and the stencil size is set to $2000\mu m \times 2000\mu m$. The size and the blank width of each character are

Table 4.3: Result Comparison for 1D-OSP

	char	CP	Greedy in [113]			[113]			[57]			E-BLOW		
	#	#	T	char#	CPU(s)	T	char#	CPU(s)	T	char#	CPU(s)	T	char#	CPU(s)
1D-1	1000	1	64891	912	0.1	50809	926	13.5	**19095**	940	0.005	19479	940	1.9
1D-2	1000	1	99381	884	0.1	93465	854	11.8	35295	864	0.005	**34974**	866	1.6
1D-3	1000	1	165480	748	0.1	152376	749	9.13	69301	757	0.005	**67209**	766	1.6
1D-4	1000	1	193881	691	0.1	193494	687	7.7	**92523**	703	0.005	93766	703	4.6
1M-1	1000	10	63811	912	0.1	53333	926	13.5	39026	938	0.01	**36800**	945	3.2
1M-2	1000	10	104877	884	0.1	95963	854	11.8	77997	864	0.01	**75303**	874	3.0
1M-3	1000	10	172834	748	0.1	156700	749	9.2	138256	758	0.56	**132773**	774	7.8
1M-4	1000	10	200498	691	0.1	196686	687	7.7	176228	698	0.36	**173620**	711	8.8
1M-5	4000	10	274992	3604	1.0	255208	3629	1477.3	204114	3660	0.03	**201492**	3681	34.0
1M-6	4000	10	437088	3341	1.0	417456	3346	1182	357829	3382	0.03	**348007**	3420	45.0
1M-7	4000	10	650419	3000	1.0	644288	2986	876	568339	3016	0.59	**559655**	3070	43.4
1M-8	4000	10	820013	2756	1.0	809721	2734	730.7	731483	2760	0.42	**721149**	2818	49.5
Avg.	-	-	270680.4	1597.6	0.4	259958.3	1594.0	362.5	209123.8	1611.7	0.17	205352.3	1630.7	16.6
Ratio	-	-	**1.32**	0.98	0.03	**1.27**	0.98	21.9	**1.02**	0.99	0.01	**1.0**	1.0	1.0

similar to those in [113].

For 1D-OSP, Table 4.3 compares E-BLOW with the greedy method in [113], the heuristic framework in [113], and the algorithms in [57]. We have obtained the programs of [113] and executed them in our machine. The results of [57] are directly from their paper. Column "char #" is number of character candidates, and column "CP#" is number of character projections. For each algorithm, we report **"T", "char#" and "CPU(s)"**, where "T" is the writing time of the E-Beam system, "char#" is the character number on final stencil, and "CPU(s)" reports the runtime. From Table 4.3 we can see E-BLOW achieves better performance than both greedy method and heuristic method in [113]. Compared with E-BLOW, the greedy method has 32% more system writing time, while [113] introduces 27% more system writing time. One possible reason is that different from the greedy/heuristic methods, E-BLOW proposes mathematical formulations to provide global view. Additionally, due to the successive rounding scheme, E-BLOW is around $23\times$ faster than the work in [113].

E-BLOW is further compared with one recent 1D-OSP solver [57] in Table 4.3. E-BLOW found stencil placements with best E-Beam system writing time for 10 out of 12 test cases. In addition, for all the MCC system cases (1M-1, . . . , 1M-8) E-BLOW outperforms [57]. One possible reason is that to optimize the overall throughput of the MCC system, a global view is necessary to balance the throughputs among different regions. E-BLOW utilizes the mathematical formulations to provide such global optimization.

Although the linear programming solvers are more expensive than the deterministic heuristics in [57], the runtime of E-BLOW is reasonable that each case can be finished in 20 seconds on average.

We further demonstrate the effectiveness of the fast ILP convergence and post-insertion. We denote **E-BLOW-0** as E-BLOW without these two techniques, and denote **E-BLOW-1** as E-BLOW with these techniques. Fig. 4.11 and Fig. 4.12 compare E-BLOW-0 and E-BLOW-1, in terms of system writing time and runtime, respectively. From Fig. 4.11 we can see that applying fast ILP convergence and post-insertion can effectively E-Beam system throughput, that is, averagely 9% system writing time reduction can be achieved. In addition, Fig. 4.12 demonstrates the performance of the fast ILP convergence. We can see that in 11 out of 12 test cases, the fast ILP convergence can effectively reduce E-BLOW CPU time. The possible reason for the slow down in case 1D-4 is that when fast convergence is called, if there are still many unsolved a_{ij} variables, ILP solver may suffer from runtime overhead problem. However, if more successive rounding iterations are applied before ILP convergence, less runtime can be reported.

For 2D-OSP, Table 4.4 gives the similar comparison. For each algorithm, we also record "T", "char #" and "CPU(s)", where the meanings are the same with that in Table 4.3. Compared with E-BLOW, although the greedy algorithm is faster, its design results would introduce 41% more system writing time. Furthermore, compared with E-BLOW, although the framework in [113] puts 2% characters onto stencil, it gets 15% more system

Table 4.4: Result Comparison for 2D-OSP

	char	CP	Greedy in [113]			[113]			E-BLOW		
	#	#	T	char #	CPU(s)	T	char #	CPU(s)	T	char #	CPU(s)
2D-1	1000	1	159654	734	2.1	107876	826	329.6	105723	789	65.5
2D-2	1000	1	269940	576	2.4	166524	741	278.1	170934	657	52.5
2D-3	1000	1	290068	551	2.6	210496	686	296.7	178777	663	56.4
2D-4	1000	1	327890	499	2.7	240971	632	301.7	179981	605	54.7
2M-1	1000	1	168279	734	2.1	122017	811	313.7	91193	777	58.6
2M-2	1000	1	283702	576	2.4	187235	728	286.1	163327	661	48.7
2M-3	1000	1	298813	551	2.6	235788	653	289.0	162648	659	52.3
2M-4	1000	1	338610	499	2.7	270384	605	285.6	195469	590	53.3
2M-5	4000	10	824060	2704	19.0	700414	2913	3891.0	687287	2853	59.0
2M-6	4000	10	1044161	2388	20.2	898530	2624	4245.0	717236	2721	60.7
2M-7	4000	10	1264748	2101	21.9	1064789	2410	3925.5	921867	2409	57.1
2M-8	4000	10	1331457	2011	22.8	1176700	2259	4550.0	1104724	2119	57.7
Avg.	-	-	550115	1218.1	8.3	448477	1324	1582.7	389930.5	1291.9	56.375
Ratio	-	-	**1.41**	0.94	**0.15**	**1.15**	1.02	**28.1**	**1.0**	1.0	**1.0**

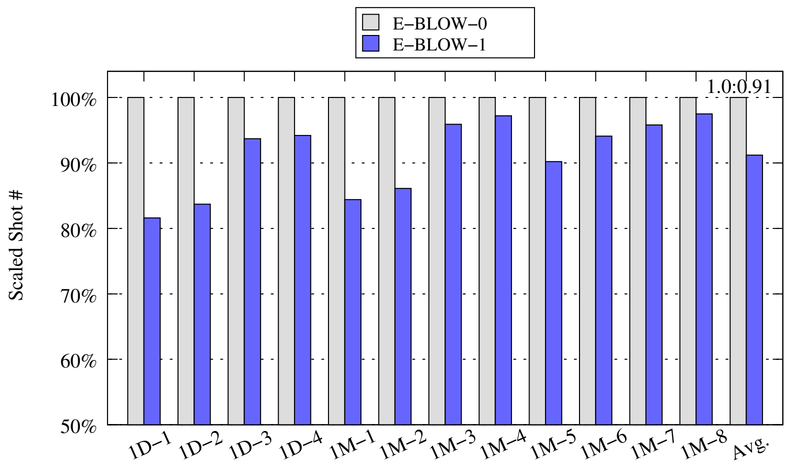


Figure 4.11: The comparison of E-Beam system writing times between E-BLOW-0 and E-BLOW-1.

writing time. The possible reason is that in E-BLOW the characters with similar writing time are clustered together. The clustering method can help to speed-up the packaging, so E-BLOW is 28× faster than [113]. In addition, after clustering the character number can be reduced. With smaller solution space, the simulated annealing engine is easier to achieve a better solution, in terms of system writing time.

From both tables we can see that compared with [113], E-BLOW can achieve a better tradeoff between runtime and system throughput.

We further compare the E-BLOW with the ILP formulations (4.2) and (4.5). Although for both OSP problems the ILP formulations can find optimal solutions theoretically, they may suffer from runtime overhead. Therefore, we randomly generate nine small benchmarks, five for 1D-OSP (“1T-x”) and four for 2D-OSP (“2T-x”). The sizes of all the character candidates are set

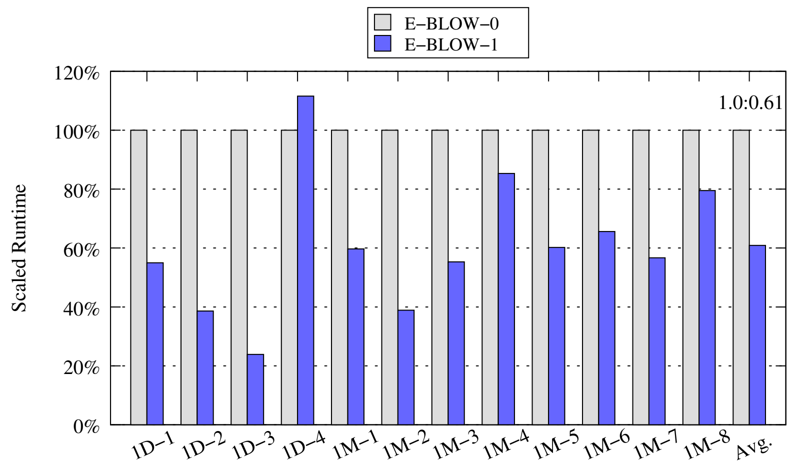


Figure 4.12: The comparison of runtime between E-BLOW-0 and E-BLOW-1.

to $40\mu m \times 40\mu m$. For 1D-OSP benchmarks, the row number is set to 1, and the row length is set to 200. The comparisons are listed in Table 4.5, where column "candidate#" is the number of character candidates. "**ILP**" and "**E-BLOW**" represent the ILP formulation and our E-BLOW framework, respectively. In ILP formulation, column "binary#" gives the binary variable number. For each mode, we report "T", "char#" and "CPU(s)", where "T" is E-Beam system writing time, "char#" is character number on final stencil, and "CPU(s)" is the runtime. Note that in Table 4.5 the ILP solutions are optimal.

Let us compare E-BLOW with ILP formulation for 1D cases (1T-1, . . . , 1T-5). E-BLOW can achieve the same results with ILP formulations, meanwhile it is very fast that all cases can be finished in 0.2 seconds. Although ILP formulation can achieve optimal results, it is very slow that a case with 14 character candidates (1T-5) can not be solved in one hour. Next, let us

Table 4.5: ILP v.s. EBLOW

	candidate#	ILP				E-BLOW		
		binary#	T	char#	CPU(s)	T	char#	CPU(s)
1T-1	8	64	434	6	0.5	434	6	0.1
1T-2	10	100	1034	6	26.1	1034	6	0.2
1T-3	11	121	1222	6	58.3	1222	6	0.2
1T-4	12	144	1862	6	1510.4	1862	6	0.2
1T-5	14	196	NA	NA	>3600	2758	6	0.1
2T-1	6	66	60	6	37.3	207	5	0.1
2T-2	8	120	354	6	40.2	653	7	0.1
2T-3	10	190	1050	6	436.8	4057	4	0.1
2T-4	12	276	NA	NA	>3600	4208	5	0.2

compare E-BLOW with ILP formulation for 2D cases (2T-1, . . . , 2T-4). For 2D cases ILP formulations are slow that if the character candidate number is 12, it cannot finish in one hour. E-BLOW is fast, but with some solution quality penalty.

Although the integral variable number for each case is not huge, we find that in the ILP formulations, the solutions of corresponding LP relations are vague. Therefore, expensive search method may cause unacceptable runtimes. From these cases ILP formulations are impossible to be directly applied in OSP problem, as in MCC system character number may be as large as 4000.

4.2.5 Summary

We have proposed E-BLOW, a tool to solve OSP problem in MCC system. For 1D-OSP, a successive relaxation algorithm and a dynamic programming based refinement are proposed. For 2D-OSP, a KD-Tree based clustering method is integrated into simulated annealing framework. Experimental results show that compared with previous works, E-BLOW can achieve better performance in terms of shot number and runtime, for both MCC system and traditional EBL system. As EBL, including MCC system, are widely used for mask making and also gaining momentum for direct wafer writing, we believe a lot more research can be done for not only stencil planning, but also EBL aware design.

4.3 L-Shape based Layout Fracturing

For EBL writing, a fundamental step is *layout fracturing*, where the layout pattern is decomposed into numerous non-overlapping rectangles. Subsequently the layout is prepared and exposed by an EBL writing machine onto the mask or the wafer, where each fractured rectangle is shot by one variable shaped beam (VSB).

As the minimum feature size further decreases, the number of rectangles in the layout is steadily increased. First, longer writing time and larger data volume are caused by highly complex optical proximity correction (OPC). Besides, the introduction of advanced lithographic techniques, e.g., DPL/MPL, add more masks in the mask manufacturing. Since the manufacturing cost is directly associated with increasing write time and data volume, the cost is also steadily increased. In addition, the low throughput has been and is still the bottleneck of EBL writing.

To overcome this manufacturing problem, several optimization methods have been proposed to reduce the EBL writing time to a reasonable level [81] [29] [113]. Among them, the L-shape shot strategy is a very simple yet effective approach to reduce the e-beam mask writing time, and thus reduce the mask manufacturing cost and improve the throughput [81] [29]. Besides, this technique can be also applied to reduce the cost of lithographic process. The conventional EBL writing is based on rectangular VSB shots. As illustrated in Fig. 4.13(a), the electrical gun generates an initial beam, which becomes uniform through the shaping aperture. Then the second aperture finalizes the target shape with a limited maximum size. As an improved technique, the printing process of the L-shape shot is illustrated in Fig. 4.13(b). One additional aperture, the third aperture, is employed to create L-shape shots. To take advantage of this new printing process, new fracturing methodology is needed to provide L-shape in the fractured layout. L-shape shot strategy has the potentiality to reduce the EBL writing time or cost by 50% if all rectangles are combined into L-shapes. For example in Fig. 4.14, instead of four rectangles, using L-shape fracturing only requires two L-shape shots.

Note that the layout fracturing problem is different from the general polygon decomposition problem in geometrical science. In order to consider yield control and CD control, the minimum width of each shot should be above a certain threshold value ϵ. A shot whose minimum width is $< \epsilon$ is called a *sliver*. In the layout fracturing, sliver minimization is an important objective [52]. As shown in Fig. 4.15, two fractured layouts can achieve the same shot number 2. However, because of sliver, the fractured result in Fig.

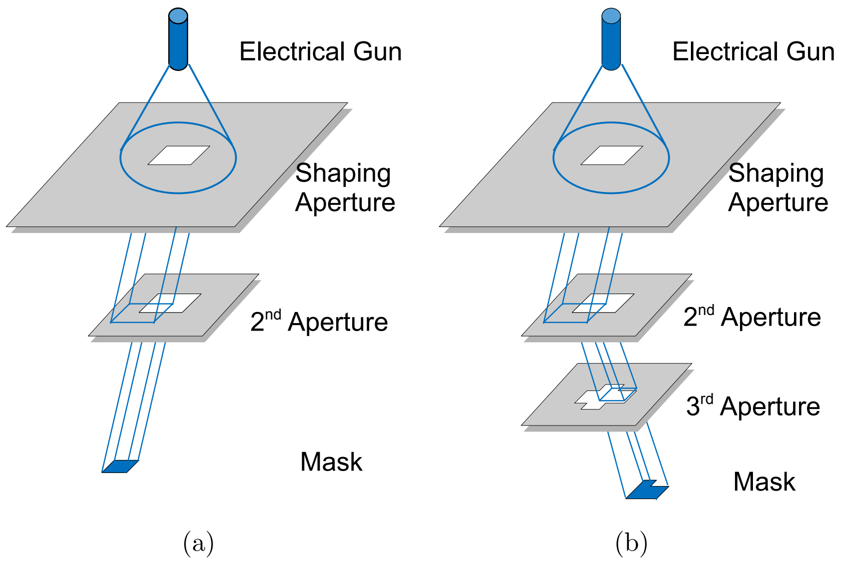


Figure 4.13: (a) Traditional rectangular EBL writing process. (b) L-shape writing process with one additional aperture.

4.15 (a) is worse than that in Fig. 4.15 (b). It shall be noted that the layout in Fig. 4.15 can be written in one L-shaped shot without any sliver.

For traditional rectangular shots, several papers have studied the layout fracturing problem [52] [53] [26] [66] [46]. Kahng et. al proposed an integer linear programming (ILP) formulation, and some matching based speed-up techniques [52] [53]. Recently, Ma et. al [66] presented a heuristic algorithm to generate rectangular shots and further reduce the sliver. Compared with the rectangular fracturing problem, the L-shape fracturing problem is new and there is only limited work, mostly describing methodology, but no systematic algorithm has been proposed so far. [81] reported the initial results that L-shape fracturing can further save about 38% of shot count, but no algorithmic

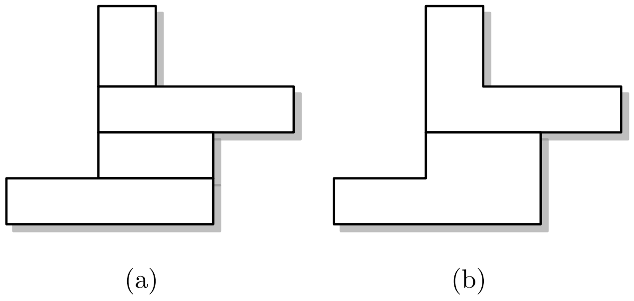


Figure 4.14: Examples of polygon fracturing. (a) Rectangular shots with 4 shot number. (b) L-shape shots with 2 shot number.

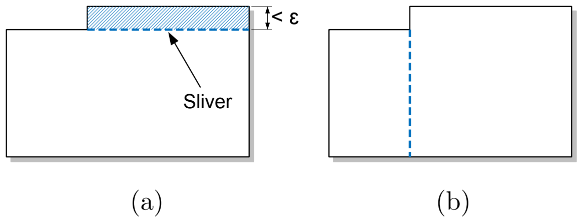


Figure 4.15: (a) Fracturing with one sliver. (b) Fracturing without sliver.

details are provided. For the general decomposition problem of polygon into L-shapes, several heuristic methods are proposed [28] [63]. However, since these heuristic methods only consider horizontal decomposition, which would result in numerous slivers, they cannot be applied to the layout fracturing problem.

This section presents the first systematic study for EBL L-shape fracturing considering the sliver minimization. We propose two algorithms for the L-shape fracturing problem. The first method, called *RM*, starts from rectangles generated by any previous fracturing framework, and merge them into L-shapes. A maximum weighted matching algorithm is proposed to find the optimal merging solution, where the shot count and the sliver can be minimized simultaneously. To further overcome the intrinsic limitations of rectangular merging, we propose another fracturing algorithm, called *DLF*. Through effectively detect and take advantage of some special cuts, DLF can directly fracture the layout into a set of L-shapes in $O(n^2 logn)$ time. The experimental results show that our algorithms are very promising for both shot count reduction and sliver minimization. In addition, DLF can even achieve significant speed-up compared with previous state-of-the-art rectangular fracturing algorithm [66].

The rest of the section is organized as follows. Section 4.3.1 presents the basics and problem formulation. Section 4.3.2 provides RM, the merging based algorithm, which will also be used as a baseline. In Section 4.3.3 we propose the DLF algorithm to directly fracture polygons into L-shapes. Section 4.3.4 presents experimental results, followed by summary in Section 4.3.5.

4.3.1 Problem Formulation

We first introduce some notations and definitions to facilitate the problem formulation. For convenience, we use the term polygon to refer to rectilinear polygons in the rest of this chapter.

Let P be an input polygon with n vertices, we define the concave vertices as follows.

Definition 8 (Concave Vertex)**.** *The concave vertex of a polygon is one at which the internal angle is* 270^o*.*

Let c be the number of concave vertices in P, [76] gave the relationship between n and c: $n = 2c + 4$. If the number of concave vertices c is odd, polygon P is called *odd polygon*; otherwise, P is called *even polygon*.

Definition 9 (Cut). *A cut of a polygon P is a horizontal or vertical line segment at least one of whose endpoints is incident on a concave vertex. The other endpoint is obtained by extending the line segment inside P until it first encounters the boundary of P.*

If both endpoints of a cut are concave vertices in the original polygon, then the cut is called a *chord*. If a cut has odd number of concave vertices to one side or another, then the cut is called an *odd-cut*. If an cut is not only odd-cut but also chord, it is called an *odd-chord*. These concepts are illustrated in Fig. 4.16, where vertices b, e, h are concave vertices, edges $\bar{bh}, \bar{ej}$ are odd-cuts, and edge $\bar{bh}$ is chord. Note that $\bar{bh}$ is an odd-chord.

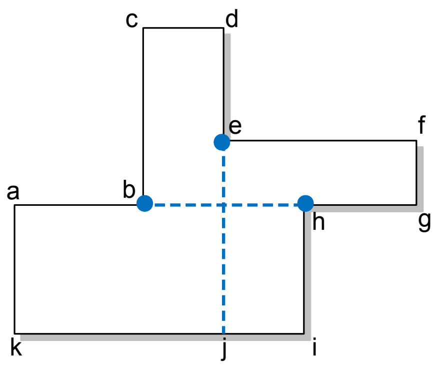


Figure 4.16: Concepts of concave vertices and cuts.

Definition 10 (L-shape). *An L-shape is a polygon shaped in the form of the letter L.*

An L-shape can be also viewed as a combination of two rectangles with a common coordinate. There are two easy ways to check whether a polygon is an L-shape. First, we can check whether the number of vertices equals to 6, i.e., $n = 6$. Besides, we can check whether there is only one concave vertex, i.e., $c = 1$.

Definition 11 (Sliver Length)**.** *For an L-shape or a rectangle, if the width of its bounding box B is above ϵ, its sliver length is 0. Otherwise, the sliver length is the length of B.*

Problem 11 (**L-shape based Layout Fracturing**)**.** *Given an input layout which is specified by polygons, our goal is to fracture it into a set of L-shapes and/or rectangles to minimize the number of shots, and meanwhile minimize the silver length of fractured shots.*

4.3.2 Rectangular Merging (RM) Algorithm

Given the rectangles generated by any rectangular fracturing methodology, we propose an algorithm, called RM, to merge them into a set of L-shapes. The main idea is that if two rectangles share a common coordinate, they can be combined into one L-shape. Although this idea is straightforward, the benefit is obvious that the previous rectangular fracturing algorithms can be re-used. Besides, the RM algorithm is used as a baseline in comparison with our another algorithm, DLF, which will be described in Section 4.3.3.

Given the input rectangles, the RM algorithm can find the optimal L-shape merging solution. Meanwhile, the shot count and sliver length can be

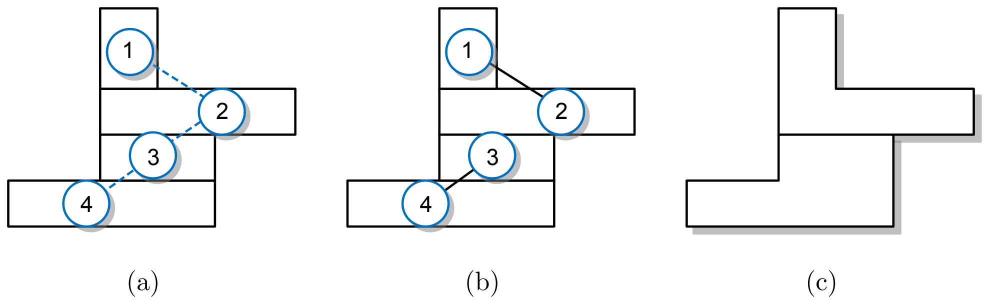


Figure 4.17: Example of RM algorithm. (a) Graph construction. (b) Maximum matching result. (c) Corresponding rectangular merging.

minimized simultaneously.

First we construct a merging graph G to represent the relationships among all the input rectangles. Each vertex in G represents a rectangle. There is an edge between two vertices if and only if those two rectangles can be merged into an L-shape. For example shown in Fig. 4.17, after rectangular fracturing, four rectangles are generated. The constructed merging graph G is illustrated in Fig. 4.17(a), where the three edges show that there are three ways to generate L-shapes. L-shape merging can be viewed as edge selection from the merging graph G. Note that one rectangle can only be assigned to one selected edge, that is, no two selected edges share a common end point. For example, rectangle 2 can only belongs to one L-shape, and thus only one edge can be chosen between edges $\overline{12}$ and $\overline{23}$.

By utilizing the merging graph, the best edge selection can be solved by finding a maximum matching. Therefore, the rectangular merging can be formulated as a maximum matching problem. In the case of Fig. 4.17, the result of the maximum matching is illustrated in Fig. 4.17(b), and the corresponding L-shape fracturing result is shown in Fig. 4.17(c).

In order to consider the sliver minimization, we assign weights to the edges to represent whether the merging would remove one sliver. For example, if there is still one sliver even two rectangles v_i and v_j are merged into one L-shape, we assign less weight to edge e_{ij}. Otherwise, larger weight is assigned. Therefore, the rectangular merging can be formulated as maximum weighted matching. Even in general graphs, the maximum weighted matching can be solved in $O(nmlogn)$ time [35], where the n is the number of vertices, and the m is the number of edges in G.

4.3.3 Direct L-Shape Fracturing (DLF) Algorithm

Although the RM algorithm described above can provide the optimal merging solution for given rectangles, it may suffer from several limitations. First, the polygon is fractured into rectangles first, and followed by a merging stage. This strategy, however, has some redundant or unnecessary operations. For the case in Fig. 4.14, instead of complex rectangles generation, only one cut is enough for the L-shape fracturing. Second, the rectangular fracturing may ignore some internal features of L-shape fracturing, which could sacrifices the whole performance. To overcome all these limitations, in this section we propose a novel algorithm, called DLF, to directly fracture polygon into L-shapes.

We observe that the solution space for the L-shape fracturing can be very large. Given a polygon, there can exist several fracturing solutions with the same shot count. For example, as shown in Fig. 4.18, the input polygon has at least five different fracturing solutions with two shots.

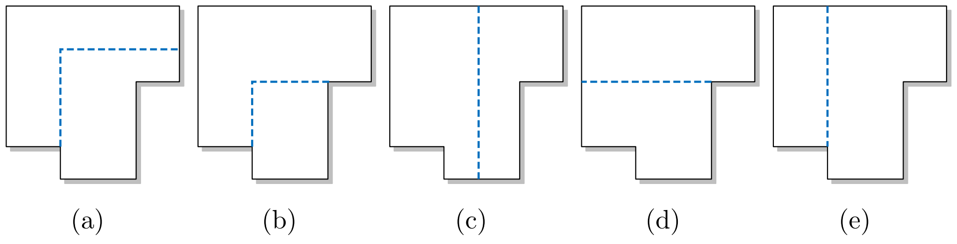


Figure 4.18: Five fracturing solutions with the same shot count.

Note that a cut has the following property: if the polygon is decomposed through a cut, the concave vertex that is one of the endpoints of the cut is no longer concave in either of the two resulting polygons. Our L-shape fracturing algorithm, DLF, takes advantage of this property. Each time a polygon is cut, DLF searches one appropriate odd-cut to decompose the polygon. It was shown in [76] that odd-cut always exists and $\lfloor c/2 \rfloor + 1$ "guards" are necessary and sufficient to cover all the interiors of a polygon with c concave vertices. Therefore, we can obtain the following lemma.

Lemma 10. *A polygon with c concave vertices can be decomposed into L-shapes with upper bound number $N_{up} = \lfloor c/2 \rfloor + 1$.*

Fig. 4.19 shows the overall flow of our DLF algorithm. We will effectively use chords and cuts to reduce the problem size while containing or even reducing the L-shape fracturing number upper bound. The first step is to detect all chords (i.e., horizontal or vertical cuts made by concave points), in

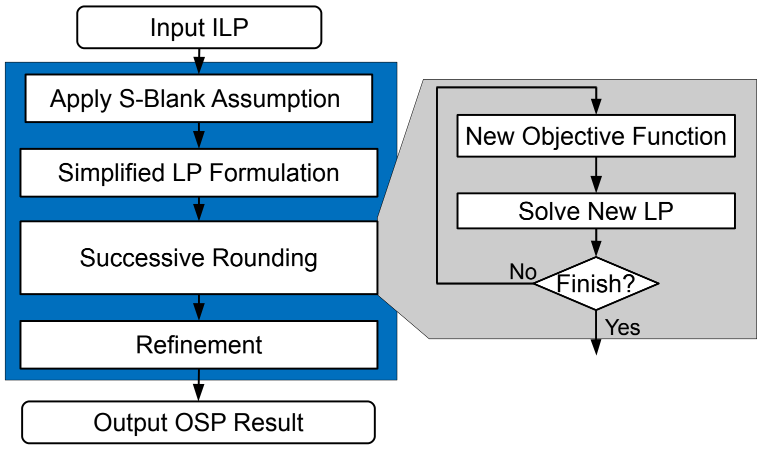


Figure 4.19: Overall flow of DLF algorithm.

particular odd-chords as they may reduce the L-shape upper bound. We will then perform sliver-aware chord selection to decompose the original polygon P into a set of sub-polygons. Then for each sub-polygon, we will perform sliver aware L-shape fracturing, where odd-cuts are detected and selected to iteratively cut the polygon into a set of L-shapes. The reason we differ chord and cut during polygon fracturing is that chord is a special cut with both end points being concave points in the original polygon. That way, we can design more efficient algorithm for odd cut/chord detection.

4.3.3.1 Sliver Aware Chord Selection

The first step of DLF algorithm is sliver aware chord selection. Cutting through chords decomposes the whole polygon P into a set of sub-polygons. By this way the problem size is reduced. We can further prove that cutting through a chord does not increase the L-shape upper bound N_{up}.

Lemma 11. *Decompose a polygon through a chord does not increase the L-shape upper bound number N_{up}.*

Proof. Cut the polygon along this chord, and let c_1 and c_2 be the number of concave vertices in the two pieces produced. Since $c = c_1 + c_2 + 2$, then using Lemma 10 we have

$$\begin{aligned}\lfloor c_1/2 \rfloor + 1 + \lfloor c_2/2 \rfloor + 1 &\leq \lfloor (c_1 + c_2)/2 \rfloor + 2 \\ &= \lfloor (c-2)/2 \rfloor + 2 = \lfloor c/2 \rfloor + 1\end{aligned}$$

□

Chord selection has been proposed in rectangular fracturing [53] [66], but for L-shape fracturing, odd-chord shall be selected as they can even reduce the number of L-shapes.

Lemma 12. *Decomposing a even polygon along an odd-chord can reduce the L-shape upper bound number N_{up} by 1.*

The proof is similar to that for Lemma 11. The only difference is that since c is even and c_1, c_2 are odd, $\lfloor c_1/2 \rfloor + 1 + \lfloor c_2/2 \rfloor + 1 < \lfloor c/2 \rfloor + 1$. Note that for an odd polygon, all chords are odd. For a even polygon, Lemma 12 provides a guideline to select chords. An example is illustrated in Fig. 4.20, which contains two chords $\bar{bh}$ and $\bar{hk}$. Since the number of concave vertices to both side of chord $\bar{bh}$ are odd (1), $\bar{bh}$ is an odd-chord. Cut along $\bar{bh}$, as shown in Fig. 4.20(a), can achieve two L-shots. However, cut along another chord

$\bar{hk}$, which is not an odd-chord, would need three shots. Note that in an odd polygon, although all chords are odd, cutting along them may not reduce N_{up}, but it will not increase N_{up} either.

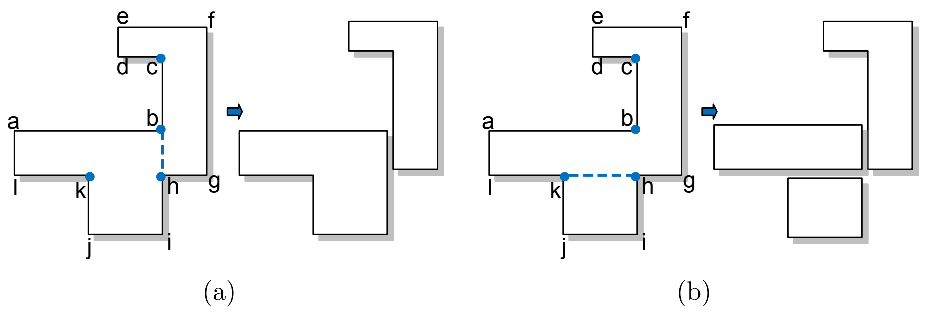


Figure 4.20: Examples to illustrate Lemma 12. (a) Cut along odd-chord $\bar{bh}$ results in two L-shape shots. (b) Cut along chord $\bar{hk}$ would cause one more shot.

For any even polygon P, we propose the odd-chord search procedure as follows. Each vertex v_i is assigned with one Boolean parity p_i. Starting from an arbitrary vertex with any parity assignment, we proceed clockwise around the polygon. If the next vertex v_j is concave, then $p_j = \neg p_i$, where p_i is the parity of current vertex v_i. Otherwise p_j is assigned to p_i. This parity assignment can be completed during one clockwise traverse in $O(n)$ time. An example of parity assignment starting from a(0) is shown in Fig. 4.21, where each vertex is associated with one parity.

Theorem 7. *In an even polygon, a chord $\bar{ab}$ is odd iff $p_a = p_b$.*

Given the parity values, the odd-chord detection can be performed using Theorem 7. For each concave vertex v_i, a plane sweep is applied to search

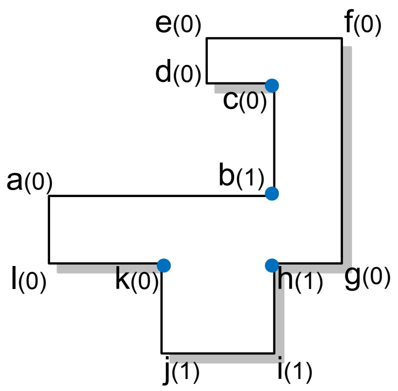


Figure 4.21: To detect odd-chords in even polygon, each vertex is associated with one Boolean parity.

any chord containing v_i. The plane sweep for each vertex can be finished in $O(logn)$, and the number of vertices is $O(n)$. Therefore, in an even polygon odd-chords detection can be completed in $O(nlogn)$ time.

After all chords are detected, chord selection is applied to choose as many chords as possible to divide the input polygon into a set of independent sub-polygons. Note that if a chord is selected to cut the polygon, it releases the concavity of its two endpoints. Therefore, if two chords intersect with each other, at most one of them could be selected. For example, in Fig. 4.21, chords $\bar{bh}$ and $\bar{hk}$ cannot be selected simultaneously. The relationship among the chords can be represented as a bipartite graph [53], and the vertices in left and right columns indicate the horizontal and vertical chords, respectively. Therefore, finding the most chords compatible with each other corresponds to finding the maximum independent set in the bipartite graph, which can be reduced to maximum matching problem, and therefore, can be done in polynomial time. It shall be noted that if the input polygon is a even polygon, because of Theorem 7, we prefer to choose odd-chords. Therefore, the bipartite graph is modified by assigning weights to the edges. In addition, sliver minimization is integrated into the chord selection. When an odd-chord candidate is detected, we calculate the distance between it and the boundary of the polygon. If the distance is less than ϵ, cutting this odd-chord would cause sliver, then we will discard this candidate.

4.3.3.2 Sliver Aware L-Shape Fracturing

After chord selection, the input polygon P is decomposed into m sub-polygons (denoted as $P_1, P_2, ..., P_m$). For each polygon P_i, we will recursively fracture it until reaching final L-shapes and/or rectangles. Our fracturing algorithm is based on odd-cut selection. The main idea is that each time we pick up one odd-cut, and decompose the polygon into two pieces through this odd-cut. Iteratively we fracture the polygon into several L-shapes. Note that our fracturing algorithm considers the sliver minimization, i.e., we try to minimize the sliver length during fracturing.

The first question is how to detect all the odd-cuts efficiently. Our method is similar to that for odd-chord detection. Each vertex v_i is assigned an order number o_i, and a Boolean parity p_i. Start at an arbitrary vertex, each vertex v_i is assigned an order o_i. We initialize the Boolean parity p to zero, and proceed clockwise around the polygon. If the next vertex v_i is normal, label its p_i as p; if v_i is concave, assign p to $\neg p$, and label its p_i with the new p value. For each concave vertex v_a, we search cuts from two directions (horizontal and vertical) from it. Here we denote $(a, \bar{bc})$ as the cut with one endpoint at vertex v_a and the other endpoint at edge $\bar{bc}$. For each cut $(a, \bar{bc})$ detected, whether it is an odd-cut can be checked in constant time using the following Theorem 8.

Theorem 8. *In an odd polygon, a cut $(a, \bar{bc})$ is an odd-cut if and only if the following condition is satisfied:*

$$\begin{cases} p_a = p_b, & \text{if } \ o_a > o_b \\ p_a \neq p_b, & \text{if } \ o_a < o_b \end{cases}$$

Due to space limit, the detailed proof is omitted. An example of odd-cut detection is shown in Fig. 4.22. There are three concave vertices, v_b, v_f and v_i in the odd polygon. Start from each concave vertex, we have searched all six cuts. Applying Theorem 8, we find out two odd-cuts $(b, \bar{fg})$ and $(i, \bar{cd})$.

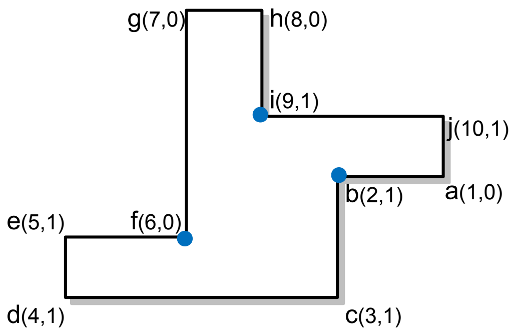


Figure 4.22: Odd-cut detection using order number and parity.

The details of our L-shape fracturing are described in Algorithm 19. Given the input polygon P, if it is already an L-shape or rectangle, then the fracturing is completed. Otherwise, we find all odd-cuts as described above

Algorithm 19 LShapeFracturing(P)

```
Require: Polygon $P$.
 1: if  $P$ is L-shape or rectangle then
 2:     Output $P$ as one of results; return
 3: end if
 4: Find all odd-cuts;
 5: Choose cut $cc$ considering the sliver minimization;
 6: if Cannot find legal odd-cut then
 7:     Generate an auxiliary cut $cc$;
 8: end if
 9: Cut $P$ through $cc$ into two polygons $P1$ and $P2$;
10: Update one vertex and four edges;
11: LShapeFracturing($P1$);
12: LShapeFracturing($P2$);
```

(line 5). From all the odd-cuts detected, we choose one cc, and cut the P into two pieces P_1 and P_2 (lines $10 - 11$). Then we recursively apply L-shape fracturing to P_1 and $P2$ (lines $12 - 13$).

Note that during the polygon decomposition, we do not need to re-calculate the order number and parity of each vertex. Instead, when a polygon is divided into two parts, we only update one vertex and four edges, while all other information can be maintained. If polygon P is cut through odd-cut $(a, \bar{bc})$, a new vertex, namely d, is generated. For the new vertex d, its order number $o_d = o_b$ and its parity $p_d = p_b$. Edge $\bar{bc}$ is replaced by edges $\bar{bd}$ and $\bar{dc}$. Besides, two edges $\bar{ad}$ and $\bar{da}$ are inserted. The update method is simple and easy to implement. An example of such update is shown in Fig. 4.23.

Sliver minimization is integrated into our L-shape fracturing algorithm. In Algorithm 19, when picking up one cut from all odd-cuts, we try to avoid

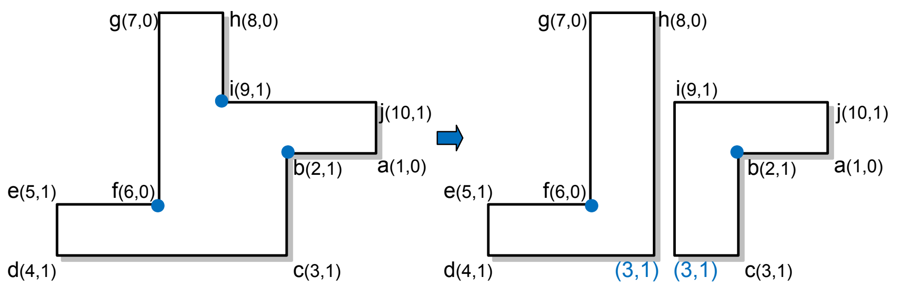


Figure 4.23: Only one vertex and four edges need to be updated during polygon decomposition.

any sliver. For example, as illustrated in Fig. 4.24(a), there are three odd-cuts, but all of them would cause sliver. Instead of selecting any of them, we generate an auxiliary cut in the middle (see Fig. 4.24(b)). Because of the auxiliary cut, the polygon can be fractured without introducing any sliver. In addition, if there are several odd-cuts not causing sliver, we pick the cut using the following rules: (1) We prefer the cut which partitions the polygon into two balanced sub-polygons; (2) If the polygon is more horizontal than vertical, we prefer a vertical cut, and vice verse.

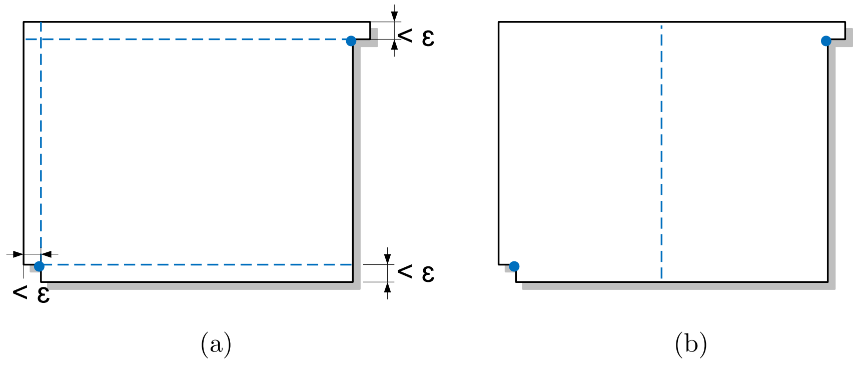


Figure 4.24: Auxiliary cut generation. (a) Here every odd-cut would cause sliver. (b) Decompose through on auxiliary cut can avoid sliver.

Given a polygon with n vertices, finding all concave vertices need $O(n)$ time. For each concave vertex v_i, searching cut starting from it needs $O(logn)$ time. Using Theorem 8, checking whether the cut is odd-cut needs $O(1)$, thus finding all odd-cuts needs $O(nlogn)$ time. Note that given a polygon with c concave vertices, if no auxiliary cut is generated, the L-shape fracturing can be completed through $\lfloor c/2 \rfloor$ odd-cuts. When auxiliary cuts are applied, there are at most $c-1$ cuts to fracture the input polygon. Therefore, we can achieve the following theorem.

Theorem 9. *The sliver aware L-shape generation can find a set of L-shapes in $O(n^2 logn)$ time.*

It shall be noted that if our objective is only to minimize the shot number, no auxiliary cut would be introduced, thus at most $\lfloor c/2 \rfloor + 1$ L-shapes are generated. In other words, the shot number would be less or equals to the theoretical upper bound N_{up}.

4.3.3.3 Speedup Technique

We observe that in practice during the execution of Algorithm 19, many odd-cuts do not intersect. In other words, many odd-cuts are compatible, and could be used to decompose the polygon at the same time. Instead of only picking one odd-cut at one time, we can achieve further speed-up by selecting multiple odd-cuts simultaneously.

If the polygon is an odd polygon, this speed-up is easily implemented. In the odd polygon, there is only one type of odd-cut: a cut that has an odd

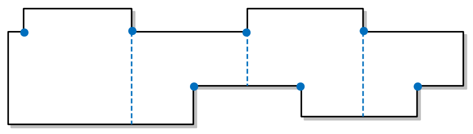


Figure 4.25: Speed-up for odd polygon, where all three odd-cuts are compatible.

number of concave vertices to each side. Partitioning the polygon along such odd-cut can leave all other odd-cuts remaining to be odd-cuts. For example, Fig. 4.25(a) shows an odd polygon, where all three odd-cuts are compatible, and can be picked up simultaneously. Through fracturing the polygon along the three odd-cuts, the L-shape fracturing problem is resolved directly.

However, this speed-up technique cannot be directly applied to an even polygon, since it may cause more shot number. The reason is that when an even polygon is cut into two pieces, some odd-cuts may no longer be odd-cuts in the sub-polygons. For example, as shown in Fig. 4.26(a), in this even-polygon all six cuts are odd-cuts and compatible. However, if we use all these compatible cuts for fracturing, we would end up with seven rectangular shots, which is obviously sub-optimal. To overcome this issue, for each even-polygon we introduce one artificial concave vertex. Through this artificial concave vertex, the polygon is translated into an odd polygon. Because of Lemma 13, this translation does not increase the total shot number. As shown in Fig. 4.26(b), in the modified odd polygon, all compatible odd-cuts can be used for fracturing without causing more shot number.

Lemma 13. *Introducing one artificial concave vertex to an even polygon does* *not increase the L-shape upper bound* N_{up}.

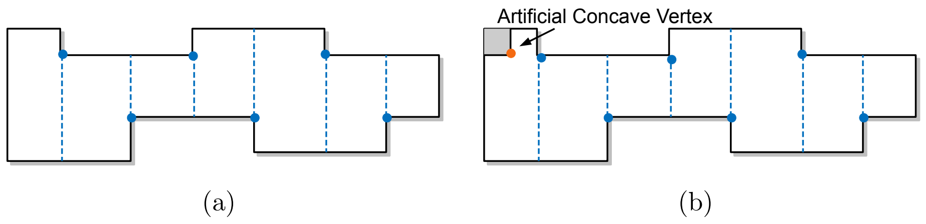


Figure 4.26: Speed-up for even polygon. (a) all cuts are odd-cuts. (b) Introducing one artificial concave vertex, translate the even polygon into an odd polygon.

Through employing this speed-up technique, for most cases, the odd-cut detection can be applied only once, therefore the DLF algorithm could be completed in $O(nlogn)$ time in practice.

4.3.4 Experimental Results

We implemented our two L-shape fracturing algorithms, RM and DLF, in C++. Since RM needs a rectangular fracturing method to generate initial rectangles, we implemented a state-of-the-art algorithm proposed in [66]. Based on the generated rectangles, RM algorithm is applied to merge them into a set of L-shapes. LEDA package [73] is adopted for the maximum weighted matching algorithm.

The experiments are performed on an Intel Xeon 3.0GHz Linux machine with 32G RAM. ISCAS 85&89 benchmarks are scaled down to 28nm logic node, followed by accurate lithographic simulations performed to the Metal 1 layers. All involved lithography simulations in the *Calibration Phase* are applied under industry-strength RET (OPC). For all the resulting post-OPC layers, OpenAccess 2.2 [4] is adopted for interfacing.

Table 4.6 shows the results of our DLF algorithm in comparison with the approaches in [66] and the RM algorithm. Since the framework [66] is adopted to provide the input rectangles, the RM algorithm is denoted as " [66]+RM". Column "poly#" lists the number of polygons of each test circuit. All fracturing methods are evaluated with the sliver parameter $\epsilon = 5nm$. For each method, columns "shots", "sliver", and "CPU" denote the shot number, total sliver length, and runtime, respectively. First we compare the fracturing algorithm in [66] and the RM algorithm. From the table we can see that as an incremental algorithm, the RM algorithm can further reduce the shot number by 37%, and the sliver length by 45%. Meanwhile, the runtime increasing is reasonable: RM algorithm introduces 41% more runtime. Besides, we compare our DLF algorithm with other two methods. We can see that DLF demonstrates the best performance, in terms of both runtime and performance. Compared with traditional sliver aware rectangular fracturing [66], it can achieve around 9× speed-up. Besides, the shot number and the sliver length can be significantly reduced (39% and 82%, respectively). Even compared with RM algorithm, DLF is better in terms of performance: it can further reduce the shot number and the sliver length by 3.2% and 67%, respectively.

In order to evaluate the scalability of our algorithm, we summarize all the run time from Table 4.6, and display in Fig. 4.27. Here the X axis denotes the number of polygons (e.g., the problem size), and the Y axis shows the

Table 4.6: Runtime and Performance Comparisons

Circuts	poly#	[66]			[66]+RM			DLF		
		shots	sliver (μm)	CPU(s)	shots	sliver (μm)	CPU(s)	shots	sliver (μm)	CPU(s)
C432	1109	6898	48.3	8.51	4371	23.5	10.0	4214	7.4	1.87
C499	2216	13397	96.0	16.9	8325	45.0	19.5	8112	11.8	2.6
C880	2411	17586	160.5	24.93	11020	84.4	29.7	10653	28.6	3.8
C1355	3262	23283	185.2	29.44	14555	87.8	33.6	13936	24.8	5.1
C1908	5125	35657	333.6	48.78	22352	181.1	57.3	21540	88.0	7.68
C2670	7933	56619	525.4	84.11	35424	274.4	96.9	34102	114.8	11.89
C3540	10189	74632	668.5	114.33	46617	360.0	133.7	44901	129.8	15.98
C5315	14603	108761	950.4	176.89	67795	488.9	200.8	65222	190.2	23.85
C6288	14575	103148	819.2	175.65	64987	382.0	201.0	62416	86.1	22.64
C7552	21253	151643	1334.6	242.77	94902	717.6	280.0	91157	290.7	32.02
S1488	4611	37126	303.7	55.03	22984	146.2	64.7	22099	31.6	8.14
S38417	67696	454307	4040.2	727.1	285049	2293.0	1020	275054	729	88.5
S35932	26267	163956	1470.4	228.02	103960	808.3	256.4	100629	284	34.85
S38584	168319	1096363	10045.2	2268.6	690054	5777.0	3565.2	666906	1801.7	216.39
S15850	34660	231681	2012.8	329.99	145745	1085.1	414.6	140879	320	44.7
avg.	-	171670	1533.0	302.1	107876	850.3	425.6	104121	275.9	34.7
ratio	-	1	1	1	0.63	0.55	1.41	**0.61**	**0.18**	**0.11**

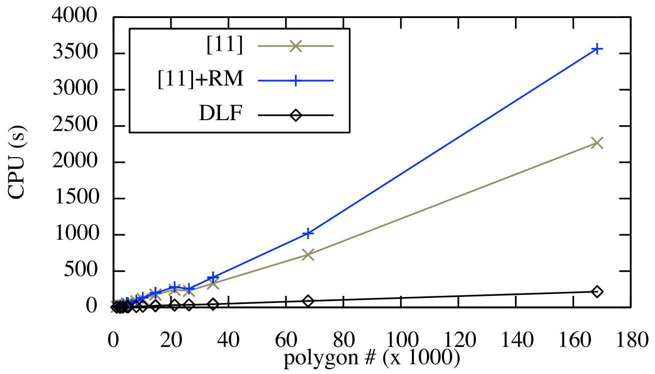


Figure 4.27: Comparison on algorithm scalability.

runtime. We can see that DLF algorithm scales better than both [66] and RM algorithm.

4.3.5 Summary

In this section we have proposed two novel algorithms for EBL with the new L-shape based layout fracturing for shot number and sliver minimization. The rectangular merging (RM) based algorithm is optimal for a given set of rectangular fractures. However, to get better performance, we show that the direct L-shape fracturing (DLF) algorithm is superior by directly decomposing the original layouts into a set of L-shapes. DLF obtained the best results in all metrics, including shot count, sliver, as well as runtime compared to the previous state-of-the-art rectangular fracturing with RM. To our best knowledge, this is the first systematic and algorithmic effort in EBL L-shaped fracturing with sliver minimization. As EBL is widely used for mask making and also gaining momentum for direct wafer writing, we believe a lot more research can be done, for not only layout fracturing but also EBL-aware physical design.

Chapter 5

Conclusions and Future Works

In this dissertation, we have proposed a set of algorithms/methodologies to resolve issues in modern design for manufacturing problems with advanced lithography. Our major contributions include:

- In Chapter 2, we tackled the challenge of layout decompositions for different patterning techniques. In Section 2.2 we have proven that triple patterning layout decomposition is NP-hard. Besides, we have proposed a number of optimization techniques to solve the layout decomposition problem: (a) integer linear programming (ILP) formulation to search optimal solution; (b) effective graph based simplification techniques to reduce the problem size; (c) novel semidefinite programming (SDP) based algorithms to achieve further balance in terms of runtime and solution quality; To further reduce the variations in decomposed results, in Section 2.3 we proposed a high performance layout decomposer providing more balanced density. In Section 2.4 we proposed a comprehensive study for LELE-EC layout decomposition. End-cut candidates are generated considering potential hotspots, and the core layout decomposition is formulated as an integer linear programming (ILP). In Section 2.5 we extend to general multiple patterning problems. The proposed decomposer consists of holistic algorithmic processes, such as semidefinite programming based algorithm, linear color assignment, and novel GH-tree based graph division. Experimental evaluations have demonstrated that our decomposer is effective and efficient to obtain high quality solution.

- In Chapter 3, we presented a coherent framework, including standard cell compliance and detailed placement, to enable TPL friendly design. Considering TPL constraints during early design stages, such as standard cell compliance, improves the layout decomposability. With the precoloring solutions of standard cells, we have presented a TPL aware detailed placement where the layout decomposition and placement can be resolved simultaneously. In addition, we proposed a linear dynamic programming to solve TPL aware detailed placement with maximum displacement, which can achieve good trade-off in terms of runtime and performance.

- In Chapter 4, we study the design for manufacturing with E-Beam lithography. In Section 4.2 we presented E-BLOW, the first study for OSP problem in MCC system. We have proven that both 1D-OSP and 2D-OSP problems are NP-hard. We formulated integer linear programming (ILP) to co-optimizing characters selection and physical placements on stencil. To handle 1D-OSP problem, we proposed a set of algorithms, including simplified formulation, successive relaxation, and post refinement. To handle 2D-OSP problem, we designed a KD-Tree based clustering algorithm to achieve speed-up. In Section 4.3 we have proposed two algorithms for the L-shape fracturing problem. The first method, called *RM*, starts from rectangles generated by any previous fracturing framework, and merge them into L-shapes. The second fracturing algorithm, called *DLF* can effectively detect and take advantage of some special cuts. Therefore, DLF can directly fracture the layout into a set of L-shapes in $O(n^2 logn)$ time.

With the above explorations and discussions, we have demonstrated the unique role of design for manufacturing (DFM) in the process of the advanced lithography techniques. With many challenges in this emerging field, we expect to see more future works along this direction as the advanced lithography technique will still have a lot of room to improve and continue the Moore's law benefit. For example, the following are some future research directions and open problems:

- We have handled layout decomposition for different patterning techniques. However, there is still some room to improve the performance. For instance, for triple patterning with end-cutting, only expensive ILP based method is proposed. Such method may suffer from long runtime penalty, and may be not able to scale to whole chip level decomposition. Therefore, how to solve this layout decomposition problem with better runtime and solution quality balance is an open question.

- We have integrated triple patterning constraints into early design stages. It would be interesting to consider how to handle other lithography rules or constraints in early design stages.
- Next generation lithography techniques such as directed self-assembly (DSA), extrme ultra violet (EUV), and nanoimprint lithography (NIL) can be further studied and evaluated as options for future manufacturing.

Bibliography

[1] Cadence SOC Encounter. `http://www.cadence.com`.

[2] Mentor Calibre. `http://www.mentor.com`.

[3] NanGate FreePDK45 Generic Open Cell Library. `http://www.si2.org/openeda.si2.org/projects/nangatelib`.

[4] [Online]. Available: . `www.si2.org/?page=69`.

[5] Predictive Technology Model ver. 2.1. `http://ptm.asu.edu`.

[6] Synopsys Design Compiler. `http://www.synopsys.com`.

[7] Saurabh N Adya and Igor L Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 11(6):1120–1135, 2003.

[8] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: a survey. *Integr. VLSI J.*, 19:1–81, August 1995.

[9] VAN OOSTEN Anton, NIKOLSKY Peter, HUCKABAY Judy, GOOSSENS Ronald, and NABER Robert. Pattern split rules! a feasibility study of rule based pitch decomposition for double patterning. In *Proc. of SPIE*, volume 6730, 2007.

[10] Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. part i: Discharging. *Illinois Journal of Mathematics*, 21(3):429–490, 1977.

[11] Yukiyasu Arisawa, Hajime Aoyama, Taiga Uno, and Toshihiko Tanaka. EUV flare correction for the half-pitch 22nm node. In *Proc. of SPIE*, volume 7636, 2010.

[12] C.P. Ausschnitt and P. Dasari. Multi-patterning overlay control. In *Proc. of SPIE*, volume 6924, 2008.

[13] Shayak Banerjee, Zhuo Li, and Sani R Nassif. ICCAD-2013 CAD contest in mask optimization and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 271–274, 2013.

[14] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[15] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11:613 – 623, 1999.

[16] Y. Borodovsky. Lithography 2009 overview of opportunities. In *Semicon West*. 2009.

[17] Ulrich Brenner and Jens Vygen. Faster optimal single-row placement with fixed ordering. In *IEEE/ACM Proc. Design, Automation and Test in Eurpoe (DATE)*, pages 117–121, 2000.

[18] Li-Wen Xinyu Bao Bencher Chang and H-S Chris Philip Wong. Experimental demonstration of aperiodic patterns of directed self-assembly by block copolymer lithography for random logic circuit layout. In *IEEE International Electron Devices Meeting*, 2010.

[19] Pinhong Chen and Ernest S. Kuh. Floorplan sizing by linear programming approximation. In *IEEE/ACM Design Automation Conference (DAC)*, 2000.

[20] Tung-Chieh Chen, Minsik Cho, David Z Pan, and Yao-Wen Chang. Metal-density-driven placement for CMP variation and routability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(12):2145–2155, 2008.

[21] Zihao Chen, Hailong Yao, and Yici Cai. SUALD: Spacing uniformity-aware layout decomposition in triple patterning lithography. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 566–571, 2013.

[22] Chris Chu and Wai-Kei Mak. Flexible packed stencil design with multiple shaping apertures for e-beam lithography. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 137–142, 2014.

[23] Christopher Cork, Jean-Christophe Madre, and Levi Barnes. Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns. In *Proc. of SPIE*, volume 7028, 2008.

[24] Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, MA, USA, 1990.

[25] M. Dawande, J. Kalagnanam, P. Keskinocak, F.S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, 4:171–186, 2000.

[26] Brian Dillon and Tim Norris. Case study: The impact of vsb fracturing. In *Proc. of SPIE*, volume 7028, 2008.

[27] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Dokl*, volume 11, pages 1277–1280, 1970.

[28] H. Edelsbrunner, J. O'Rourke, and E. Welzl. Stationing guards in rectilinear art galleries. *Comput. Vision, Graphics, Image Process*, 28:167–176, 1984.

[29] A. Elayat, T. Lin, E. Sahouria, and S. F. Schulze. Assessment and comparison of different approaches for mask write time reduction. In *Proc. of SPIE*, volume 8166, 2011.

[30] S.-Y. Fang, S.-Y. Chen, and Y.-W. Chang. Native-conflict and stitch-aware wire perturbation for double patterning technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 31(5):703–716, may 2012.

[31] Shao-Yun Fang, Yao-Wen Chang, and Wei-Yu Chen. A novel layout decomposition algorithm for triple patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(3):397–408, March 2014.

[32] Shao-Yun Fang, Wei-Yu Chen, and Yao-Wen Chang. A novel layout decomposition algorithm for triple patterning lithography. In *IEEE/ACM Design Automation Conference (DAC)*, pages 1185–1190, 2012.

[33] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *IEEE/ACM Design Automation Conference (DAC)*, pages 175–181, 1982.

[34] Aki Fujimura. Design for E-Beam: design insights for direct-write maskless lithography. In *Proc. of SPIE*, volume 7823, 2010.

[35] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, March 1986.

[36] Jhih-Rong Gao, Bei Yu, Ru Huang, and David Z. Pan. Self-aligned double patterning friendly configuration for standard cell library considering placement. In *Proc. of SPIE*, volume 8684, 2013.

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

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

[39] Rani S Ghaida, Kanak B Agarwal, Sani R Nassif, Xin Yuan, Lars W Liebmann, and Puneet Gupta. Layout decomposition and legalization for double-patterning technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 32(2):202–215, 2013.

[40] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):551–570, 1961.

[41] Satoshi Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Trans. on Circuits and Systems*, 28(1):12–18, 1981.

[42] Mohit Gupta, Kwangok Jeong, and Andrew B. Kahng. Timing yield-aware color reassignment and detailed placement perturbation for bimodal cd distribution in double patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29(8):1229–1242, August 2010.

[43] Gurobi Optimization Inc. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2014.

[44] Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

[45] Shiyan Hu, Patrik Shah, and Jiang Hu. Pattern sensitive placement perturbation for manufacturability. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 18(6):1002–1006, 2010.

[46] Shangliang Jiang, Xu Ma, and Avideh Zakhor. A recursive cost-based approach to fracturing. In *Proc. of SPIE*, volume 7973, 2011.

[47] Ellis L Johnson, George L Nemhauser, and Martin WP Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

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

[49] Andrew B. Kahng, Chul-Hong Park, Xu Xu, and Hailong Yao. Layout decomposition approaches for double patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29:939–952, June 2010.

[50] Andrew B Kahng, Sherief Reda, and Qinke Wang. Architecture and details of a high quality, large-scale analytical placer. In *IEEE/ACM*

International Conference on Computer-Aided Design (ICCAD), pages 891–898, 2005.

[51] Andrew B Kahng, Paul Tucker, and Alex Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In *IEEE/ACM Asia and South Pacific Design Automation Conference (AS-PDAC)*, pages 241–244, 1999.

[52] Andrew B. Kahng, Xu Xu, and Alex Zelikovsky. Yield-and cost-driven fracturing for variable shaped-beam mask writing. In *Proc. of SPIE*, volume 5567, 2004.

[53] Andrew B. Kahng, Xu Xu, and Alex Zelikovsky. Fast yield-driven fracture for variable shaped-beam mask writing. In *Proc. of SPIE*, volume 6283, 2006.

[54] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45:246–265, March 1998.

[55] Michael Kaufmann and Dorothea Wagner. *Drawing graphs: methods and models*, volume 2025. Springer, 2001.

[56] Jian Kuang and Evangeline F.Y. Young. An efficient layout decomposition approach for triple patterning lithography. In *IEEE/ACM Design Automation Conference (DAC)*, pages 69:1–69:6, 2013.

[57] Jian Kuang and Evangeline F.Y. Young. A highly-efficient row-structure stencil planning approach for E-Beam lithography with overlapped characters. In *ACM International Symposium on Physical Design (ISPD)*, 2014.

[58] Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

[59] Lars Liebmann, David Pietromonaco, and Matthew Graf. Decomposition-aware standard cell design flows to enable double-patterning technology. In *Proc. of SPIE*, volume 7974, 2011.

[60] Burn J Lin. Successors of ArF water-immersion lithography: EUV lithography, multi-E-Beam maskless lithography, or nanoimprint? *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 7(4):040101–040101, 2008.

[61] Burn J. Lin. Lithography till the end of Moore's law. In *ACM International Symposium on Physical Design (ISPD)*, pages 1–2, 2012.

[62] Yen-Hung Lin, Bei Yu, David Z. Pan, and Yih-Lang Li. TRIAD: A triple patterning lithography aware detailed router. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 123–129, 2012.

[63] Mario A. Lopez and Dinesh P. Mehta. Efficient decomposition of polygons into L-shapes with application to VLSI layouts. *ACM Transactions*

on Design Automation of Electronic Systems (TODAES), 1(3):371–395, July 1996.

[64] Kevin Lucas, Chris Cork, Bei Yu, Gerry Luk-Pat, Ben Painter, and David Z. Pan. Implications of triple patterning for 14 nm node design and patterning. In *Proc. of SPIE*, volume 8327, 2012.

[65] Qiang Ma, Hongbo Zhang, and Martin D. F. Wong. Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology. In *IEEE/ACM Design Automation Conference (DAC)*, pages 591–596, 2012.

[66] Xu Ma, Shangliang Jiang, and Avideh Zakhor. A cost-driven fracture heuristics to minimize sliver length. In *Proc. of SPIE*, volume 7973, 2011.

[67] Chris Mack. *Fundamental principles of optical lithography: the science of microfabrication*. John Wiley & Sons, 2008.

[68] Wai-Kei Mak and Chris Chu. E-Beam lithography character and stencil co-optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33, 2014.

[69] S. Manakli, H. Komami, M. Takizawa, T.Mitsuhashi, and L. Pain. Cell projection use in mask-less lithography for 45nm & 32nm logic nodes. In *Proc. of SPIE*, volume 7271, 2009.

[70] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[71] T. Maruyama, M. Takakuwa, Y. Kojima, Y. Takahashi, K. Yamada, J. Kon, M. Miyajima, A. Shimizu, Y. Machida, H. Hoshino, H. Takita, S. Sugatani, and H. Tsuchikawa. EBDW technology for EB shuttle at 65nm node and beyond. In *Proc. of SPIE*, volume 6921, 2008.

[72] Takashi Maruyama, Yasuhide Machida, Shinji Sugatani, Hiroshi Takita, Hiromi Hoshino, Toshio Hino, Masaru Ito, Akio Yamada, Tetsuya Iizuka, Satoshi Komatsue, Makoto Ikeda, and Kunihiro Asada. CP element based design for 14nm node EBDW high volume manufacturing. In *Proc. of SPIE*, volume 8323, 2012.

[73] Kurt Mehlhorn and Stefan Naher. *LEDA: A platform for combinatorial and geometric computing.* Cambridge University Press, 1999.

[74] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 12:1518–1524, 1996.

[75] Richard Neapolitan and Kumarss Naimipour. *Foundations of algorithms.* Jones & Bartlett Publishers, 2010.

[76] Joseph O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21:118–130, 1983.

[77] L. Pain, M. Jurdit, J. Todeschini, S. Manakli, B. Icard, B. Minghetti, G. Bervin, A. Beverina, F. Leverd, M. Broekaart, P. Gouraud, V. De Jonghe, Ph. Brun, S. Denorme, F. Boeuf, V. Wang, and D. Henry. Electron beam direct write lithography flexibility for asic manufacturing an opportunity for cost reduction. In *Proc. of SPIE*, volume 5751, 2005.

[78] David Z Pan, Bei Yu, and J-R Gao. Design for manufacturing with emerging nanolithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 32(10):1453–1472, 2013.

[79] Hans C. Pfeiffer. New prospects for electron beams as tools for semiconductor lithography. In *Proc. of SPIE*, volume 7378, 2009.

[80] Neil Robertson, Daniel P Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *ACM Symposium on Theory of computing*, pages 571–575. ACM, 1996.

[81] Emile Sahouria and Amanda Bowhill. Generalization of shot definition for variable shaped e-beam machines for write time reduction. In *Proc. of SPIE*, volume 7823, 2010.

[82] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. Comput.*, 38:62–81, January 1989.

[83] Masahiro Shoji, Tadao Inoue, and Masaki Yamabe. Extraction and utilization of the repeating patterns for CP writing in mask making. In *Proc. of SPIE*, volume 7748, 2010.

[84] Makoto Sugihara, Taiga Takata, Kenta Nakamura, Ryoichi Inanami, Hiroaki Hayashi, Katsumi Kishimoto, Tetsuya Hasebe, Yukihiro Kawano, Yusuke Matsunaga, Kazuaki Murakami, and Katsuya Okumura. Cell library development methodology for throughput enhancement of character projection equipment. *IEICE Transactions on Electronics*, E89-C:377–383, 2006.

[85] Jian Sun, Yinghai Lu, Hai Zhou, and Xuan Zeng. Post-routing layer assignment for double patterning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 793–798, 2011.

[86] S. Sutanthavibul, E. Shragowitz, and J.B. Rosen. An analytical approach to floorplan design and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 10(6):761 –769, jun 1991.

[87] Taraneh Taghavi, Charles Alpert, Andrew Huber, Zhuo Li, Gi-Joon Nam, and Shyam Ramji. New placement prediction and mitigation techniques for local routing congestion. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 621–624, 2010.

[88] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic

graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):61–79, 1988.

[89] Xiaoping Tang and Minsik Cho. Optimal layout decomposition for double patterning technology. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 9–13, 2011.

[90] Robert Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2:160–161, 1974.

[91] Haitong Tian, Yuelin Du, Hongbo Zhang, Zigang Xiao, and M.D.F. Wong. Constrained pattern assignment for standard cell based triple patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 57–64, 2013.

[92] Haitong Tian, Hongbo Zhang, Qiang Ma, Zigang 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)*, pages 57–64, 2012.

[93] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Rev.*, 38:49–95, March 1996.

[94] Jens Vygen. Algorithms for detailed placement of standard cells. In *IEEE/ACM Proc. Design, Automation and Test in Eurpoe (DATE)*, pages 321–324, 1998.

[95] Christian Wagner and Noreen Harned. EUV lithography: Lithography gets extreme. *Nature Photonics*, 4(1):24–26, 2010.

[96] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[97] Alfred Kwok-Kit Wong. *Resolution enhancement techniques in optical lithography*, volume 47. SPIE press, 2001.

[98] Yue Xu and Chris Chu. GREMA: graph reduction based efficient mask assignment for double patterning technology. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 601–606, 2009.

[99] Yue Xu and Chris Chu. A matching based decomposer for double patterning lithography. In *ACM International Symposium on Physical Design (ISPD)*, pages 121–126, 2010.

[100] Jae-Seok Yang, Katrina Lu, Minsik Cho, Kun Yuan, and David Z. Pan. A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 637–644, 2010.

[101] Hiroshi Yasuda, Takeshi Haraguchi, and Akio Yamada. A proposal for an MCC (multi-column cell with lotus root lens) system to be used as a mask-making e-beam tool. In *Proc. of SPIE*, volume 5567, 2004.

[102] Bei Yu, Jhih-Rong Gao, Duo Ding, Yongchan Ban, Jae-Seok Yang, Kun Yuan, Minsik Cho, and David Z Pan. Dealing with IC manufacturability in extreme scaling. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 240–242, 2012.

[103] Bei Yu, Jhih-Rong Gao, and David Z. Pan. L-Shape based layout fracturing for E-Beam lithography. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 249–254, 2013.

[104] Bei Yu, Jhih-Rong Gao, and David Z. Pan. Triple patterning lithography (TPL) layout decomposition using end-cutting. In *Proc. of SPIE*, volume 8684, 2013.

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

[106] Bei Yu and David Z. Pan. Layout decomposition for quadruple patterning lithography and beyond. In *IEEE/ACM Design Automation Conference (DAC)*, pages 1–6, 2014.

[107] Bei Yu, Xiaoqing Xu, Jhih-Rong Gao, and David Z Pan. Methodology for standard cell compliance and detailed placement for triple patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 349–356, 2013.

[108] Bei Yu, Kun Yuan, Jhih-Rong Gao, and David Z. Pan. E-BLOW: E-Beam lithography overlapping aware stencil planning for MCC system. In *IEEE/ACM Design Automation Conference (DAC)*, pages 70:1–70:7, 2013.

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

[110] Kun Yuan and David Z. Pan. WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 32–38, 2010.

[111] Kun Yuan, Jae-Seok Yang, and David Z. Pan. Double patterning layout decomposition for simultaneous conflict and stitch minimization. In *ACM International Symposium on Physical Design (ISPD)*, pages 107–114, 2009.

[112] Kun Yuan, Jae-Seok Yang, and David Z. Pan. Double patterning layout decomposition for simultaneous conflict and stitch minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29(2):185–196, feb. 2010.

[113] Kun Yuan, Bei Yu, and David Z. Pan. E-Beam lithography stencil planning and optimization with overlapped characters. *IEEE Transactions*

on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 31(2):167–179, Feb. 2012.

[114] Hongbo Zhang, Yuelin Du, M. D. Wong, and Rasit Topaloglu. Self-aligned double patterning decomposition for overlay minimization and hot spot detection. In *IEEE/ACM Design Automation Conference (DAC)*, pages 71–76, 2011.

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

Vita

Bei Yu received a Bachelor's degree in Information and Compute Science from the University of Electronic Science and Technology of China in 2007 and a Master's degree in Computer Science & Technology from Tsinghua University, China, in 2010. He started his Ph.D. program at the University of Texas at Austin in 2010, with research advisor David Z. Pan. He has interned at Mentor Graphics in 2011 summer and Oracle in 2013 summer.

Bei Yu's research interests include design for manufacturability and optimization algorithms with applications in CAD. The impacts of his research were recognized with Chinese Government Award for Outstanding Self-Financed Students Abroad in 2013, Silver Medal in ACM Student Research Contest in 2013, SPIE Education Scholarship in 2013, IBM Ph.D. Scholarship in 2012, ICCAD'13 Best Paper Award, ASPDAC'12 Best Paper Award, three other Best Paper Award Nominations (DAC'14, ASPDAC'13, ICCAD'11), and two Awards in ICCAD CAD Contests in 2012 and 2013, respectively.

Permanent address: disyulei@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.