Bei Yu · David Z. Pan

Design for Manufacturability with Advanced Lithography

Springer

Design for Manufacturability with Advanced Lithography

Bei Yu • David Z. Pan

Design for Manufacturability with Advanced Lithography

Springer

Bei Yu
ECE Department
The University of Texas
Austin, TX, USA

David Z. Pan
ECE Department
The University of Texas
Austin, TX, USA

ISBN 978-3-319-20384-3 ISBN 978-3-319-20385-0 (eBook)
DOI 10.1007/978-3-319-20385-0

Library of Congress Control Number: 2015949238

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

To Le and Mandy
Bei

To Shaoling and Jeffrey
David

Preface

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-28 nm technology nodes. To bridge this gap, design for manufacturability (DFM) is a must to co-optimize both design and lithography process at the same time.

In this book, we have aimed to present the most state-of-the-art research results on DFM with multiple patterning lithography (MPL) and electron beam lithography (EBL). Note that we have made no attempt to include everything, or even everything which is important in DFM. For example, design challenges toward extreme ultraviolet (EUV), directed self-assembly (DSA), and nanoimprint lithography (NIL) are not covered in this book. We hope this book will function as a concise introduction and demonstration on the design and technology co-optimization.

DFM for advanced lithography could be defined very differently under different circumstances. In general, progress in advanced lithography happens along three different directions:

- New patterning technique (e.g., layout decomposition for different patterning techniques)
- New design methodology (e.g., lithography aware standard cell design and physical design)
- New illumination system (e.g., layout fracturing for EBL system, stencil planning for EBL system)

For the research direction of new patterning technique, we study the layout decomposition problems for different patterning technique and explore four important topics. We present the proof that triple patterning layout decomposition is NP-hard. We propose a number of CAD optimization and integration techniques to solve different decomposition problems. For the research direction of new design methodology, we will show the limitation of traditional design flow. That is, ignoring advanced lithography constraints in early design stages may limit the

potential to resolve all the lithography process conflicts. To overcome the limitation, we propose a coherent framework, including standard cell compliance and detailed placement, to enable lithography friendly design. For the EBL illumination system, we focus on two topics to improve the throughput of the whole EBL system. 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 submicron domain.

We are particularly grateful to Dr. Bei Yu's PhD dissertation committee members, as the major material of this book is based on his dissertation. In particular, we would like to thank Prof. Ross Baldick for his technical suggestions and comments on the optimization formulations. We would like to thank Prof. Ray Chen for his comments on future applications. We would like to thank Prof. Michael Orshansky for his technical suggestions during the development of this book. We would like to thank Prof. Nur A. Touba for his kindness and support. We would like to thank Dr. Kevin Lucas for the great insights and helpful comments to patterning techniques during the years.

We are grateful to our colleagues, Dr. Charles Alpert (Cadence), Prof. Yao-Wen Chang (National Taiwan University), Dr. Salim Chowdhury (Oracle), Prof. Chris Chu (ISU), Dr. Brian Cline (ARM), Dr. Gilda Garreton (Oracle), Prof. Shiyan Hu (MTU), Prof. Ru Huang (Peking University), Dr. Zhuo Li (Cadence), Dr. Lars Liebmann (IBM), Dr. Gerard Luk-Pat (Synopsys), Dr. Rajendran Panda (Oracle), Dr. Jing Su (ASML), Prof. Martin Wong (UIUC), Dr. Greg Yeric (ARM), Prof. Xuan Zeng (Fudan University), and Dr. Yi Zou (ASML), for their valuable help, suggestions, and discussions on early draft of this book.

We would like to express our gratitude to the colleagues and alumni of the UTDA group at the University of Texas who gave us detailed expert feedback (e.g., Yongchan Ban, Ashutosh Chakraborty, Minsik Cho, Duo Ding, Jhih-Rong Gao, Derong Liu, Yen-Hung Lin, Yibo Lin, Che-Lun Hsu, Tetsuaki Matsunawa, Jiaojiao Ou, Jiwoo Pak, Subhendu Roy, Biying Xu, Xiaoqing Xu, Jae-Seok Yang, Wei Ye, Kun Yuan, Boyang Zhang, Yilin Zhang). Only through those inspiring discussions and productive collaborations that this book could be developed and polished.

We thank Xuming Zeng and Aaron Zou for editing and proofreading many chapters of the book. We also thank the EDAA and the Springer Press publication team for their help and support in the development of this text.

Last but not least, we would like to thank our families for their encouragement and support, as they endured the time demands that writing a book has imposed on us.

Austin, TX, USA — Bei Yu
Austin, TX, USA — David Z. Pan
July 2015

Contents

Acronyms

CD	Critical dimension
CP	Character projection
DFM	Design for manufacturability
DPL	Double patterning lithography
DSA	Directed self-assembly
EBL	Electron beam lithography
EUV	Extreme ultra violet
ICC	Independent component computation
ILP	Integer linear programming
IVR	Iterative vertex removal
LELE	Litho-etch-letho-etch process
MCC	Multi-column cell
MPL	Multiple patterning lithography
MPLD	Multiple patterning layout decomposition
NIL	Nanoimprint lithography
OPC	Optical proximity correction
OSP	Overlapping aware stencil planning
QPL	Quadruple patterning lithography
QPL	Quadruple patterning layout decomposition
RET	Reticle enhancement technique
SADP	Self-aligned double patterning
SDP	Semidefinite programming
TPL	Triple patterning lithography
TPLD	Triple patterning layout decomposition
VLSI	Very large scale integrated circuits
VSB	Variable shaped beam

Chapter 1
Introduction

Shrinking the feature size of very large scale integrated (VLSI) circuits 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 challenge for sub-32 nm technology nodes [1, 2]. Before addressing these challenges, we introduce some preliminaries of the current main-stream 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 [3]:

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

where λ is the wavelength of the light source (currently 193 nm), k_1 is the 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) [4]. 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 and wafer. But it is hard to find new liquid material to get more than 1.35 NA value in the near future [5]. Therefore, the current optical lithography system is reaching its fundamental limit and severe variations are observed on the wafer at sub-32 nm 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 Figs. 1.2 and 1.3, in MPL the original layout design is divided into several masks. Then

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_1

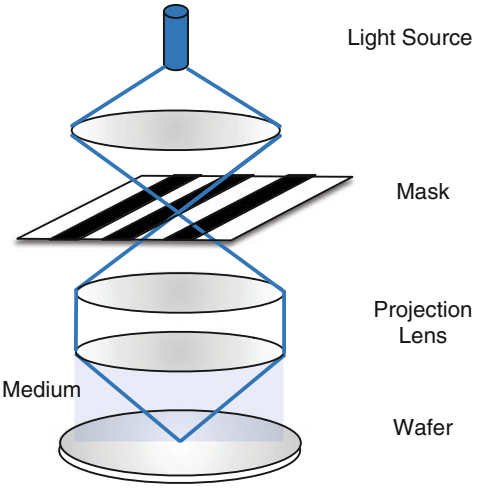

Fig. 1.1 Schematic diagram of conventional lithography system

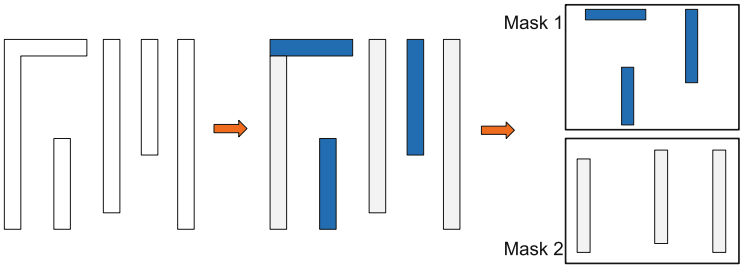

Fig. 1.2 Double patterning lithography (DPL) process

each mask is implemented through one exposure-etch step, through which the layout can be produced. Generally speaking, MPL consists of double patterning lithography (DPL) (Fig. 1.2), triple patterning lithography (TPL) (see Fig. 1.3), or even quadruple patterning lithography (QPL) [6–8]. There are two main types of DPL with different manufacturing processes: litho-etch-litho-etch (LELE) [6] and self-aligned double patterning (SADP) [9]. The advantage of MPL is that the effective pitch can improve thus the lithography resolution can be further enhanced [10]. Thus DPL has been heavily developed by industry for 22 nm technology node, while triple patterning or quadruple patterning has been explored in industry test-chip designs [11].

In the longer future (for the logic node beyond 14 nm), 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) [1]. As shown in Fig. 1.4, EBL is a maskless technology that shoots desired patterns directly into the silicon wafer using charged particle beam [12]. 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

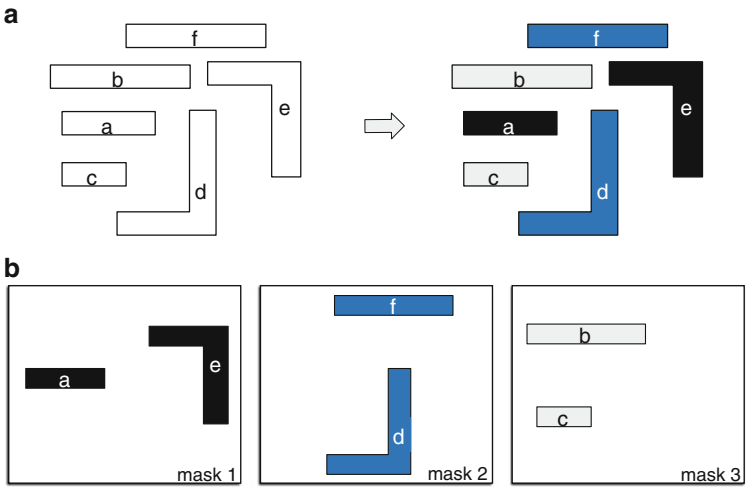

Fig. 1.3 Multiple patterning lithography (MPL) process: the initial layout is divided into several masks, and then each mask is implemented through one exposure-etch step

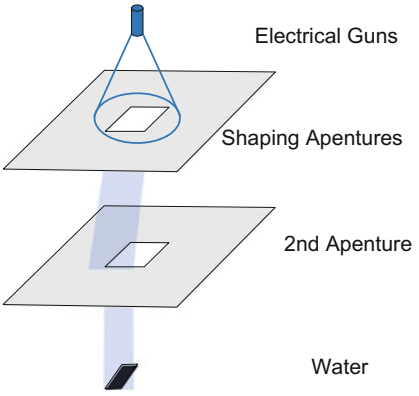

Fig. 1.4 Electron beam lithography (EBL) process

critical dimension (CD) control. In addition, due to the capability of accurate pattern generation, EBL system has been developed for several decades [13]. 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 also be resolved by inserting stitch to split a pattern into two touching parts. However, this introduces stitches, which lead to yield loss because of overlay error. Therefore, two of the main objectives in layout decomposition are conflict minimization and stitch minimization. An example of triple patterning layout decomposition is shown in Fig. 1.3, where all features in input layout are divided into three masks (colors).

Challenges 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. 1.4, 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 [14]. 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, EUV is challenged by issues such as lack of power sources, resists, and defect-free masks [15, 16]. Directed self-assembly (DSA) is a technique to phase block copolymers to construct nanostructure. However, so far this technique can only be used to generate contact or via layer patterns [17].

1.2 Overview of This Book

In this book, we present our research results on design for manufacturing (DFM) for MPL and EBL [2, 7, 8, 18–22]. Figure 1.5 shows the typical design flow and our proposed research works in the corresponding design stages. The goal of this book is to resolve three DFM challenges in advanced lithography: new patterning technique, new design methodology, and new EBL system.

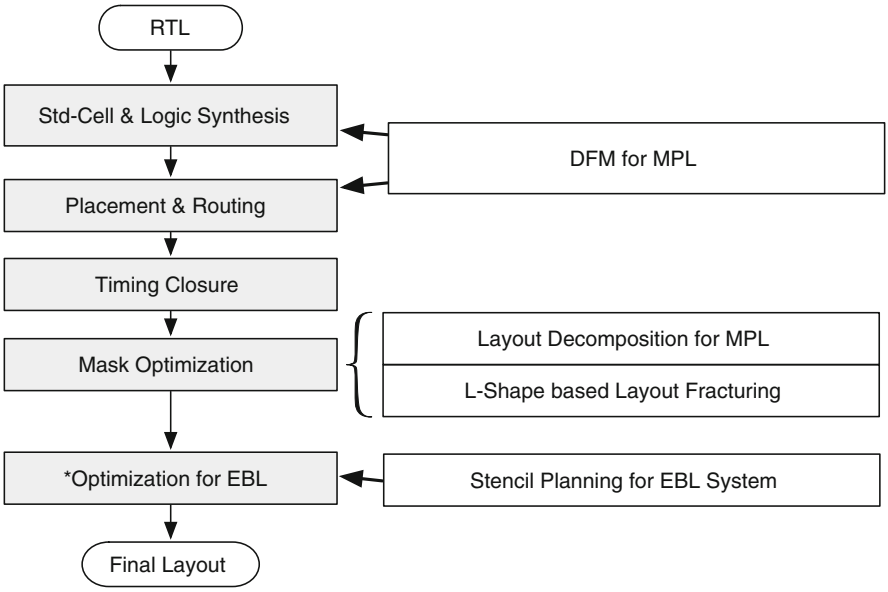

Fig. 1.5 The proposed DFM techniques in their corresponding design stages

References

1. Pan, D.Z., Yu, B., Gao, J.-R.: Design for manufacturing with emerging nanolithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(10), 1453–1472 (2013)
2. Yu, B., Gao, J.-R., Ding, D., Ban, Y., Yang, J.-S., Yuan, K., Cho, M., Pan, D.Z.: Dealing with IC manufacturability in extreme scaling. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 240–242 (2012)
3. Mack, C.: Fundamental Principles of Optical Lithography: The Science of Microfabrication. Wiley, Chichester (2008)
4. Wong, A.K.-K.: Resolution Enhancement Techniques in Optical Lithography, vol. 47. SPIE Press (2001). DOI:10.1117/3.401208
5. Lin, B.J.: Successors of ArF water-immersion lithography: EUV lithography, multi-e-beam maskless lithography, or nanoimprint? J. Micro/Nanolithog. MEMS MOEMS **7**(4), 040101 (2008)
6. Kahng, A.B., Park, C.-H., Xu, X., Yao, H.: Layout decomposition for double patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 465–472 (2008)
7. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2011)
8. Yu, B., Pan, D.Z.: Layout decomposition for quadruple patterning lithography and beyond. In: ACM/IEEE Design Automation Conference (DAC), pp. 53:1–53:6 (2014)
9. Zhang, H., Du, Y., Wong, M.D., Topaloglu, R.: Self-aligned double patterning decomposition for overlay minimization and hot spot detection. In: ACM/IEEE Design Automation Conference (DAC), pp. 71–76 (2011)
10. Lucas, K., Cork, C., Yu, B., Luk-Pat, G., Painter, B., Pan, D.Z.: Implications of triple patterning for 14 nm node design and patterning. In: Proceedings of SPIE, vol. 8327 (2012)

11. Bakshi, V.: EUV Lithography, vol. 178. SPIE Press (2009). DOI:10.1117/3.769214
12. Pain, L., Jurdit, M., Todeschini, J., Manakli, S., Icard, B., Minghetti, B., Bervin, G., Beverina, A., Leverd, F., Broekaart, M., Gouraud, P., Jonghe, V.D., Brun, P., Denorme, S., Boeuf, F., Wang, V., Henry, D.: Electron beam direct write lithography flexibility for ASIC manufacturing an opportunity for cost reduction. In: Proceedings of SPIE, vol. 5751 (2005)
13. Pfeiffer, H.C.: New prospects for electron beams as tools for semiconductor lithography. In: Proceedings of SPIE, vol. 7378 (2009)
14. Yuan, K., Yu, B., Pan, D.Z.: E-beam lithography stencil planning and optimization with overlapped characters. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **31**(2), 167–179 (2012)
15. Arisawa, Y., Aoyama, H., Uno, T., Tanaka, T.: EUV flare correction for the half-pitch 22nm node. In: Proceedings of SPIE, vol. 7636 (2010)
16. Wagner, C., Harned, N.: EUV lithography: lithography gets extreme. Nat. Photon. **4**(1), 24–26 (2010)
17. Chang, L.-W., Bao, X., Chris, B., Philip Wong, H.-S.: Experimental demonstration of aperiodic patterns of directed self-assembly by block copolymer lithography for random logic circuit layout. In: IEEE International Electron Devices Meeting (IEDM), pp. 33.2.1–33.2.4 (2010)
18. Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., Pan, D.Z.: A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 163–169 (2013)
19. Yu, B., Gao, J.-R., Pan, D.Z.: Triple patterning lithography (TPL) layout decomposition using end-cutting. In: Proceedings of SPIE, vol. 8684 (2013)
20. Yu, B., Xu, X., Gao, J.-R., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 349–356 (2013)
21. Yu, B., Yuan, K., Gao, J.-R., Pan, D.Z.: E-BLOW: e-beam lithography overlapping aware stencil planning for MCC system. In: ACM/IEEE Design Automation Conference (DAC), pp. 70:1–70:7 (2013)
22. Yu, B., Gao, J.-R., Pan, D.Z.: L-shape based layout fracturing for e-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 249–254 (2013)

Chapter 2
Layout Decomposition for Triple Patterning

2.1 Introduction

Layout decomposition is a key challenge for multiple patterning lithography (MPL) (Fig. 2.1). 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 also be 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 [1–6]. A complete flow was proposed in [1] to optimize splitting locations with integer linear programming (ILP). Xu et al. [3] provided an efficient graph reduction-based algorithm for stitch minimization. Yang et al. [4] and Tang and Cho [6] proposed min-cut based approaches to reduce stitch number. To enable simultaneous conflict and stitch minimization, ILP was adopted [1, 2] with different feature pre-slicing techniques. A matching based decomposer was proposed to minimize both the conflict number and the stitch number [5].

It shall be noted that for double patterning, even with stitch insertion, there may be some native conflicts [7]. Figure 2.2a 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. 2.2b. 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 [8].

There are investigations on triple patterning aware design [9–11] and triple patterning layout decomposition (TPLD) [8, 12–20]. Cork et al. [12] proposed a three coloring algorithm adopting SAT formulation. Ghaida et al. [21] reused the double patterning techniques. Fang et al. [14], Kuang and Young [16],

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_2

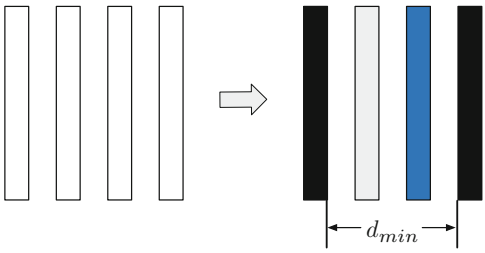

Fig. 2.1 TPL layout decomposition example

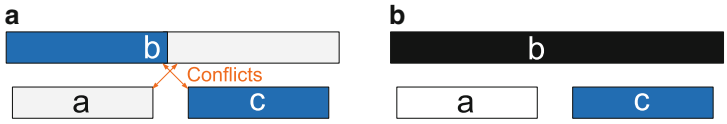

Fig. 2.2 (**a**) In double patterning, even stitch insertion cannot avoid the native conflicts. (**b**) Native conflict in double patterning might be resolved by triple patterning

Zhang et al. [18], and Chen et al. [20] proposed different heuristic methods for the TPLD problem. For row based layout design, [15, 17] presented polynomial time decomposition algorithms.

2.2 Layout Decomposition for Triple Patterning

2.2.1 Preliminaries and Problem Formulation

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

Layout Graph and Decomposition Graph

Given an input layout specified by features in polygonal shapes, construct a *layout graph* [1] by Definition 2.1.

Definition 2.1 (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 the minimum coloring distance min_s.

One example of layout graph is illustrated in Fig. 2.3a. 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 [1] is performed, where projected segments are highlighted by bold lines in Fig. 2.3b.

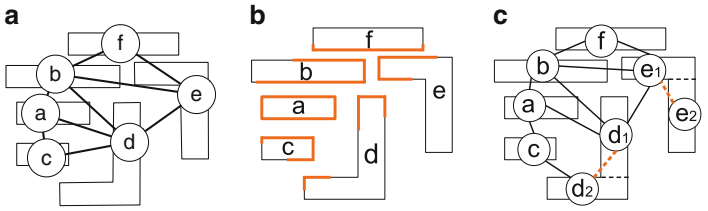

Fig. 2.3 Layout graph construction and decomposition graph construction. (**a**) Layout graph for given input, where all edges are conflict edges; (**b**) the vertex projection; (**c**) corresponding decomposition graph, where *dash edges* are stitch edges

Based on the projection result, all the legal splitting locations are computed. Then a *decomposition graph* [22] is constructed by Definition 2.2.

Definition 2.2 (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 its two vertices are within the minimum coloring distance min_s. An edge is in SE iff its two vertices are associated with the same polygonal shape, necessitating a stitch.

An example of a decomposition graph (DG) is shown in Fig. 2.3c. Note that the conflict edges are marked with black lines, while stitch edges are marked with dashed lines. Here each stitch edge is a stitch candidate.

Stitch Candidate Generation

Stitch candidate generation is one of the most important steps in parsing a layout, as it not only determines the number of vertices 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. Kahng et al. [1] and Xu and Chu [3] propose different methodologies for generating the DP candidates. In this section, we show that DP candidates may be redundant or lose some useful candidates, and that they cannot be directly applied in TPLD problem. We therefore provide a procedure to generate appropriate stitch candidates for triple patterning lithography.

Here, we provide two examples demonstrating 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. 2.4a, the stitch can be removed because no matter what color is assigned to features b and c, 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

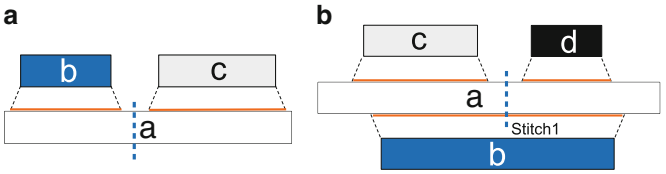

Fig. 2.4 Examples of (**a**) redundant stitch; (**b**) lost stitch

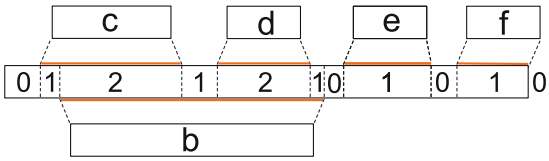

Fig. 2.5 The projection sequence of the feature is 01212101010, and the last 0 is a default terminal zero

may cause the stitch loss problem, i.e., some useful stitch candidates cannot be detected and inserted in layout decomposition. In DPL, stitch candidates have just one requirement: they cannot intersect any projection. For example, as shown in Fig. 2.4b, because this stitch intersects with the projection of feature b, it cannot be a DP candidate. However, if features b, c, and d are assigned with three different colors, introducing the stitch is required to resolve the conflict. In other words, the requirement for stitches 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 requirement as *lost stitches*.

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

Definition 2.3 (Projection Sequence). After the projection, the feature is divided into several segments, each of which is labeled with a number representing how many other features are projected onto it. The sequence of numbers on these segments is the projection sequence.

Instead of analyzing each feature and all of its respective 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. 2.5, where the middle feature has five conflicting features, b, c, d, e, and f. Based on the projection results, the feature is divided into ten segments. After 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 the projection sequence, we summarize the rules for redundant stitches and lost stitches. First, motivated by the case in Fig. 2.4a, we can summarize the redundant stitches as follows: if the projection sequence begins with

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**

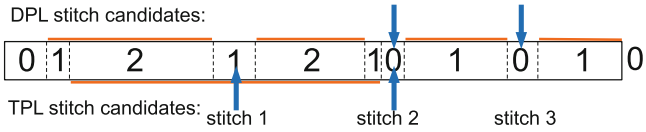

Fig. 2.6 Stitch candidates generated for DPL and TPL

"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. Thus, 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. 2.4b 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, each multiple-pin feature is first decomposed into several two-pin features. For each feature, we can then 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. 2.6. In 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.

Problem Formulation

We define the *TPLD* problem as follows.

Problem 2.1 (Triple Patterning Layout Decomposition). Given a layout specified by features in polygonal shapes, we can construct the decomposition graphs. TPLD assigns all the vertices of DG one of three colors (masks) with the goal of minimizing the costs of the stitches and the conflicts.

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

TPLD problem is an extension of the double patterning layout decomposition (DPLD) problem, and both of them simultaneously minimize the conflict number and the stitch number. For the DPLD problem, Xu et al. [5] showed that if the decomposition graph is planar, it can be resolved in polynomial time. At first glance, compared with DPLD, TPLD seems easier due to the presence of an additional 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 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 for 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 share the same color. The PG3C problem seeks to minimize the coloring conflict number. We have the following lemma:

Lemma 2.1. *The PG3C problem is NP-hard.*

The correctness of Lemma 2.1 stems from the conclusion that deciding whether a planar graph is 3-colorable is NP-complete [23]. For a planar graph, checking whether it is 3-colorable cannot be finished in polynomial time; therefore, 3-coloring a planar graph with minimum cost cannot be finished in polynomial time.

Theorem 2.1. *TPLD problem is NP-hard.*

Proof. We prove this theorem by showing that 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** [24], where the drawing plane is subdivided by horizontal and vertical gridlines of unit spacing λ. The vertices $\{V\}$ are represented by rectangles placed such that the borders of the rectangles overlap the gridlines by $\lambda/4$. The edges $\{E\}$ are mapped to non-overlapping paths in the gridlines. Please refer to [24] 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 the PG3C instance. For example, given a PG3C instance in Fig. 2.7a, the corresponding orthogonal drawing and TPLD instance are illustrated in Fig. 2.7b, c, respectively. Here no stitch candidate is introduced. Since an orthogonal drawing

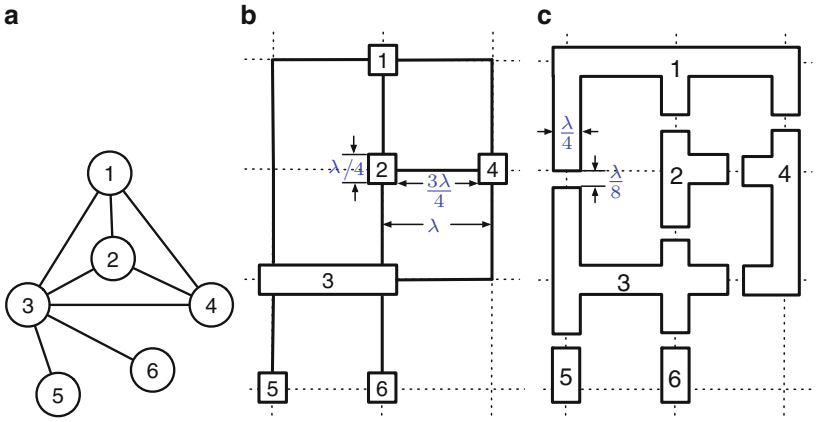

Fig. 2.7 Reducing PG3C to TPLD. (**a**) An instance of PG3C; (**b**) the transferred orthogonal drawing; (**c**) the corresponding TPLD instance

can be constructed in polynomial time [25], the whole reduction can be finished in polynomial time. Minimizing the number of conflicts in the original PG3C instance is thus equivalent to minimizing the number of conflict in the constructed TPLD instance, which completes the proof.

2.2.2 Algorithms

The overall decomposition flow is illustrated in Fig. 2.8. 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 a 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 for each DG component, we propose two approaches. One is based on ILP, which can resolve the problem exactly but may suffer a runtime overhead. The other is a semidefinite programming (SDP) based algorithm: instead of using ILP, we formulate the problem into a vector programming problem, then its relaxed version can be resolved through SDP. After a mapping stage, the SDP solution can then be translated into a color assignment solution. Finally, we merge all DG components together to achieve the final TPLD result.

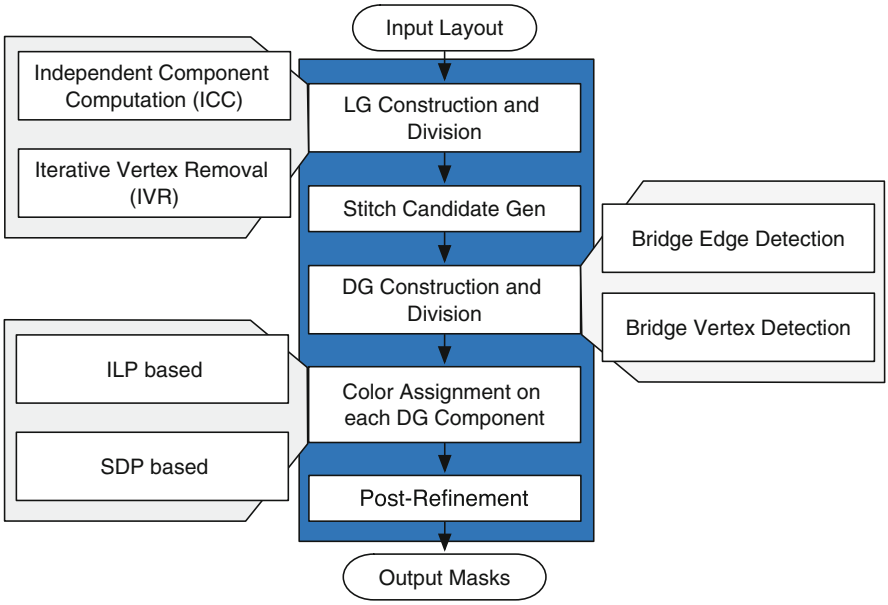

Fig. 2.8 Overview of our decomposition flow

ILP Based Color Assignment

On the decomposition graph (DG), we carry out a color assignment, which is a critical step in the layout decomposition flow. Color assignment gives each vertex one of three colors (masks). First, we will give a general mathematical formulation of the color assignment. Then, we will show that it can be solved through ILP, which is commonly used for the DPLD problem [1, 2].

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 number of conflicts and the number of stitches. The parameter α is a user-defined parameter for assigning relative importance between the conflicts and stitches.

$$\min \quad \sum_{e_{ij} \in CE} c_{ij} + \alpha \sum_{e_{ij} \in SE} s_{ij} \tag{2.1}$$

$$\text{s.t.} \quad c_{ij} \leftarrow (x_i = x_j) \qquad \forall e_{ij} \in CE \tag{2.1a}$$

$$s_{ij} \leftarrow x_i \oplus x_j \qquad \forall e_{ij} \in SE \tag{2.1b}$$

$$x_i \in \{0, 1, 2\} \qquad \forall i \in V \tag{2.1c}$$

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 ith 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}

In Eq. (2.1), x_i is a variable for the three colors of rectangles r_i, c_{ij} is a binary variable for each conflict edge $e_{ij} \in CE$, and s_{ij} is a binary variable for each stitch edge $e_{ij} \in SE$. Constraint (2.1a) is used to evaluate the number of conflicts when touch vertices r_i and r_j are assigned the same color (mask). Constraint (2.1b) is used to calculate the number of stitches. 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. (2.1a) and (2.1b) can be linearized only when x_i is a 0–1 variable [1], for which it is difficult 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 as (2.2).

$$\min \sum_{e_{ij} \in CE} c_{ij} + \alpha \sum_{e_{ij} \in SE} s_{ij} \tag{2.2}$$

$$\text{s.t. } x_{i1} + x_{i2} \leq 1 \tag{2.2a}$$

$$x_{i1} + x_{j1} \leq 1 + c_{ij1} \quad \forall e_{ij} \in CE \tag{2.2b}$$

$$(1 - x_{i1}) + (1 - x_{j1}) \leq 1 + c_{ij1} \quad \forall e_{ij} \in CE \tag{2.2c}$$

$$x_{i2} + x_{j2} \leq 1 + c_{ij2} \quad \forall e_{ij} \in CE \tag{2.2d}$$

$$(1 - x_{i2}) + (1 - x_{j2}) \leq 1 + c_{ij2} \quad \forall e_{ij} \in CE \tag{2.2e}$$

$$c_{ij1} + c_{ij2} \leq 1 + c_{ij} \quad \forall e_{ij} \in CE \tag{2.2f}$$

$$x_{i1} - x_{j1} \leq s_{ij1} \quad \forall e_{ij} \in SE \tag{2.2g}$$

$$x_{j1} - x_{i1} \leq s_{ij1} \quad \forall e_{ij} \in SE \tag{2.2h}$$

$$x_{i2} - x_{j2} \leq s_{ij2} \quad \forall e_{ij} \in SE \tag{2.2i}$$

$$x_{j2} - x_{i2} \leq s_{ij2} \qquad \forall e_{ij} \in SE \tag{2.2j}$$

$$s_{ij} \geq s_{ij1}, s_{ij} \geq s_{ij2} \qquad \forall e_{ij} \in SE \tag{2.2k}$$

$$x_{ij} \text{ is binary} \tag{2.2l}$$

The objective function is the same as in (2.1), where both minimize the weighted summation of the number of conflicts and stitches. 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)–(2.2f) are equivalent to constraint (2.1a), where the 0–1 variable c_{ij1} demonstrates whether x_{i1} equals to x_{j1}, and c_{ij2} demonstrates whether x_{i2} equals to x_{j2}. The 0–1 variable c_{ij} is true only if two vertices connected by conflict edge e_{ij} have the same color, e.g. both c_{ij1} and c_{ij2} are true.

Constraints (2.2g)–(2.2k) are equivalent to constraint (2.1b). The 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.

SDP Based Color Assignment

Although ILP formulation (2.2) can optimally solve the color assignment problem in theory, for practical design it may suffer from a runtime overhead problem. In this section, we show that instead of expensive ILP, the color assignment can be also formulated using vector programming, with three unit vectors representing three different colors. The vector programming problem is then relaxed and solved through SDP. Given the solutions of SDP, we develop a mapping process to obtain the final color assignment solutions. Note that our algorithm is fast enough such that both SDP formulation and the mapping process can be finished in polynomial time.

There are three possible colors in the color assignment. We set a unit vector $\vec{v}_i$ for every vertex v_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. 2.9, 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 $2\pi/3$.

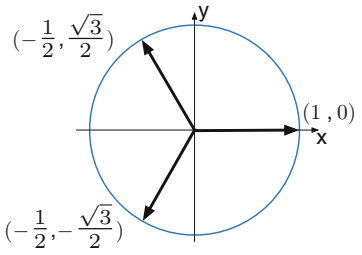

Fig. 2.9 Three vectors $(1, 0), (-\frac{1}{2}, \frac{\sqrt{3}}{2}), (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$ represent three different colors

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 [26]:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3} \left(\vec{v}_i \cdot \vec{v}_j + \frac{1}{2} \right) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} (1 - \vec{v}_i \cdot \vec{v}_j) \tag{2.3}$$

$$\text{s.t. } \vec{v}_i \in \left\{ (1,0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2} \right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2} \right) \right\} \tag{2.3a}$$

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 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:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3} \left(\vec{y}_i \cdot \vec{y}_j + \frac{1}{2} \right) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} (1 - \vec{y}_i \cdot \vec{y}_j) \tag{2.4}$$

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

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

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 $\vec{y}_i = (v_{i1}, v_{i2}, 0, 0, \ldots, 0)$, i.e., $\vec{y}_i \cdot \vec{y}_j = 1$ and $\vec{y}_i \cdot \vec{y}_j = \vec{v}_i \cdot \vec{v}_j$ in this solution. Here, the dimension of vector $\vec{y}_i$ is $|V|$, that is, the number of vertices in the 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, the solution to (2.4) provides a lower bound approximation to that in (2.3). After removing the constant factor from the objective function, we derive the following vector programming problem.

$$\min \sum_{e_{ij} \in CE} (\vec{y}_i \cdot \vec{y}_j) - \alpha \sum_{e_{ij} \in SE} (\vec{y}_i \cdot \vec{y}_j) \tag{2.5}$$

$$\text{s.t. } (2.4\text{a})\text{–}(2.4\text{b})$$

Without the discrete constraint (2.3a), programs (2.4) and (2.5) are no longer NP-hard. To solve (2.5) in polynomial time, we will show that it is equivalent to a SDP problem. SDP is similar to LP in that both have 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. The relaxation based on SDP actually has better theoretical results than those based on LP [27].

Consider the following standard SDP:

$$\text{SDP: } \min \quad A \bullet X \tag{2.6}$$

$$x_{ii} = 1, \quad \forall i \in V \tag{2.6a}$$

$$x_{ij} \geq -\frac{1}{2}, \quad \forall e_{ij} \in CE \tag{2.6b}$$

$$X \succeq 0 \tag{2.6c}$$

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

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

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

Lemma 2.2. *A symmetric matrix X is positive semidefinite if and only if $X = VV^T$ for some matrix V.*

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

Theorem 2.2. *The semidefinite program (2.6) and the vector program (2.5) are equivalent.*

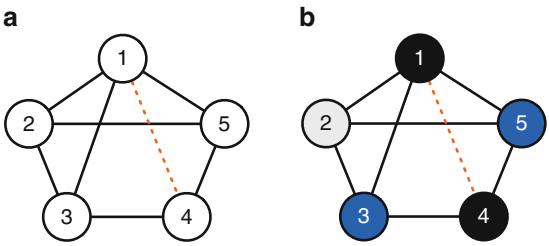

Fig. 2.10 A simple example of SDP. (**a**) Input DG component; (**b**) color assignment result with 0 conflicts and 0 stitches

Proof. Given solutions $\{\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_m\}$ 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 2.2, 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).

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 of $\vec{v}_i \cdot \vec{v}_j$. Instead of trying to calculate all $\vec{v}_i$ through Cholesky decomposition, we pay attention to the x_{ij} value itself. Basically, if x_{ij} is close to 1, then vertices i and j tend to be in the same color; if x_{ij} is close to -0.5, vertices i and j tend to be in different colors.

For most of the 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. 2.10. 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 Eq. (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:

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

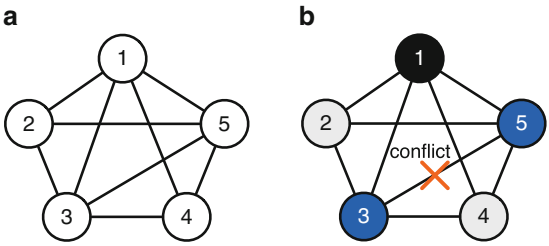

Fig. 2.11 A complex example. (**a**) Input DG component; (**b**) color assignment result with 1 conflict

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. 2.10b.

The matrix X generated from Fig. 2.10 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 containing conflicts and stitches, some values in the matrix X may not be so clear.

We use Fig. 2.11 for illustration. The decomposition graph in Fig. 2.11a contains a four-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).

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

From X, we can see that $x_{24} = 1.0$. Vertices 2 and 4 should therefore share the same color. It is not as clear-cut for x_{13}, x_{15}, and x_{35}, which are all -0.13. For those vague values, we propose a mapping process to discover the final color assignment. The mapping algorithm is as follows. 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 to calculate the guideline used 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), triplets are constructed from particular values and sorted by x_{ij} (lines 1–2). Our mapping can then 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

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**

be close to 1, and th_{sp} should be close to -0.5. If $x_{ij} > th_{unn}$, implying that x_{ij} is close to 1, then we apply operation **Union**(i, j) to merge vertices r_i and r_j to form a large vertex (i.e., they are in the same color). Similarly, if $x_{ij} < th_{sp}$, implying that x_{ij} is close to -0.5, then operation **Separate**(i, j) is used to label vertices r_i and r_j 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 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 the largest remaining 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 on the disjoint-set is amortized constant time [28]. Let n be the number of vertices, then the number of triplets is $O(n^2)$. Sorting all the triplets requires $O(n^2 \log n)$. 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 $O(n^2 \log n)$. Applying Algorithm 2 to the matrix in (2.8), we can get the final color assignment (see Fig. 2.11b), where one conflict between vertices 3 and 5 is reported.

Graph Division

To achieve further speed-up, instead of solving the color assignments on one decomposition graph (DG), we propose several techniques to divide the graph into a bunch of components. Then the color assignment can be solved for each component independently.

Given an input layout, first a layout graph (LG) is constructed. We propose two methods for dividing and simplifying the LG in order to reduce the problem size.

The first division technique is called *ICC*. In a layout graph representing a real design, we observe many isolated clusters. By breaking down the whole layout

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**

graph into several independent components, we partition the initial large layout graph into several small ones. After solving the TPLD problem for each isolated component, the overall solution can be taken as the union of all the components without affecting the global optimality. It should be noted that ICC is a well-known technique that 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 second technique is called IVR, as described in Algorithm 3. Initially, 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 the other vertices. This removing process will continue until all the vertices are at least of degree three. All the vertices that are temporarily removed are stored in stack S. The decomposition graph is then constructed for the remaining vertices. After solving the color assignment for each DG component, the removed vertices are recovered one by one.

If all the vertices in one layout graph can be temporarily removed (pushed onto the stack S), the TPLD problem can be solved optimally in linear time. An example is illustrated in Fig. 2.12, where all the vertices can finally be pushed onto the stack. Even though some vertices still remain, our IVR technique can shrink the problem size dramatically. We observe that this technique can also further partition the layout graph into several independent components.

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

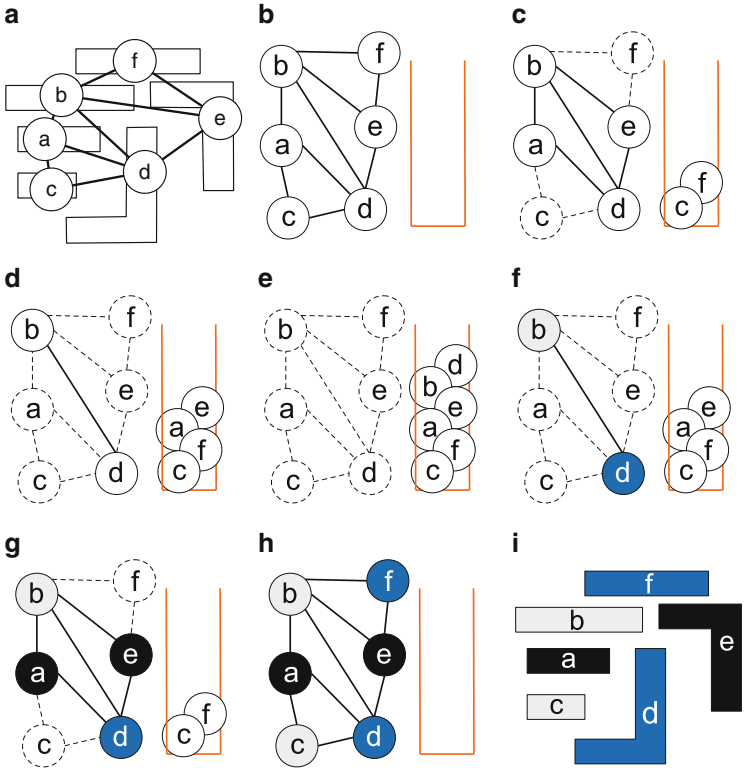

Fig. 2.12 An example of iterative vertex removal (IVR), where the TPLD problem can be solved in linear time: (**a**) layout graph; (**b**, **c**, **d**, **e**) iteratively remove and push in vertices whose degrees are less than three; (**f**, **g**, **h**) after assigning colors for the remaining vertices, iteratively pop up and recover the vertices and assign any legal color; (**i**) TPLD can be finished after the iterative vertex recovery

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. 2.13. 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 [29] to detect all bridge edges in the 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. 2.14. This simplification method is effective

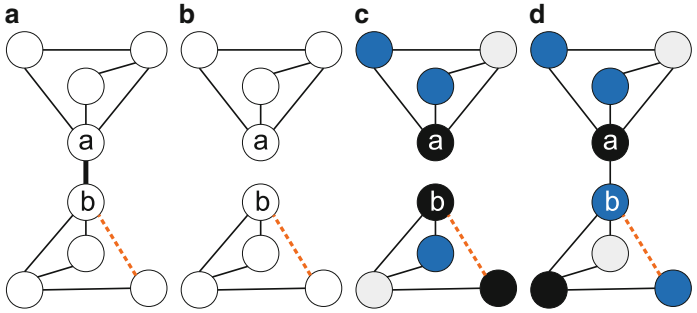

Fig. 2.13 Bridge edge detection and removal. (**a**) Initial decomposition graph. (**b**) After bridge edge detection, remove edge e_{ab}. (**c**) In two components we carry out layout decomposition. (**d**) Rotate colors in the lower component to add bridge

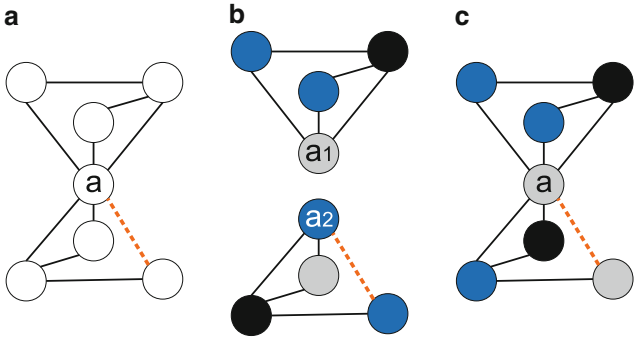

Fig. 2.14 Bridge vertex detection and duplication. (**a**) Initial decomposition graph. (**b**) After bridge vertex detection, duplicate vertex a. (**c**) Rotate the colors in lower sub-graph to merge vertices a_1 and a_2

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.

Post Refinement

Although the graph division techniques can dramatically reduce the computational time to solve the TPLD problem, Kuang and Young [16] pointed out that for some cases IVR may lose some optimality. One example is illustrated in Fig. 2.15. The simplified layout graph (Fig. 2.15b) can be inserted stitch candidates and assigned legal colors (see Fig. 2.15b). However, when we recover removed vertices, the vertex degree of a increases to 3, and there is no available color for it (see Fig. 2.15c). The reason for this conflict is that during stitch candidate generation, vertex a is not considered.

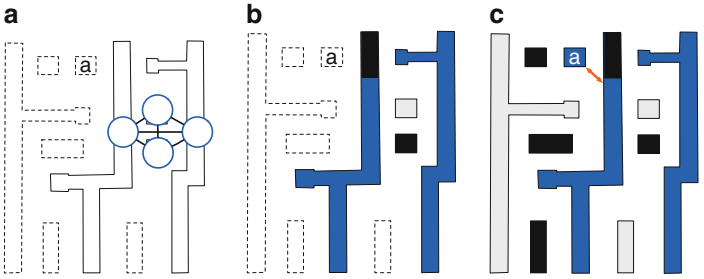

Fig. 2.15 Iterative vertex removal may introduce additional conflicts. (**a**) Layout graph after iterative vertex removal; (**b**) stitch generation and color assignment on the graph; (**c**) after adding back the simplified vertices, one additional conflict is introduced to vertex *a*

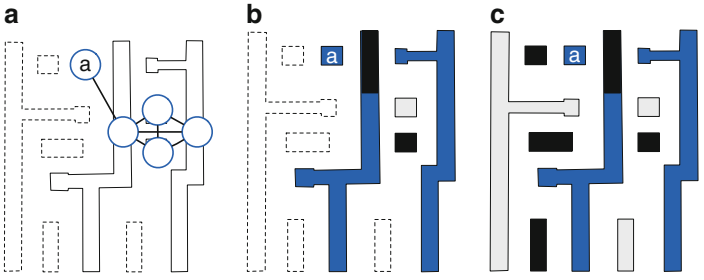

Fig. 2.16 An example of post-refinement. (**a**) Extend layout graph to include *a*; (**b**) stitch generation and color assignment on the new graph; (**c**) no additional conflict at final solution

We propose a post-refinement to resolve the conflicts caused by IVR. 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. 2.16. The initial layout graph would be extended to include vertex *a* (Fig. 2.16a), thus the position of vertex *a* would be considered during stitch candidate generation. As shown in Fig. 2.16c, no additional conflict would be introduced after recovering all vertices from stack. Note that according to our initial results, although re-solving color assignment requires more computational time, only small part of DG components need to apply the post-stage.

2.2.3 Experimental Results

We implemented our algorithm in C++ and tested it on an Intel Core 2.9 GHz Linux machine. We chose GUROBI [30] as the ILP solver and CSDP [31] as the SDP solver. ISCAS benchmarks from [2, 4] were scaled down and modified for use as our test cases. The metal one layer was used for experimental purposes, because it represents one of the most complex layers in terms of layout decomposition.

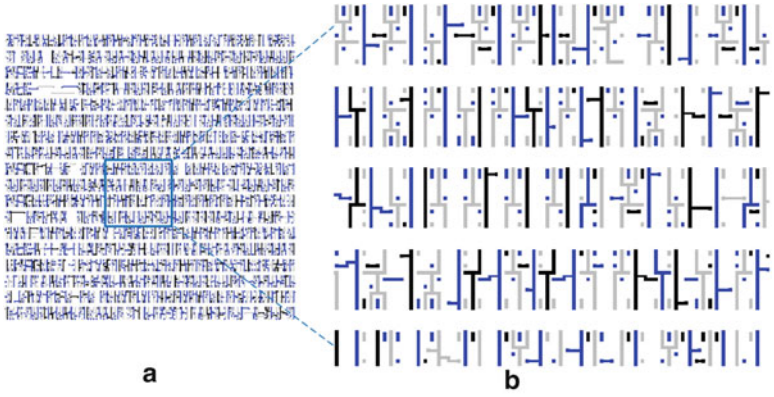

Fig. 2.17 Part of S1488 decomposition result

The minimum color spacing min_s was set to 120 for the first ten cases and 100 for the last five cases, as in [8, 13, 14]. Parameter α was set to 0.1, so the decomposition cost was equivalent to $cn\# + 0.1 \cdot st\#$, where $cn\#$ and $st\#$ denote the conflict number and the stitch number, respectively.

Figure 2.17 illustrates part of the decomposition result for case S1488, which can be decomposed in 0.1 s.

First, we demonstrate the effectiveness of our stitch candidate generation. Table 2.2 compares the performance and runtime of ILP for 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 all graph division techniques are applied here. The columns "st#" and "cn#" denote the number of stitches and conflicts, respectively. Column "CPU(s)" is computational time in seconds. As discussed in Sect. 2.2.1, applying TP stitch generates more stitch candidates. We can see from Table 2.2 that 30 % more runtime would be introduced as a result. However, TP stitch overcomes the lost stitch problem present in DP stitch, so the decomposition cost is reduced by 40 %. In other words, compared with DP stitch, TP stitch can provide higher performance in terms of the number of stitches and conflicts.

Second, we show the effectiveness of the graph division, which consists of a set of techniques: ICC, IVR, and bridge detection. Through applying these division techniques, the decomposition graph size can be reduced. Generally speaking, ILP requires less runtime for a smaller decomposition graph. 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 number of stitch edges and conflict edges, respectively. From Table 2.3, we can see that, compared with only using the ICC technique, also applying IVR and bridge detection is more effective: the number of stitch edges can be reduced by 92 %, while the number of conflict edges can be reduced by 93 %.

Table 2.2 DP stitch v.s. TP stitch

	ILP w/o. TP stitch				ILP w. TP stitch			
Circuit	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**

The columns "st#" and "cn#" show the number of stitches and conflicts 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 runtime for some smaller cases. For some large circuits, the runtime of "ILP w. ICC" is unacceptable, i.e., longer than 2 h. Note that if no ICC technique is used, even for small circuits like C432, the runtime of ILP is unacceptable.

Here, we discuss the SDP solutions in greater detail. 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. Figure 2.18 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 this post-refinement. Table 2.4 lists the decomposition results of two SDP-based algorithms, where the columns indicate whether refinement was used. As shown in Table 2.4, by adding an additional post-refinement stage, the decomposition costs can be reduced by 14 % at the expense of only 6 % additional computational time.

Finally, we compare our decomposition algorithms with state-of-the-art layout decomposers [13, 14], as shown in Table 2.5. Columns "**ILP w. All**" and

Table 2.3 Effectiveness of graph division

	ILP w. ICC						ILP w. 4SPD					
Circuit	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	11,346	4214	507	1	0	55.4	62	21	3	1	0	0.31
C2670	19,716	7026	614	6	0	72.7	148	56	10	7	1	0.5
C3540	24,076	8849	827	8	1	100.6	218	90	20	8	2	1.31
C5315	34,668	12,789	1154	9	0	134.4	263	111	18	9	1	0.94
C6288	31,943	11,276	2325	215	1	340.7	3389	1386	293	217	18	16.2
C7552	49,496	18,407	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	122,120	52,726	5298	56	19	653.2	10,432	5255	1510	76	19	5.14
S35932	288,171	115,739	15,804	N/A	N/A	>7200	32,073	16,237	4713	99	47	13.15
S38584	299,613	127,063	16,235	N/A	N/A	>7200	30,663	15,515	4517	157	43	11.6
S15850	298,423	125,722	13,226	N/A	N/A	>7200	26,468	13,312	3807	130	40	10.7
Avg.	80,722	33,053	–	–	–	>1554.7	6984.5	3495.2	–	49.9	11.7	4.25
Ratio	**11.6**	**9.46**	–	–	–	**>365.5**	**1.0**	**1.0**	–	–	–	**1.0**

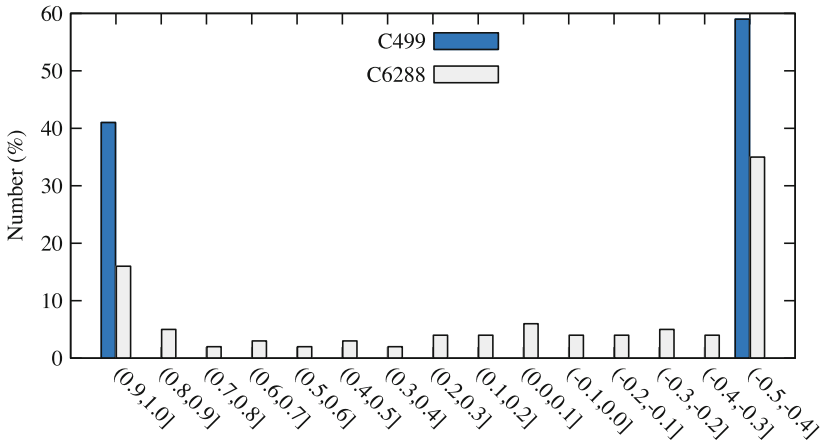

Fig. 2.18 Value distribution in matrix X for cases C499 and C6288

Table 2.4 Effectiveness of post-refinement

	SDP w/o. refinement				SDP w. refinement			
Circuit	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**

"**SDP w. All**" denote an ILP-based algorithm and our SDP-based algorithm, respectively. Here "w. All" means all other techniques, i.e. TP stitch, graph division, and post-refinement, are all applied. From Table 2.5 we can see that compared to SDP based methods, the decomposers [13, 14] are faster but incur 35 % more and 18 % greater decomposition cost, respectively. Although the ILP-based algorithm has better performance in terms of conflict number, it also has the worst runtime.

Table 2.5 Performance comparison with other decomposers

	DAC'12 [13]			TCAD [14]			ILP w. All			SDP w. All		
Circuit	st#	cn#	Cost	st#	cn#	Cost	st#	cn#	Cost	st#	cn#	Cost
C432	6	0	0.6	6	0	0.6	4	0	0.4	4	0	0.4
C499	0	0	0	0	0	0	0	0	0	0	0	0
C880	15	1	2.5	8	1	1.8	8	0	0.8	8	0	0.8
C1355	7	1	1.7	4	1	1.4	3	0	0.3	3	0	0.3
C1908	0	1	1.0	0	1	1	1	0	0.1	1	0	0.1
C2670	12	2	3.2	11	2	3.1	8	1	1.8	8	1	1.8
C3540	14	3	4.4	11	3	4.1	8	1	1.8	9	1	1.9
C5315	11	3	4.1	11	3	4.1	9	0	0.9	9	0	0.9
C6288	342	20	54.2	243	20	44.3	210	6	27	210	6	27
C7552	46	3	7.6	37	3	6.7	26	0	2.6	26	0	2.6
S1488	4	0	0.4	4	0	0.4	2	0	0.2	2	0	0.2
S38417	122	20	32.2	82	20	28.2	71	19	26.1	71	19	26.1
S35932	103	46	56.3	63	46	52.3	79	45	52.9	79	45	52.9
S38584	280	36	64	176	36	53.6	146	36	50.6	146	36	50.6
S15850	201	36	56.1	146	36	50.6	126	35	47.6	120	36	48
Avg.	77.5	11.5	19.22	53.5	11.5	16.8	46.7	9.53	14.21	46.4	9.6	14.24
Ratio	–	–	**1.35**	–	–	**1.18**	–	–	**1.00**	–	–	**1.0**

Table 2.6 Runtime comparison with other decomposers

Circuit	DAC'12 [13] CPU(s)	TCAD'14 [14] CPU(s)	ILP w. All CPU(s)	SDP w. All CPU(s)
C432	0.03	0.11	0.99	0.25
C499	0.1	0.02	0.27	0.12
C880	0.11	0.12	0.5	0.15
C1355	0.11	0.14	0.18	0.11
C1908	0.07	0.08	0.3	0.18
C2670	0.1	0.1	0.54	0.26
C3540	0.11	0.11	1.43	0.51
C5315	0.17	0.17	1.2	0.57
C6288	0.17	0.17	18.4	3.42
C7552	0.27	0.28	2.8	1.04
S1488	0.09	0.1	0.11	0.11
S38417	0.82	0.74	5.3	2.38
S35932	2.1	2.1	13.9	6.6
S38584	2.3	2.3	12.1	6.52
S15850	2.04	2	11.1	6.36
Avg.	0.57	0.57	4.61	1.91
Ratio	**0.30**	**0.30**	**2.42**	**1.0**

Compared to ILP, our SDP-based algorithm provides a much better tradeoff between runtime and performance, achieving very comparable results (1 % of conflict difference), but more than a 3× speed-up (Table 2.6).

In order to further evaluate the scalability of all the decomposers, we created six additional benchmarks ("c5_total"–"c10_total") to compare different algorithms on very dense layouts. Tables 2.7 and 2.8 compare the results. As we can see, although ILP can achieve the best decomposition results, its high runtime complexity makes it impossible to solve one of the large dense layouts, even all the graph division techniques are adopted. On the other hand, although the decomposers [13, 14] are faster, in that all the cases can be finished in 1 s, they introduce 92 % and 87 % more decomposition cost, respectively. In some cases, they introduce hundreds of conflicts more than the SDP-based algorithm. 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, the SDP-based algorithm can achieve a good tradeoff in terms of runtime and performance.

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 fewer conflicts and stitches, but also a more balanced density. We focus on the coloring algorithms

Table 2.7 Performance comparisons with other decomposers on very dense layouts

	DAC'12 [13]			TCAD [14]			ILP w. All			SDP w. All		
Circuit	st#	cn#	Cost	st#	cn#	Cost	st#	cn#	Cost	st#	cn#	Cost
c5_total	433	248	291.3	374	248	285.4	504	21	71.4	497	30	79.7
c6_total	954	522	617.4	869	521	607.9	N/A	N/A	N/A	1056	179	284.6
c7_total	1111	623	734.1	938	624	717.8	1113	283	394.3	1103	304	414.3
c8_total	1910	727	918	1671	727	894.1	1676	409	576.6	1636	433	596.6
c9_total	834	525	608.4	635	525	588.5	865	256	342.5	853	267	352.3
c10_total	2463	1144	1390.3	2120	1146	1358	2510	355	606	2472	403	650.2
Avg.	1284.2	631.5	759.9	1101.2	631.8	742.0	N/A	N/A	N/A	1269.5	269.3	396.3
Ratio	–	–	**1.92**	–	–	**1.87**	–	–	–	–	–	**1.0**

Table 2.8 Runtime comparisons with other decomposers on very dense layouts

Circuit	DAC'12 [13] CPU(s)	TCAD'14 [14] CPU(s)	ILP w. All CPU(s)	SDP w. All CPU(s)
c5_total	0.16	0.15	120.2	12.0
c6_total	0.3	0.26	>3600	33.97
c7_total	0.3	0.33	451.4	22.2
c8_total	0.53	0.51	260.1	28.4
c9_total	0.4	0.4	79.8	11.8
c10_total	0.86	0.88	802.6	50.4
Avg.	0.425	0.42	885.7	26.5
Ratio	**0.02**	**0.02**	**>33.5**	**1.0**

and leave other layout-related optimizations of 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 achieves not only fewer conflicts, but also more balanced density; (3) Propose several techniques to speed up the layout decomposition; (4) Achieve the best results in solution quality while maintaining better balanced density (i.e., less EPE).

2.3.1 Preliminaries and Problem Formulation

Why Balanced Density?

In layout decomposition, especially for TPL, density balance should also be considered along with conflict and stitch minimization. A good pattern density balance is also expected to be a consideration in mask CD and registration control [32], while unbalanced density would cause lithography hotspots as well as lowered CD uniformity due to irregular pitches [4]. However, from the algorithmic perspective, achieving a balanced density in TPL is harder than in DPL. (1) In DPL, two colors can be more implicitly balanced; by contrast, in TPL, oftentimes existing strategies try to do DPL first and then do some "patch" with the third mask, which causes a big challenge in explicitly considering the density balance. (2) Due to the additional color, the solution space is much larger [12]. (3) To reduce potential hotspots, local density balance should be considered instead of global density balance, since neighboring patterns are one of the main sources of hotspots. As shown in Fig. 2.19a, b, when only the global density balance is considered, feature *a* is assigned white

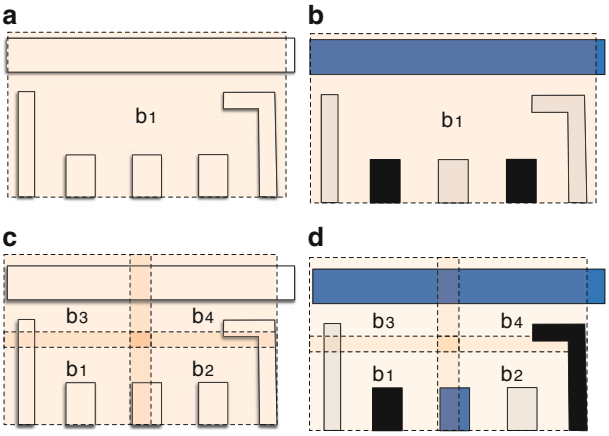

Fig. 2.19 Decomposed layout with (**a**, **b**) global balanced density. (**c**, **d**) Local balanced density in all bins

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. 2.19c). Feature a intersects bins b_1 and b_2, therefore it is colored as blue to maintain the local density balances for both bins (see Fig. 2.19d).

Problem Formulation

Given an input layout specified by features in polygonal shapes, we partition the layout into a set of n bins $S = \{b_1, \ldots, b_n\}$. Note that neighboring bins may partially overlap. 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 $d_{kc} = \sum den_{ki}$, such that any feature r_i assigned color c. Therefore, we can define the local density uniformity as follows:

Definition 2.4 (Local Density Uniformity). For bin $b_k \in S$, its local density uniformity, denoted by DU_k, is $\max\{d_{kc}\}/\min\{d_{kc}\}$ given three densities d_{k1}, d_{k2}, and d_{k3} for three masks. The local density uniformity is used to measure the ratio difference of the densities. A lower value means better local density balance.

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 2.2 (Density Balanced Layout Decomposition). Given a layout which is specified by features in polygonal shapes, the layout graphs and decomposition

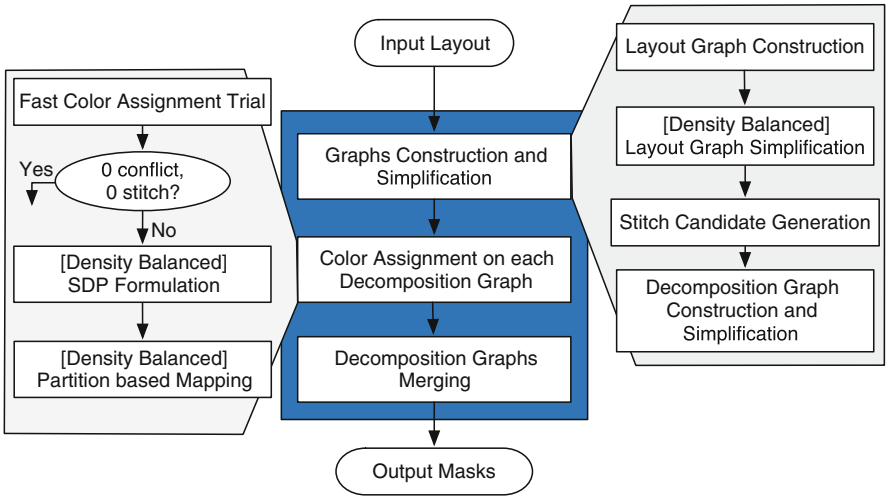

Fig. 2.20 Overall flow of proposed density balanced decomposer

graphs are constructed. Our goal is to assign all vertices in the decomposition graph three colors (masks) to minimize the number of stitches and conflicts, 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. 2.20. It consists of two stages: graph construction and simplification, and color assignment. Given the input layout, layout graphs and decomposition graphs are constructed. Then, graph simplifications [8, 13] are applied to reduce the problem size. Two additional graph simplification techniques are introduced. During stitch candidate generation, the methods described in [16] are applied to search all stitch candidates for TPL. In the second stage, for each decomposition graph, each vertex is assigned one color. Before calling the SDP formulation, a fast color assignment trial is proposed to achieve better speed-up (see Sect. 2.3.2).

Stitch candidate generation is one important step in parsing a layout. Fang et al. [13] and Ghaida et al. [21] 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, which are labeled with numbers representing how many other features are projected onto them. If one segment is projected by fewer than two features, then a stitch can be introduced. Note that to reduce the problem size, we restrict the number of maximum stitch candidates on each feature.

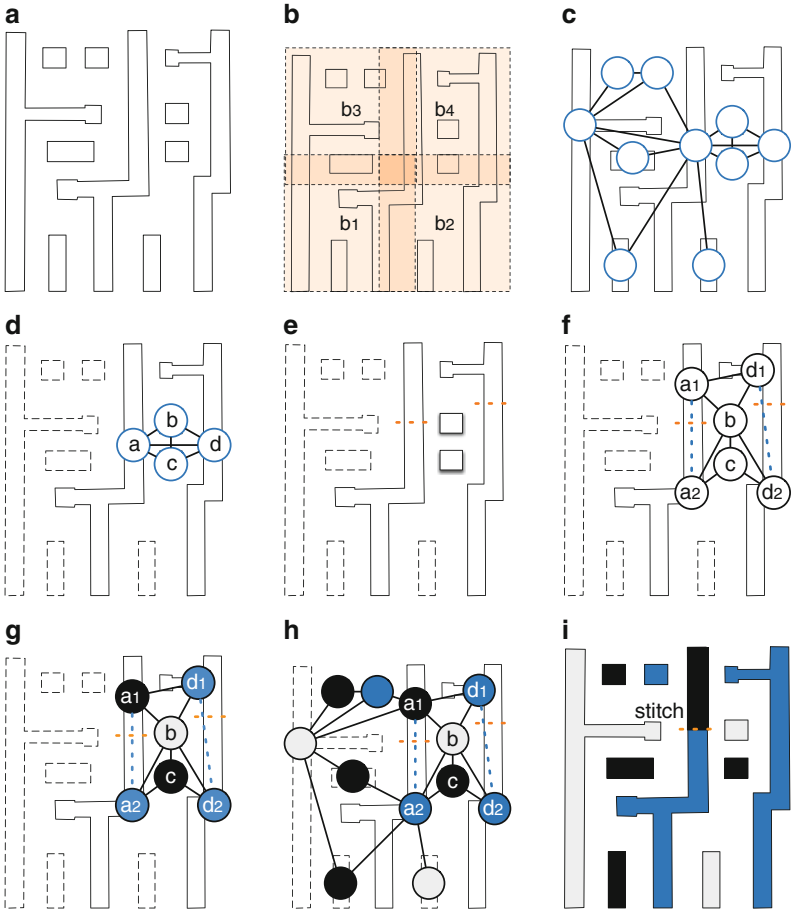

Fig. 2.21 An example of the layout decomposition flow

Figure 2.21 illustrates the decomposition process step by step. Given the input layout as in Fig. 2.21a, we partition it into a set of bins $\{b_1, b_2, b_3, b_4\}$ (see Fig. 2.21b). Then the layout graph is constructed (see Fig. 2.21c), 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 degrees are less than or equal to two are iteratively removed from the graph. The simplified layout graph, shown in Fig. 2.21d, only contains vertices a, b, c, and d. Figure 2.21d shows the results of the projection. After stitch candidate generation [16], there are two stitch candidates for TPL (see Fig. 2.21e). Based on the two stitch candidates, vertices a and d are each divided into two vertices. The constructed decomposition graph is given in Fig. 2.21f. It maintains all the information about conflict edges and stitch

Table 2.9 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

candidates, where the solid edges are the conflict edges while the dashed edges are the stitch edges, which function as stitch candidates. In each decomposition graph, a color assignment, which contains SDP formulation and partition based mapping, is carried out. During color assignment, the six vertices in the decomposition graph are divided into three groups: $\{a_1, c\}$, $\{b\}$ and $\{a_2, d_1, d_2\}$ (see Fig. 2.21g, h). Here one stitch on feature a is introduced. Figure 2.21i shows the final decomposed layout after iteratively recovering the removed vertices. Our last process merges the decomposed graphs. Since this example only has one decomposition graph, this process is skipped.

Density balance, especially local density balance, is seamlessly integrated 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. Then, we discuss density balance in all other steps.

Density Balanced SDP Algorithm

For each decomposition graph, density-balanced color assignment is carried out. Some notations used are listed in Table 2.9.

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

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

$$\text{s.t. } c_{ij} = (x_i == x_j) \qquad \forall e_{ij} \in CE \tag{2.9a}$$

$$s_{ij} = x_i \oplus x_j \qquad \forall e_{ij} \in SE \tag{2.9b}$$

$$x_i \in \{1, 2, 3\} \qquad \forall r_i \in V \tag{2.9c}$$

$$d_{kc} = \sum_{x_i = c} den_{ki} \qquad \forall r_i \in V,\ b_k \in B \tag{2.9d}$$

$$DU_k = \max\{d_{kc}\} / \min\{d_{kc}\} \qquad \forall b_k \in B \tag{2.9e}$$

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 number of conflicts and number of stitches, respectively. The constraint (2.9e) is nonlinear, which makes the program (2.9) hard to formulate into ILP as in [8]. Similar nonlinear constraints occur in the floorplanning problem [33], where Taylor expansion is used to linearize the constraint into ILP. However, Taylor expansion introduces the penalty of accuracy. Compared with the traditional time-consuming ILP, SDP has been shown to be a better approach in terms of runtime and solution quality tradeoffs [8]. However, how to integrate the density balance into the SDP formulation is still an open question. We will now show that instead of using painful Tayor expansions, 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 observation that maximizing DU_k^* is equivalent to minimizing DU_k.

Lemma 2.3. *$DU_k^* = 2/3 \cdot \sum_{i,j \in V} den_{ki} \cdot den_{kj} \cdot (1 - \vec{v}_i \cdot \vec{v}_j)$, 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

$$\sum_i \sum_j len_i \cdot len_j \cdot (1 - \vec{v}_i \cdot \vec{v}_j) = \sum_i \sum_j len_i \cdot len_j \cdot 3/2$$
$$= 3/2 \cdot \sum_i len_i \sum_j len_j = 3/2 \cdot d_1 \cdot d_2$$

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,

$$DU_2 = d_1 \cdot d_2 + d_1 \cdot d_3 + d_2 \cdot d_3$$
$$= 2/3 \cdot \sum_{i,j \in V} len_i \cdot len_j \cdot (1 - \vec{v}_i \cdot \vec{v}_j)$$

Because of Lemma 2.3, DU_k^* can be represented as a vector inner product. Thus, we have achieved the following theorem:

Theorem 2.3. *Maximizing DU_k^* achieves 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 the DU_k^* expression. Similarly, we can eliminate the constants in the calculation of the conflict and stitch numbers. The simplified vector expression is as follows:

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

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

$$\vec{v}_i \in \left\{ (1,0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) \right\} \tag{2.10b}$$

Formulation (2.10) is equivalent to the mathematical formulation (2.9), and its solution is also NP-hard. Constraint (2.10b) requires the 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 2.4.

Theorem 2.4. *Relaxing the vector expression (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 ith row and the jth 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 limitations, the detailed proof is omitted. The solution of (2.11) is continuous instead of discrete, and provides a lower bound of vector expression (2.10). In other words, (2.11) provides an approximate solution to (2.10).

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 the same color. If X_{ij} is close to 1, features r_i and r_j tend to belong to the same color (mask), whereas if it is close to -0.5, r_i and r_j tend to be in different colors (masks). With these

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 ≤ 3 **then**
11: return;
12: **else if** graph size ≤ 7 **then**
13: Backtracking based three-way partitioning;
14: **else**
15: FM based three-way partitioning;
16: **end if**

guidelines, we can adopt a mapping procedure to finally assign all input features into three colors (masks).

In [8], 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} values tend to be the same color. The X_{ij} values can be classified into two types: clear and vague. If most of the X_{ij}s in matrix X are clear (i.e., close to 1 or -0.5), this greedy method may achieve good results. However, if the decomposition graph is not 3-colorable, some values in matrix X are vague. For the vague X_{ij}s, 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 the decomposition graph. The information is captured through constructing a graph, denoted by G_M. Through formulating the mapping as a three-way partition of the graph G_M, our mapping can provide a global view to search for better solutions.

Algorithm 4 shows our partition-based mapping procedure. Given the solutions from program (2.11), all non-zero X_{ij} values are used to form triplets that are then sorted (lines 1–2). The mapping incorporates two stages to deal with the two different types of X_{ij}: clear and vague. The first stage (lines 3–8) is similar to that in [8]. 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 Union(i, j) is applied to merge them into a large vertex. Similarly, if $X_{ij} < th_{sp}$, then Separate(i, j) is used to label r_i and r_j as incompatible. In the second stage (lines 9–16) we deal with vague X_{ij} values. During the previous stage some vertices have been merged, therefore the total number of vertices is not large. Here we construct a graph G_M to

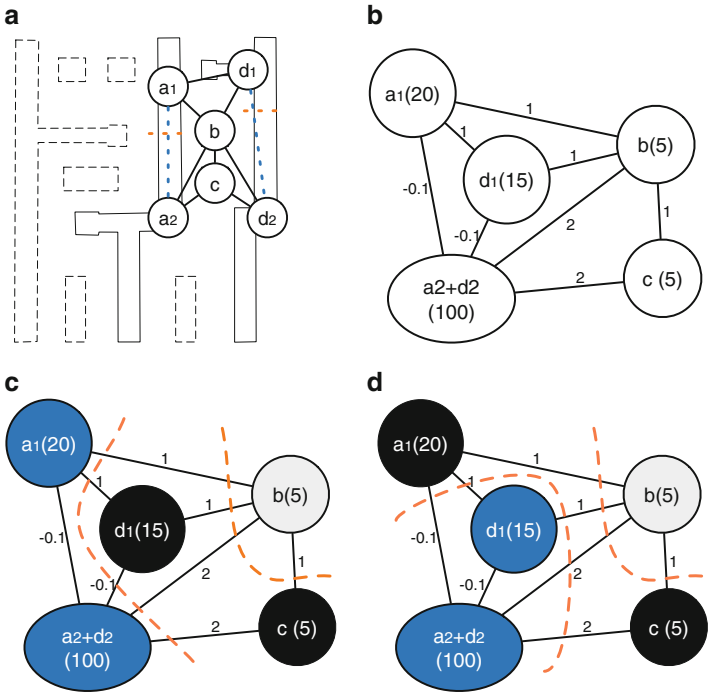

Fig. 2.22 Density balanced mapping. (**a**) Decomposition graph. (**b**) Construct graph G_M. (**c**) Mapping result with cut value 8.1 and density uniformities 24. (**d**) A better mapping with cut 8.1 and density uniformities 23

represent the relationships among all the remaining 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 restated 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 densities 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 more balanced and larger cut partitions.

An example of the density balanced mapping is shown in Fig. 2.22. Based on the decomposition graph (see Fig. 2.22a), 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. 2.22b, 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. 2.22c, d). However, their density uniformities are 24 and 23, respectively. To keep a more balanced resulting density, the second partitioning in Fig. 2.22c is adopted as color assignment result.

It is well known that the maximum-cut problem, even for a two-way partition, is NP-hard. However, we observe that in many cases, after the global SDP optimization, the graph size of G_M could be quite small, i.e., less than 7. For these small cases, we develop a backtracking based method to search the entire solution space. Note that here backtracking can quickly find the optimal solution even though three-way partitioning is NP-hard. If the graph size is larger, we propose a heuristic method, motivated by the classic FM partitioning algorithm [34, 35]. We make the following modifications to the classic FM algorithm: (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 number of vertices in G_M. The runtime of three-way maximum-cut partitioning algorithm is $O(m \log m)$. Note that the first stage of mapping needs $O(n^2 \log n)$ [8]. Since m is much smaller than n, the complexity of density balanced mapping is $O(n^2 \log n)$.

Density Balanced Graph Division

Here, we show that the layout graph simplification, which was proposed in [8], also takes local density balance into account. 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 remaining vertices, we iteratively recover all the removed vertices and assign them legal colors. Instead of randomly assigning colors, we search for legal colors that also improve the density uniformity.

Speed-up Techniques

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

- ICC [8, 13, 16];
- Vertex with Degree Less than 3 Removal [8, 13, 16];
- 2-Edge-Connected Component Computation [8, 13, 16];
- 2-Vertex-Connected Component Computation [13, 16].

Apart from the above graph simplifications, our decomposer proposes a set of novel speed-up techniques, which are introduced in the following.

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 [8]. During stitch candidate generation, forbidding cut vertices to be stitch candidates can be helpful for later decomposition graph simplification. Figure 2.23a shows a layout graph, where feature a is a cut vertex, since its

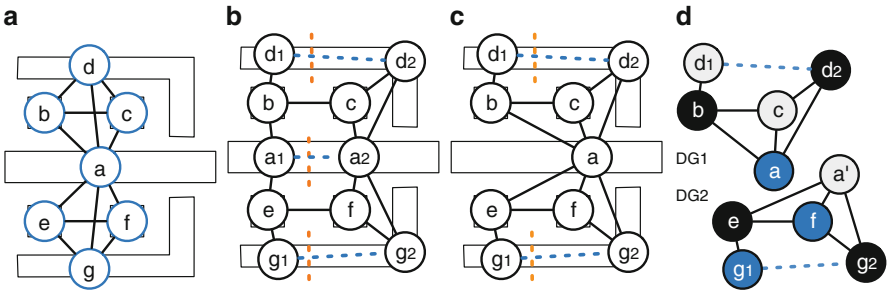

Fig. 2.23 Layout graph cut vertex stitch forbiddance

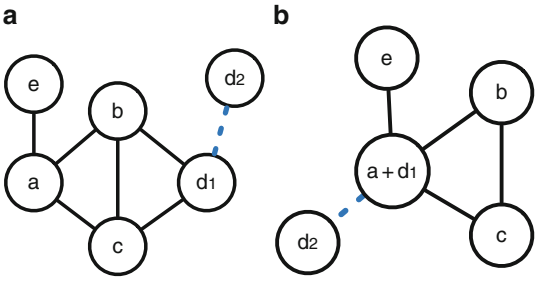

Fig. 2.24 DG vertex clustering to reduce the decomposition graph size

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. 2.23b, which is hard to simplify further. If we prohibit a from being a stitch candidate, the corresponding decomposition graph is shown in Fig. 2.23c, where a is still a cut vertex in the decomposition graph. Then, we can apply 2-connected component computation [13] to simplify the problem size, and apply color assignment separately (see Fig. 2.23d).

Our second technique, **Decomposition graph vertex clustering**, is a speed-up technique to further reduce the decomposition graph size. As shown in Fig. 2.24a, vertices a and d_1 share the same conflict relationships against b and c. Besides, there are no conflict edges between a and d_1. If no conflict is introduced, vertices a and d_1 should be assigned the same color, and we can cluster them together, as shown in Fig. 2.24b. Note that the stitch and conflict relationships are also merged. Applying vertex clustering in the 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 apply to all decomposition graphs. We therefore derive a fast color assignment trial before calling our SDP-based method. If no conflicts or stitches are introduced, our trial solves the color assignment problem in linear time. Note that SDP is skipped only when the decomposition graph can be colored without stitches or conflicts, so our fast trial does not degrade the quality of the solution. 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 ∃n ∈ 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 described by Algorithm 5 above. We first iteratively remove the vertices 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 a conflict or stitch.

2.3.3 Experimental Results

We implemented our decomposer in C++ and tested it on an Intel Xeon 3.0 GHz Linux machine with 32 GB RAM. ISCAS 85&89 benchmarks from [8] were used, where the minimum coloring spacing dis_m was set the same with previous studies [8, 13]. Additionally, to perform a comprehensive comparison, we also test on other two benchmark suites. The first suite is six dense benchmarks ("c9_total"–"s5_total"), while the second suite is two synthesized OpenSPARC T1 designs "mul_top" and "exu_ecc" with the Nangate 45 nm standard cell library [36]. 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 was set to 0.1. The size of each bin was set to $10 \cdot dis_m \times 10 \cdot dis_m$. We used CSDP [31] as the SDP solver.

In the first experiment, we compared our decomposer with the state-of-the-art layout decomposers, which are not balanced density aware [8, 13, 16]. We obtain the binary files from [8, 13]. Since currently we cannot obtain the binary for the decomposer in [16], we directly use the results listed in [16]. Here our decomposer is denoted as "**SDP + PM**," where "PM" denotes the partition-based mapping. The β is set as 0. In other words, SDP + PM only optimizes for the number of stitches and conflicts. Table 2.10 shows the comparison of the various decomposers in terms

Table 2.10 Comparison of runtime and performance

Circuit	ICCAD'11 [8]				DAC'12 [13]				DAC'13 [16][a]				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**

[a]The results of DAC'13 decomposition are from [16]

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

First, we compare SDP + PM with the decomposer in [8], which is based on SDP formulation as well. From Table 2.10 we can see that the new stitch candidate generation (see [16] for more details) and partition-based mapping can achieve better performance, reducing the cost by around 55 %. Besides, SDP + PM runs nearly $4\times$ faster due to several proposed speed-up techniques, including 2-vertex-connected component computation, layout graph cut vertex stitch forbiddance, decomposition graph vertex clustering, and fast color assignment trial. Second, we compare SDP + PM with the decomposer in [13], which applies several graph based simplifications and a maximum independent set (MIS) based heuristic. From Table 2.10 we can see that although the decomposer in [13] is faster, its MIS based heuristic produces solutions that are on average 33 % more costly compared to those produced by SDP + PM. Although SDP + PM is slower, it can reduce the cost of the solution by around 6 % compared with the decomposer in [16].

In addition, we compare SDP + PM with other two decomposers [8, 13] for some very dense layouts, as shown in Table 2.11. We can see that for some cases the decomposer in [8] cannot finish in 1000 s. Compared with the work in [13], SDP + PM can reduce solution cost by 65 %. It is observed that compared with other decomposers, SDP + PM demonstrates much better performance when the input layout is dense. When the input layout is dense, each independent problem may still be quite large after graph simplification, then our SDP-based approximation can achieve better results than heuristic. It can be observed that for the last three cases our decomposer could eliminate thousands of conflicts. Each conflict may require manual layout modification or high ECO efforts, which are very time consuming. Furthermore, even though our runtime is more than [13], it is still acceptable, not exceeding 6 min for the largest benchmark.

In the second experiment, we test our decomposer for density balance. We analyze edge placement error (EPE) using Calibre WORKbench [37] on an industry-strength setup. For analyzing the EPE in our test cases, we used systematic lithography process variation, such as using ranges of focus ± 50 nm and dose ± 5 %. In Table 2.12, we compare SDP + PM with "**SDP + PM + DB**," which is our density balanced decomposer. Here β is set as 0.04; testing found that bigger β do not help, and we want to give greater weight to conflict and stitch cost. Column "cost" also lists the weighted cost of conflict and stitch, i.e., cost = cn#$+0.1\times$st#.

From Table 2.12 we can see that by integrating density balance into our decomposition flow, our decomposer (SDP+PM+DB) can reduce the amount of EPE hotspots by 14 %. The 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 conflict and stitch cost.

Table 2.11 Comparison on very dense layouts

Circuit	ICCAD 2011 [8]				DAC 2012 [13]				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	13,853	6116.3	13.1	958	10,572	2015.2	254.5
s4_total	NA	NA	NA	>10,000	3868	13,632	5231.2	13	1151	11,091	2260.1	306
s5_total	NA	NA	NA	>10,000	4650	16,152	6265.2	12.9	1391	13,683	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**

Table 2.12 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**

Table 2.13 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**

We further compare the density balance, particularly EPE distributions for very dense layouts. As shown in Table 2.13, our density balanced decomposer (SDP+PM+DB) can reduce EPE distribution number by 7 % while maintaining a runtime only 4 % longer than plain SDP implementation for very dense layouts.

In addition, we demonstrate the scalability of our decomposer, especially the SDP formulation. Penrose benchmarks from [12] are used to explore the scalability of SDP runtime. No graph simplification is applied, therefore all runtime is consumed by solving SDP formulation. Figure 2.25 illustrates the relationship

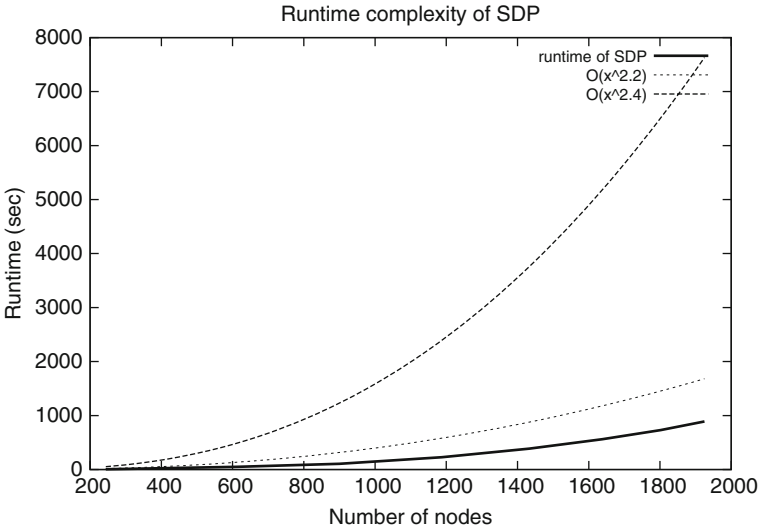

Fig. 2.25 Scalability of SDP formulation

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 $O(n^{2.2})$.

2.4 Summary

In this chapter we have proposed a set of algorithms to solve the TPLD problem. We have shown that this problem is NP-hard, thus the runtime required to solve it exactly increases dramatically with the problem size. Then we presented a set of graph division techniques to reduce the problem size. We proposed a general ILP formulation to simultaneously minimize the number of conflicts and stitches. We also proposed a novel vector program, and its SDP relaxation to improve scalability for very dense layouts. In addition, density balancing was integrated into all the key steps of our decomposition flow. Our decomposer performs better than current state-of-the-art frameworks in minimizing the cost of conflicts and stitches. As TPL may be adopted by industry for 14 nm/11 nm nodes, we believe more research will be needed to enable TPL-friendly design and mask synthesis.

References

1. Kahng, A.B., Park, C.-H., Xu, X., Yao, H.: Layout decomposition approaches for double patterning lithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**, 939–952 (2010)
2. Yuan, K., Yang, J.-S., Pan, D.Z.: Double patterning layout decomposition for simultaneous conflict and stitch minimization. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(2), 185–196 (2010)
3. Xu, Y., Chu, C.: GREMA: graph reduction based efficient mask assignment for double patterning technology. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 601–606 (2009)
4. Yang, J.-S., Lu, K., Cho, M., Yuan, K., Pan, D.Z.: A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 637–644 (2010)
5. Xu, Y., Chu, C.: A matching based decomposer for double patterning lithography. In: ACM International Symposium on Physical Design (ISPD), pp. 121–126 (2010)
6. Tang, X., Cho, M.: Optimal layout decomposition for double patterning technology. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 9–13 (2011)
7. Anton, V.O., Peter, N., Judy, H., Ronald, G., Robert, N.: Pattern split rules! a feasibility study of rule based pitch decomposition for double patterning. In: Proceedings of SPIE, vol. 6730 (2007)
8. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2011)
9. Yu, B., Xu, X., Gao, J.-R., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 349–356 (2013)
10. Ma, Q., Zhang, H., Wong, M.D.F.: Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology. In: ACM/IEEE Design Automation Conference (DAC), pp. 591–596 (2012)
11. Lin, Y.-H., Yu, B., Pan, D.Z., Li, Y.-L.: TRIAD: a triple patterning lithography aware detailed router. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 123–129 (2012)
12. Cork, C., Madre, J.-C., Barnes, L.: Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns. In: Proceedings of SPIE, vol. 7028 (2008)
13. Fang, S.-Y., Chen,W.-Y., Chang, Y.-W.: A novel layout decomposition algorithm for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 1185–1190 (2012)
14. Fang, S.-Y., Chang, Y.-W., Chen, W.-Y.: A novel layout decomposition algorithm for triple patterning lithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(3), 397–408 (2014)
15. Tian, H., Zhang, H., Ma, Q., Xiao, Z., Wong, M.D.F.: A polynomial time triple patterning algorithm for cell based row-structure layout. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 57–64 (2012)
16. Kuang, J., Young, E.F.: An efficient layout decomposition approach for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 69:1–69:6 (2013)
17. Tian, H., Du, Y., Zhang, H., Xiao, Z., Wong, M.D.F.: Constrained pattern assignment for standard cell based triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 178–185 (2013)
18. Zhang, Y., Luk, W.-S., Zhou, H., Yan, C., Zeng, X.: Layout decomposition with pairwise coloring for multiple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 170–177 (2013)

19. Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., Pan, D.Z.: A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 163–169 (2013)
20. Chen, Z., Yao, H., Cai, Y.: SUALD: spacing uniformity-aware layout decomposition in triple patterning lithography. In: IEEE International Symposium on Quality Electronic Design (ISQED), pp. 566–571 (2013)
21. Ghaida, R.S., Agarwal, K.B., Liebmann, L.W., Nassif, S.R., Gupta, P.: A novel methodology for triple/multiple-patterning layout decomposition. In: Proceedings of SPIE, vol. 8327 (2012)
22. Yuan, K., Pan, D.Z.: WISDOM: wire spreading enhanced decomposition of masks in double patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 32–38 (2010)
23. Michael, R.G., David, S.J.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
24. Kaufmann, M., Wagner, D.: Drawing Graphs: Methods and Models, vol. 2025. Springer, Berlin (2001)
25. Tamassia, R., Di Battista, G., Batini, C.: Automatic graph drawing and readability of diagrams. IEEE Trans. Syst. Man Cybern. Syst. **18**(1), 61–79 (1988)
26. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
27. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Rev. **38**(1), 49–95 (1996)
28. Cormen, T.T., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
29. Tarjan, R.E.: A note on finding the bridges of a graph. Inf. Process. Lett. **2**, 160–161 (1974)
30. Gurobi Optimization Inc.: Gurobi optimizer reference manual. http://www.gurobi.com (2014)
31. Borchers, B.: CSDP, a C library for semidefinite programming. Optim. Methods Softw. **11**, 613–623 (1999)
32. Lucas, K., Cork, C., Yu, B., Luk-Pat, G., Painter, B., Pan, D.Z.: Implications of triple patterning for 14 nm node design and patterning. In: Proceedings of SPIE, vol. 8327 (2012)
33. Chen, P., Kuh, E.S.: Floorplan sizing by linear programming approximation. In: ACM/IEEE Design Automation Conference (DAC), pp. 468–471 (2000)
34. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: ACM/IEEE Design Automation Conference (DAC), pp. 175–181 (1982)
35. Sanchis, L.A.: Multiple-way network partitioning. IEEE Trans. Comput. **38**, 62–81 (1989)
36. NanGate FreePDK45 Generic Open Cell Library. http://www.si2.org/openeda.si2.org/projects/nangatelib (2008)
37. Mentor Graphics.: Calibre verification user's manual (2008)

Chapter 3
Layout Decomposition for Other Patterning Techniques

3.1 Introduction

3.2 Layout Decomposition for Triple Patterning with End-Cutting

So far we have discussed the conventional process of TPL, called LELELE, which follows the same principle as litho-etch-litho-etch (LELE) type double patterning lithography (DPL). Here each "L" and "E" represents one lithography process and one etch process, respectively. Although LELELE has been widely studied by industry and academia, it still has two major issues. First, even with stitch insertion, there are some native conflicts in LELELE, like four-clique conflict [1]. For example, Fig. 3.1 illustrates a four-clique conflict among features *a*, *b*, *c*, and *d*. No matter how we assign the colors, there will be at least one conflict. Since this four-clique structure is common in advanced standard cell design, LELELE type TPL suffers from the native conflict problem. Second, compared to LELE type double patterning, there are more serious overlapping problems in LELELE [2].

To overcome the limitations of LELELE, Lin [3] recently proposed a new TPL manufacturing process called LELE-end-cutting (LELE-EC). As a TPL, this new manufacturing process contains three mask steps: first mask, second mask, and *trim* mask. Figure 3.2 illustrates an example of the LELE-EC process. To generate the target features in Fig. 3.2a, 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. 3.2b. Finally, a trim mask is applied to trim out the desired region as in Fig. 3.2c. 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 well suited to the LELE-EC process and can be decomposed thereby without introducing conflicts. In addition, if all cuts are properly designed or distributed, the LELE-EC process can introduce better printability than the conventional LELELE process [3].

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_3

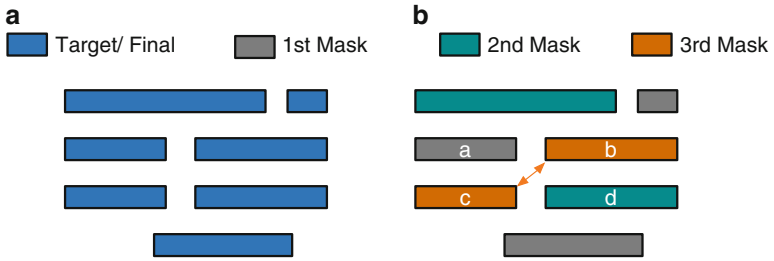

Fig. 3.1 Process of LELELE type triple patterning lithography. (**a**) Target features; (**b**) layout decomposition with one conflict introduced

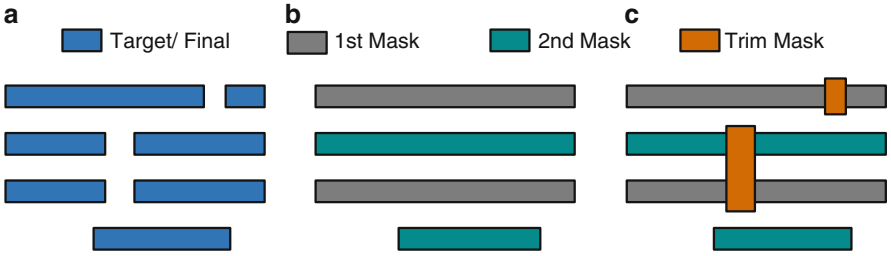

Fig. 3.2 Process of LELE-EC type triple patterning lithography. (**a**) Target features; (**b**) first and second mask patterns; (**c**) trim mask, and final decomposition without conflict

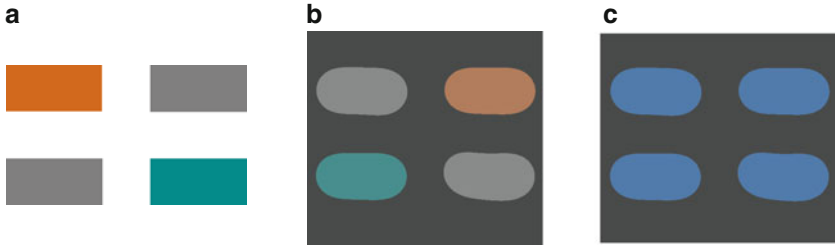

Fig. 3.3 LELELE process example. (**a**) Decomposed result; (**b**) simulated images for different masks; (**c**) combined simulated image as the final printed patterns

Figures 3.3 and 3.4 present simulated images of a design with four short features passed through the LELELE and LELE-EC processes, respectively. The lithography simulations are computed based on the partially coherent imaging system, where the 193 nm illumination source is modeled as a kernel matrix given by Banerjee et al. [4]. To model the photoresistance 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. 3.3c). Second, to reduce the round-off issues, as illustrated in Fig. 3.4b, in the LELE-EC process the short lines can be merged into longer lines, after which the trim mask is used to cut off some spaces. There may be some corner

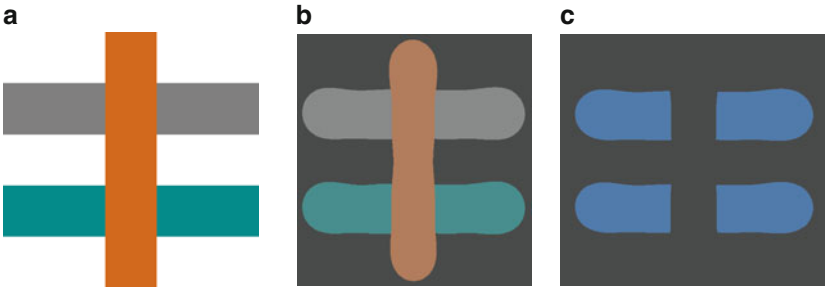

Fig. 3.4 LELE-EC process example. (**a**) Decomposed result; (**b**) simulated images for different masks, where *orange pattern* is trim mask; (**c**) combined simulated image as the final printed patterns

roundings due to the edge shorting of trim mask patterns; however, since the line shortening or rounding is a strong function of the line width [5], and we observe that trim mask patterns are usually much longer than the line-end width, we assume the rounding caused by the trim mask is insignificant. This assumption is demonstrated as in Fig. 3.4c.

Extensive research has been carried out in order to solve the corresponding design problems for LELELE-type TPL [1, 6–12]. Additionally, the related constraints have been considered in early physical design stages, like routing [13, 14], standard cell design [15, 16], and detailed placement [16]. However, only few attempts have been made to address the LELE-EC layout decomposition problem. Although introducing a trim mask improves printability, it introduces significant design challenges, especially in the layout decomposition stage. In this section, we propose a comprehensive study of LELE-EC layout decomposition. Given a layout specified by features in polygonal shapes, we extract the geometric relationships between the shapes and construct conflict graphs. The conflict graphs also model the compatibility of all end-cut candidates. Based on the conflict graphs, we use integer linear programming (ILP) to assign each vertex into one layer. Our goal in the layout decomposition is to minimize the number of conflicts, while also minimizing the amount of overlapping errors.

3.2.1 Preliminaries and Problem Formulation

Layout Graph

Given a layout which is specified by features in polygonal shapes, layout graph [1] is constructed. As shown in Fig. 3.5, 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. An edge is in CE if and only if the two corresponding features are within minimum coloring distance dis_m of each other. In other words, each edge in CE is a

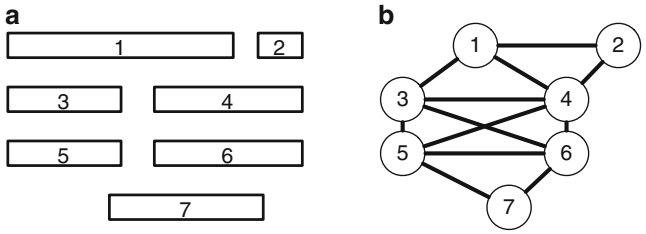

Fig. 3.5 Layout graph construction. (**a**) Input layout; (**b**) layout graph with conflict edges

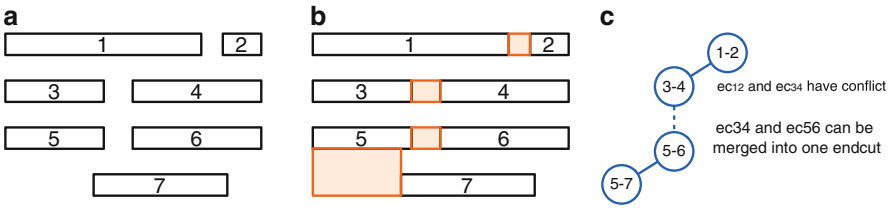

Fig. 3.6 End-cut graph construction. (**a**) Input layout; (**b**) generated end-cut candidates; (**c**) end-cut graph

conflict candidate. Figure 3.5a shows one input layout, and the corresponding layout graph is in Fig. 3.5b. Here the vertex set is $V = \{1, 2, 3, 4, 5, 6, 7\}$, while the conflict edge set is $CE = \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7)\}$. For each conflict candidate edge, 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 are merged into one feature. Then, the corresponding conflict edge can be removed. If stitches are considered during layout decomposition, some vertices in the 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 [17] for the details of stitch candidate generation.

End-Cut Graph

Since all the end-cuts are manufactured through a single exposure process, they should be distributed far away from each other. That is, two end-cuts conflict if the distance between them is smaller than the minimum end-cut distance dis_c. These conflict relationships among end-cuts are not available in layout graph, so we construct an *end-cut graph* to store the relationships. Figure 3.6a gives an example input layout, with all end-cut candidates pointed out in Fig. 3.6b. The corresponding end-cut graph is shown in Fig. 3.6c. Each vertex in the end-cut graph represents one end-cut. There is a solid edge if and only if the two end-cuts conflict with each other. There is a dash edge if and only if they are close to each other, and they can be merged into one larger end-cut.

Problem Formulation

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

Problem 3.1 (LELE-EC Layout Decomposition). Given a layout is specified by features in polygonal shapes, we can construct the layout and end-cut graphs. The LELE-EC layout decomposition assigns all vertices in the layout graph one of two colors and selects a set of end-cuts in the end-cut graph. The objective is to minimize the number of conflicts and stitches.

With the end-cut candidate set, LELE-EC layout decomposition is more complicated due to the additional constraints. Even if there are no end-cut candidates, LELE-EC layout decomposition is similar to the LELE type DPL layout decomposition. Sun et al. [18] showed that achieving the minimum number of conflicts and stitches with LELE layout decomposition is NP-hard, so it is not hard to see that LELE-EC layout decomposition is NP-hard as well. An NP-hard problem is a set of computational search problems that are difficult to solve [19]. No NP-hard problem can be solved in polynomial time in the worst case under the assumption that $P \neq NP$.

3.2.2 Algorithms

The overall flow of our layout decomposer is illustrated in Fig. 3.7. First we generate all end-cut candidates to find 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, modeling the LELE-EC layout decomposition as a coloring problem in the layout graph and an end-cut selection problem in the end-cut graph. Both the

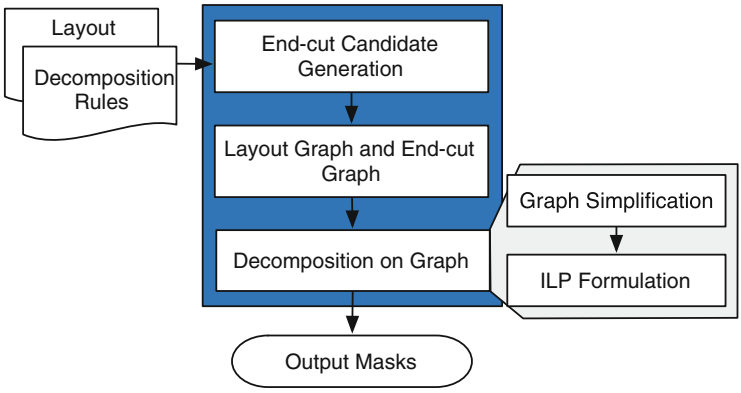

Fig. 3.7 Overall flow of our layout decomposer

coloring problem and the end-cut selection problem can be solved using one ILP formulation. Since the ILP formulation may suffer from excessive runtime in some cases, we propose several graph simplification techniques. To further reduce the problem size, we propose a step in which some end-cut candidates are pre-selected before ILP formulation. All the steps in the flow are detailed in the following sections.

End-Cut Candidate Generation

In this section, we will explain the details of our algorithm which generates all end-cut candidates. An end-cut candidate is generated between two conflicting polygonal shapes. It should be stressed that compared to the end-cut generation in [20], our methodology differs in two ways:

- 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. 3.8. We propose a shape-edge dependent algorithm to generate the end-cuts with multiple end-cut boxes.
- We consider the overlaps and variations caused by end-cuts. Two lithography simulations are illustrated in Figs. 3.9 and 3.10, respectively. In Fig. 3.9 we find some bad patterns or hotspots, due to the cuts between two long edges. In Fig. 3.10 we can see that the final patterns are in better shape. Therefore, to reduce the amount of manufacturing hotspots from trim mask, we avoid generating cuts along two long edges during end-cut candidate generation.

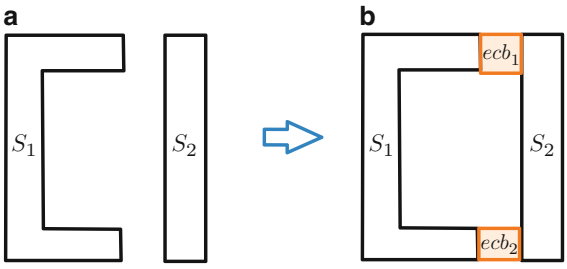

Fig. 3.8 An end-cut can have multiple end-cut boxes

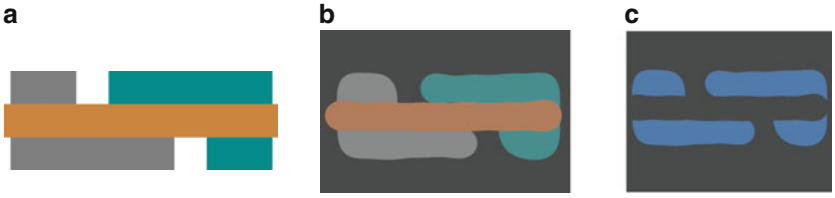

Fig. 3.9 (**a**) Decomposition example where cuts are along long edges; (**b**) simulated images for different masks; (**c**) combined simulated image with some hotspots

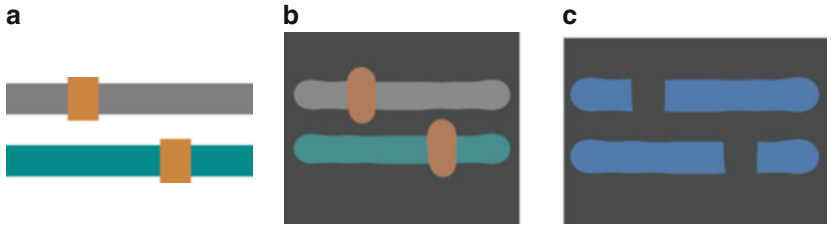

Fig. 3.10 (**a**) Decomposition example where cuts are between line ends; (**b**) simulated images for different masks; (**c**) combined simulated image with good printability

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

```
1:  Procedure generateEndCut (S1, S2);
2:  for all se1 ∈ edges(S1) do
3:      for all se2 ∈ edges(S2) do
4:          ecBox = generateEndCutBox (se1, se2);
5:          if ecBox ≠ 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 ∈ IC do
16:     Remove corner-corner end-cut boxes overlapping with edge-edge end-cut box;
17:     if ∃ set of type2 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
```

Algorithm 6 presents the key steps of generating an end-cut between two polygonal shapes S_1 and S_2. A polygonal shape consists of multiple edges. For each shape-edge pair, one taken from S_1 and another from S_2, the possibility of generating an end-cut box (*ecBox*) is considered 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 3.11 shows how end-cut boxes are generated between two shape-edges under different situations. In Fig. 3.11a, 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. In Fig. 3.11b the shape

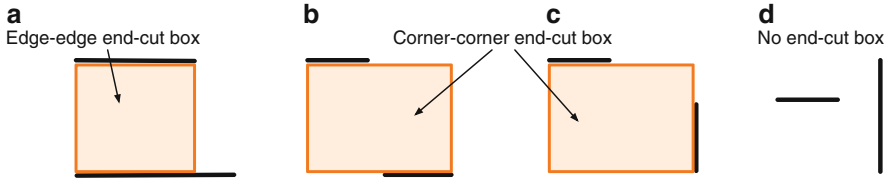

Fig. 3.11 End-cut box generation between any two shape-edges

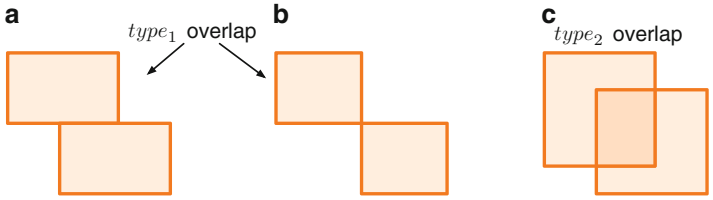

Fig. 3.12 Types of overlaps between end-cut boxes

edges are in same direction but they do not overlap, and in Fig. 3.11c, 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. 3.11d. In addition, end-cut boxes are not generated for the following cases: (1) the end-cut box is overlapping with any existing polygonal shape in the layout, (2) 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 $type_1$ 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, whereas the overlap in space is termed $type_2$ overlap as shown in Fig. 3.12. 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 we generate all of them (Lines 11–14).

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. 3.13. For $type_1$ overlaps in each ic, all end-cut boxes are generated (Line 20).

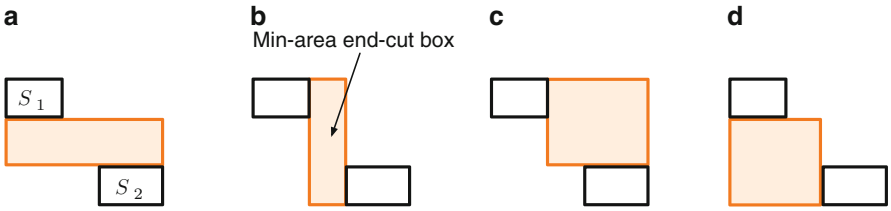

Fig. 3.13 Minimum area end-cut box is chosen for $type_2$ overlaps

Table 3.1 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 ith 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

ILP Formulations

After the construction of the layout and end-cut graphs, the LELE-EC layout decomposition problem can be reformulated as a coloring problem on the layout graph and a selection problem on the end-cut graph. At first glance, the coloring problem is similar to that in LELE layout decomposition. However, since the conflict graph cannot be guaranteed to be planar, the face graph-based methodology [21] cannot be applied here. Therefore, we use an ILP formulation to solve both the coloring and selection problems simultaneously. For convenience, some notations in the ILP formulation are listed in Table 3.1.

First, we discuss the ILP formulation when no stitch candidates are generated in layout graph. Given a set of input layout features $\{r_1, \ldots, r_n\}$, we construct the layout and end-cut graphs. Every conflict edge e_{ij} is in CE, while every 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 number of conflicts, our objective function is to minimize $\sum_{e_{ij} \in CE} c_{ij}$.

To evaluate the number of conflicts, we provide the following constraints:

$$\begin{cases} x_i + x_j \leq 1 + c_{ij} + ec_{ij} & \text{if } \exists\, ec_{ij} \in EE \\ (1 - x_i) + (1 - x_j) \leq 1 + c_{ij} + ec_{ij} & \text{if } \exists\, ec_{ij} \in EE \\ x_i + x_j \leq 1 + c_{ij} & \text{if } \nexists\, ec_{ij} \in EE \\ (1 - x_i) + (1 - x_j) \leq 1 + c_{ij} & \text{if } \nexists\, ec_{ij} \in EE \end{cases} \tag{3.1}$$

Here, ec_{ij} is a binary variable representing each end-cut candidate. If there is no end-cut candidate between adjacent features r_i and r_j and $x_i \neq x_j$, 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 in conflict with each other, at most one of them will be applied. To enable this, we introduce the following constraint.

$$ec_{ij} + ec_{pq} \leq 1, \quad \forall e_{ijpq} \in EE \tag{3.2}$$

To prevent useless end-cuts, we introduce the following constraints. If features x_i and x_j are in different colors, then $ec_{ij} = 0$.

$$\begin{cases} ec_{ij} + x_i - x_j \leq 1 \ \forall e_{ij} \in CE \\ ec_{ij} + x_j - x_i \leq 1 \ \forall e_{ij} \in CE \end{cases} \tag{3.3}$$

Without the stitch candidates, the LELE-EC layout decomposition can be formulated as shown in Eq. (3.4).

$$\begin{aligned} &\min \sum_{e_{ij} \in CE} c_{ij} \\ &\text{s.t. } (3.1), (3.2), (3.3) \end{aligned} \tag{3.4}$$

If the stitch insertion is considered, the ILP formulation is as in Eq. (3.5). Here, the objective is to simultaneously minimize the number of both conflicts and stitches. The parameter α is a user-defined parameter for assigning relative importance between the number of conflicts and stitches. The constraints (3.5a)–(3.5b) are used to calculate the number of stitches.

$$\min \sum_{e_{ij} \in CE} c_{ij} + \alpha \times \sum_{e_{ij} \in SE} s_{ij} \tag{3.5}$$

$$\text{s.t. } x_i - x_j \leq s_{ij} \qquad \forall e_{ij} \in SE \tag{3.5a}$$

$$x_j - x_i \leq s_{ij} \qquad \forall e_{ij} \in SE \tag{3.5b}$$

$$(3.1), (3.2), (3.3)$$

Graph Simplification Techniques

ILP is a classical NP-hard problem, i.e., there is no polynomial time optimal algorithm to solve it [19]. Therefore, for large layout cases, solving ILP may suffer a heavy runtime penalty in order to achieve the results. In this section, we provide a set of speed-up techniques. Note that these techniques maintain optimality. In other

words, with these speed-up techniques, ILP formulation can achieve comparable results to formulations without them.

The first speed-up 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., [17, 22, 23]).

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 for all bridge edges in the 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 number of vertices and $|E|$ is the number of edges in the layout graph.

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

3.2.3 Experimental Results

We implemented our algorithms in C++ and tested on an Intel Xeon 3.0 GHz Linux machine with 32GB RAM. Fifteen benchmark circuits from [1] are used. GUROBI [24] 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 [1, 7]. The width threshold wth, 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 3.2 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#" report the total feature number and the divided component number, respectively. For each method, we report the number of conflicts, the number of stitches, and the computational time in seconds("CPU(s)"). "Cost" is the cost function, set to conflict# $+0.1\times$ stitch#.

From Table 3.2, 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 increased by 5 %. It should be noted that stitch insertion has been shown to be an effective method to reduce the costs for both LELE and LELELE layout decompositions. However, we can see that for LELE-EC layout

Table 3.2 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	10,189	827	2	0	2	14.57	2	0	2	15.98
C8	14,603	1154	2	0	2	21.2	2	0	2	23.07
C9	14,575	2325	23	0	23	24.36	12	12	13.2	28.06
C10	21,253	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	67,696	5116	166	0	166	179.05	166	1	166.1	218.37
S3	157,455	15,176	543	0	543	506.55	530	13	531.3	563.65
S4	168,319	15,354	443	0	443	464.84	436	7	436.7	494.4
S5	159,952	12,626	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

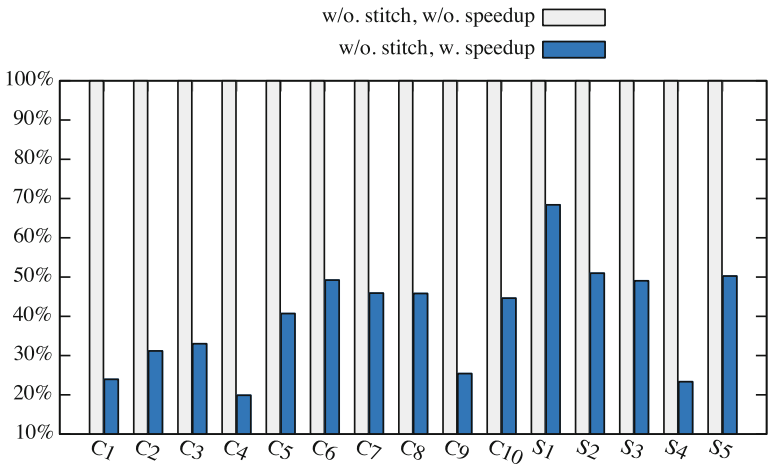

Fig. 3.14 Effectiveness of speed-up techniques when no stitch is introduced

decomposition, stitch insertion is not very effective. When also considering the overlap variability due to the stitches, 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. Figure 3.14 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. 3.14 we can see that with speed-up techniques (bridge computation and end-cut pre-selection), the runtime can be reduced by around 60 %.

Figure 3.15 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 formulations, the bridge computation and the end-cut pre-selection can reduce runtime by around 56 %.

Figure 3.16 shows four conflict examples in decomposed layout, where conflict pairs are labeled with red arrows. We can observe that some conflicts (see Fig. 3.16a, 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. 3.16b, 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.

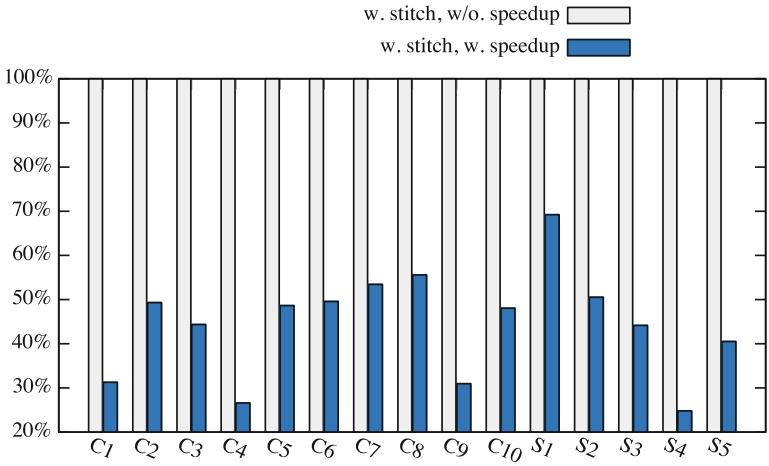

Fig. 3.15 Effectiveness of speed-up techniques when stitch is introduced

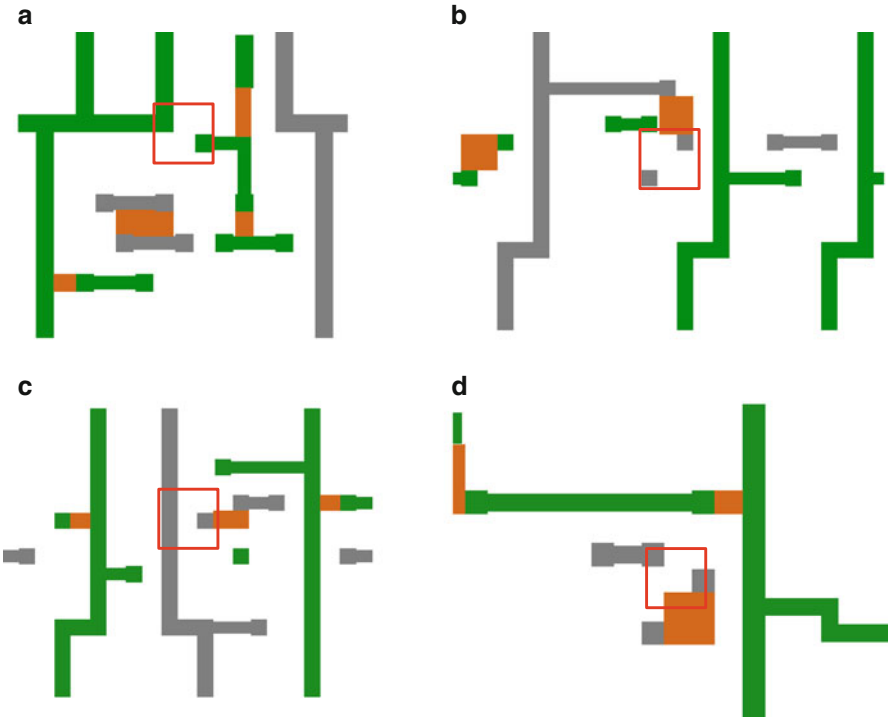

Fig. 3.16 Conflict examples in decomposed results. (**a**), (**c**) Conflicts because no additional end-cut can be inserted due to the existing neighboring end-cuts. (**b**), (**d**) Conflicts because no end-cut candidates between irregular vias

3.3 Layout Decomposition for Quadruple Patterning and Beyond

Quadruple patterning lithography (QPL) is a natural extension along the paradigm of double/triple patterning. In QPL manufacturing, four exposure/etching processes produce the initial layout. This represents an additional mask compared to triple patterning lithography, increasing the number of processing steps by 33 %. There are, however, several reasons to use QPL. First, due to the delay of other lithography techniques, such as EUV, the semiconductor industry needs to prepare CAD tools and already understands the complexity and implications of QPL. Even from a theoretical perspective, studying general multiple patterning is valuable. Second, it is observed that for triple patterning lithography, even with stitch insertion, there are several common native conflict patterns. As shown in Fig. 3.17a, contact layout within the standard cell may generate some four-clique patterns, which are indecomposable. This conflict can be easily resolved if four masks are available (see Fig. 3.17b). Third, with one more mask, some stitches may be avoided during manufacturing. The overlapping and yield issues derived from the stitches can also potentially be resolved.

The process of QPL brings up several critical design challenges, one of which is layout decomposition, where the original layout is divided into four masks (colors). Effectively solving the quadruple patterning (and general multiple patterning) problem remains an open question. In this section, we deal with this quadruple patterning layout decomposition (QPLD) problem.

Our contributions are highlighted as follows:

1. To the best of our best knowledge, this is the first layout decomposition research on the QPLD problem. We believe this work will invite further research into this field, thereby promoting the scaling of the technology node.
2. Our framework consists of holistic algorithmic processes, including a semidefinite programming-based algorithm, linear color assignment, and a novel GH tree-based graph division.

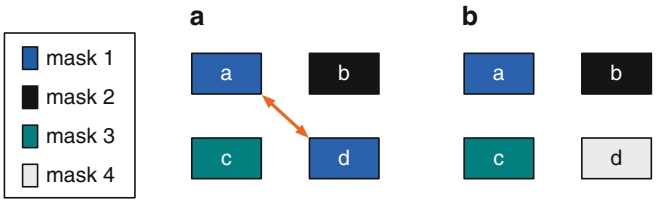

Fig. 3.17 (**a**) A common native conflict from triple patterning lithography; (**b**) the conflict can be resolved through quadruple patterning lithography

3. We demonstrate the viability of our algorithm to suit general K-patterning (K≥4) layout decomposition, which could act as advanced guidelines for future technology.

3.3.1 Problem Formulation

Given an input layout specified by features in polygonal shapes, a *decomposition graph* [1, 22] is constructed according to Definition 2.2.

Now we formally state the problem of QPLD.

Problem 3.2 (QPLD). Given an input layout specified by features in polygonal shapes and minimum coloring distance min_s, the decomposition graph can be constructed. QPLD assigns all the vertices one of four colors (masks) to minimize the number of conflicts and stitches.

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

Problem 3.3 (K-Patterning Layout Decomposition). Given an input layout, construct the decomposition graph. Each vertex in graph is assigned one of K colors (masks) to minimize the number of conflicts and stitches.

3.3.2 Algorithms

The overall flow of our layout decomposition is summarized in Fig. 3.18. We first construct a decomposition graph to transform the original geometric patterns into a graph model. By this way, the QPLD problem can be formulated as four-coloring

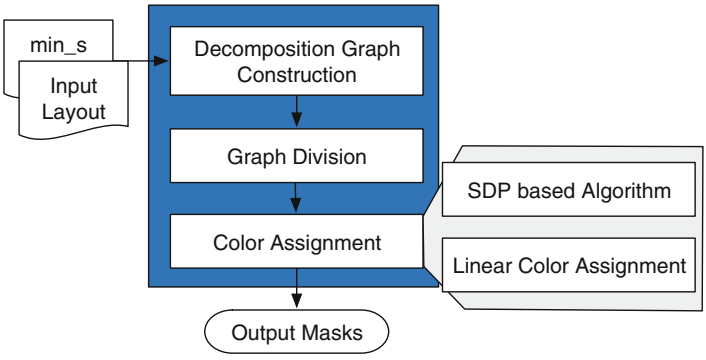

Fig. 3.18 Proposed layout decomposition flow

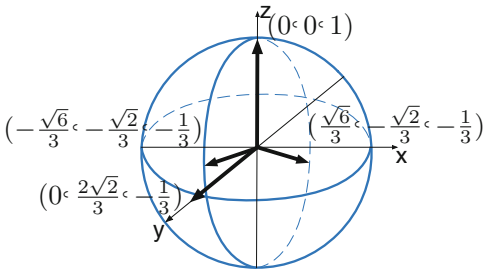

Fig. 3.19 Four vectors correspond to four colors

on the decomposition graph. To reduce the problem size, graph division techniques (see Sect. 3.3.2) 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 number of conflicts and the number of stitches. In the following section, we propose two color assignment algorithms: a semidefinite programming (SDP) based algorithm, and linear color assignment.

SDP Based Color Assignment

SDP has been successfully applied to the triple patterning layout decomposition problem [1, 10]. We will now show that SDP formulation can be extended to solve QPLD problem. To represent four different colors (masks), as illustrated in Fig. 3.19, we use four unit vectors [25]: $(0, 0, 1)$, $(0, \frac{2\sqrt{2}}{3}, -\frac{1}{3})$, $(\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$ and $(-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3})$. We construct the vectors in such a way that the inner product for any two vectors $\vec{v}_i, \vec{v}_j$ satisfies the following invariant: $\vec{v}_i \cdot \vec{v}_j = 1$ if $\vec{v}_i = \vec{v}_j$; $\vec{v}_i \cdot \vec{v}_j = -\frac{1}{3}$ if $\vec{v}_i \neq \vec{v}_j$.

Using the above vector definitions, the QPLD problem can be reformulated in the following manner:

$$\min \sum_{e_{ij} \in CE} \frac{3}{4}\left(\vec{v}_i \cdot \vec{v}_j + \frac{1}{3}\right) + \frac{3\alpha}{4} \cdot \sum_{e_{ij} \in SE} (1 - \vec{v}_i \cdot \vec{v}_j) \tag{3.6}$$

$$\text{s.t. } \vec{v}_i \in \left\{(0,0,1), \left(0, \frac{2\sqrt{2}}{3}, -\frac{1}{3}\right), \left(\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3}\right), \left(-\frac{\sqrt{6}}{3}, -\frac{\sqrt{2}}{3}, -\frac{1}{3}\right)\right\}$$

where the objective function minimizes the number of conflicts and stitches. α is a user-defined parameter, which is set as 0.1 in this work. After relaxing the discrete constraints in (3.6) and removing the constants from the objective function, we obtain the following SDP formulation.

Algorithm 7 SDP + Backtrack

Require: SDP solution x_{ij}, threshold value tth;
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**

$$\min \sum_{e_{ij} \in CE} \vec{v}_i \cdot \vec{v}_j - \alpha \sum_{e_{ij} \in SE} \vec{v}_i \cdot \vec{v}_j \tag{3.7}$$

$$\text{s.t. } \vec{v}_i \cdot \vec{v}_i = 1, \quad \forall i \in V$$

$$\vec{v}_i \cdot \vec{v}_j \geq -\frac{1}{3}, \quad \forall e_{ij} \in CE$$

After solving the SDP, we get a set of continuous solutions in a 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 tend to be in the same mask (color). A greedy mapping algorithm [1] can then be used to produce a color assignment, but its performance may not be ideal.

To overcome the limitations of the greedy method, we propose a backtrack based algorithm (see Algorithm 7) that considers both SDP results and graph information for use in our framework. The backtrack-based method accepts two arguments: the SDP solution $\{x_{ij}\}$ and a threshold value tth. In our work tth is set to 0.9. As stated before, 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). This reduces the number of vertices and thus simplifies the graph (line 4). The simplified graph is called a *merged graph* [10]. On the merged graph, the *BACKTRACK* algorithm searches for an optimal color assignment (lines 7–19).

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**

Linear Color Assignment

The backtrack-based method may still incur runtime overhead, especially for complex cases where the SDP solution cannot provide enough merging candidates. Therefore, a more efficient method of color assignment is required. One might think that the color assignment for quadruple patterning can be solved through the *four color map theorem* [26], which states that every planar graph is four-colorable. However, in emerging technology nodes, the designs are so complex that we observe many K_5 or $K_{3,3}$ structures, where K_5 is the complete graph with five vertices and $K_{3,3}$ is the complete bipartite graph with six vertices. *Kuratowski's theorem* [27] also states that the decomposition graph is not planar, so it is difficult to apply the classical four coloring technique [28].

Here we propose an efficient color assignment algorithm that targets not only planar graphs, but also general graphs. Our algorithm also runs in linear time, an improvement over the classical four coloring method [28], which runs in quadratic time.

Our linear color assignment algorithm, summarized in Algorithm 8, involves three stages. The first stage is iterative vertex removal. For each vertex v_i, we denote its conflict degree (the number of incident conflict edges) as $d_{conf}(v_i)$ and its stitch degree (the number of stitch edges) as $d_{stit}(v_i)$. The main idea is to identify the vertices with conflict degree less than 4 and stitch degree less than 2 as non-critical. Thus, they can be temporarily removed and pushed onto a stack S (lines 1–4). After coloring the remaining (critical) vertices, each vertex in the stack S is popped one at a time and assigned a legal color (lines 11–15). This strategy is safe in terms of the number of conflicts. In other words, when a vertex is popped from S, there will always be an available color that will not introduce any conflicts.

In the second stage (lines 5–9), all remaining (critical) vertices should be assigned colors one by one. However, color assignment in one specific order may be

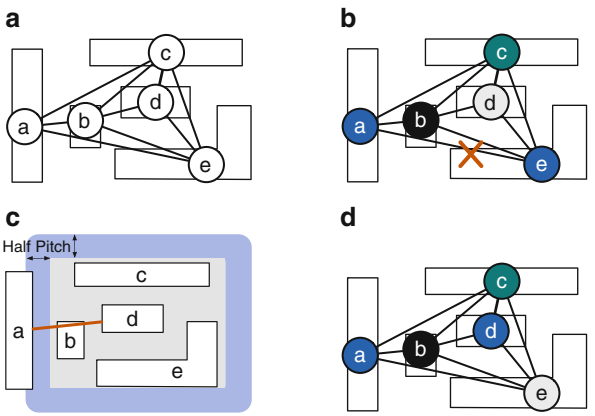

Fig. 3.20 (**a**) Decomposition graph; (**b**) greedy coloring with one conflict; (**c**) *a* is detected as color-friendly to *d*; (**d**) coloring considering color-friendly rules

stuck at a *local optimum*, which stems from the greedy nature. For example, given a decomposition graph in Fig. 3.20a, if the coloring order is *a*-*b*-*c*-*d*-*e*, when vertex *d* is the greedily selected grey color, the following vertex *e* cannot be assigned any color without conflict (see Fig. 3.20b). In other words, vertex ordering significantly impacts the coloring result.

To alleviate the impact of vertex ordering, we propose two strategies. The first strategy is called *color-friendly rules*, as in Definition 3.1. In Fig. 3.20c, all conflict neighbors of pattern *d* are labeled inside a grey box. Since the distance between *a* and *d* is within the range $(min_s, min_s + hp)$, *a* is color-friendly to *d*. Interestingly, we discover a rule that for a complex/dense layout, color-friendly patterns tend to be of 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 conflict neighbor detection; thus, it can be finished during decomposition graph construction without much additional effort.

Definition 3.1 (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, and min_s is the minimum coloring distance.

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

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.

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** $c(v_i), \forall v_i \in vec$;

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. 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 requires $O(n \log n)$ time, we only store into vector vec $vec[1]$, $vec[2]$, and $vec[3]$, successively. The vector vec[i], $1 \leq i \leq 3$, contains vertices with special conflict degree, defined as follows:

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

The key thing to note is that the vertices in vec are nearly conflict degree descending order, but construction takes linear time.

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

is assigned a color, it is labeled as SOLVED. In the third round of scanning, we greedily assign colors to any remaining UNSOLVED vertices.

For a decomposition graph with color-friendly information and n vertices, the first stage's vertex removal and stack operations take $O(n)$ time. At the second stage, as mentioned above, the coloring requires $O(n)$ time. In the third "post-refinement" stage, all vertices are traveled once, which is also $O(n)$ time. Therefore, the total complexity is $O(n)$.

GH-Tree Based 3-Cut Removal

Graph division is a technique that partitions the whole decomposition graph into a set of components, for which color assignment can be solved independently for each component. In our framework, the techniques extended from previous work are summarized as follows: (1) Independent Component Computation [1, 7–10, 21, 22, 29]; (2) Vertex with Degree Less than 3 Removal [1, 7, 9, 10][1]; (3) 2-Vertex-Connected Component Computation [7, 9, 10].

Another technique, *cut removal*, has been proven powerful in double patterning layout decomposition [1, 7, 29]. A cut of a graph is an edge whose removal partitions the graph into two components. The definition of cut can be extended to 2-cut (3-cut), which is a pair (triplet) of edges whose removal would disconnect the graph. However, while 1-cut and 2-cut detection can be finished in linear time [7], 3-cut detection is much more complicated. Here we propose an effective 3-cut detection method, which can be easily extended to detect any K-cut (K $\geq$ 3).

Figure 3.21a shows a graph with a 3-cut $(a - d, b - e, c - f)$, and two components can be derived by removing this 3-cut. After color assignment the 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 for one component. For a 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 $(c(v) + i)$. 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 itself. An example of such a color rotation operation is illustrated in Fig. 3.21b, c, where conflict between vertices c, f would need to be removed to interconnect two components together. Here all the vertices in component 2 are rotated by 1 (see Fig. 3.21c). We have the following Lemma:

Lemma 3.1. *In the QPLD problem, color rotation after interconnecting 3-cut does not increase the number of conflicts.*

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 from the edges. Since all vertices are 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

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

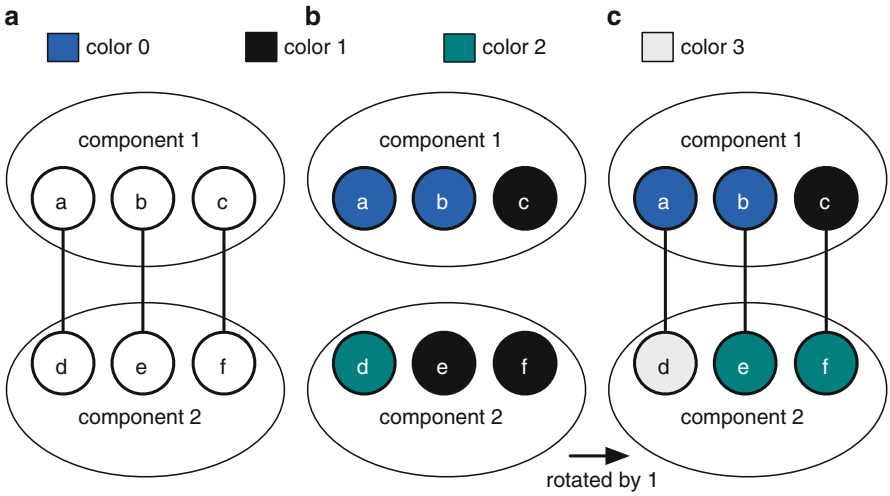

Fig. 3.21 An example of 3-cut detection and removal

that would cause one conflict. At most three options are infeasible on the 3-cut. Therefore, there will always remain an option that does not introduce any conflict on the 3-cut.

To detect all 3-cuts, we have the following Lemma:

Lemma 3.2. *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 when divided by a 3-cut.*

Based on Lemma 3.2, 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 the cut pairs may take prohibitively long and be impractical for complex layout designs.

Gomory and Hu [30] showed that the cut values between all 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. 3.22a, the corresponding GH-tree is shown in Fig. 3.22b, where the value on edge e_{ij} is the minimum cut number between vertices v_i and v_j. Because of Lemma 3.2, to divide the graph through 3-cut removal, all the edges with value less than 4 would be removed. The final three components are shown in Fig. 3.22c.

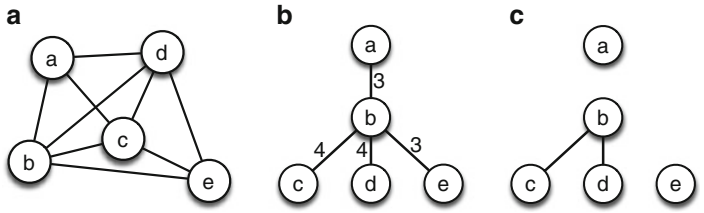

Fig. 3.22 (**a**) Decomposition graph; (**b**) corresponding GH-tree; (**c**) components after 3-cut removal

Algorithm 10 GH-Tree Based 3-Cut Removal

Require: Decomposition graph $G = \{V, CE, SE\}$;
1: Construct GH-tree as in [31];
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. We first construct a GH-tree based on the algorithm by Gusfield [31] (line 1). Dinic's blocking flow algorithm [32] is applied to aid in GH-tree construction. All edges in the GH-tree with weights less than four are then removed (line 2). After solving the connected components problem (line 3), we can assign colors to each component separately (lines 4–5). Finally, color rotation is applied to interconnect all 3-cuts back (line 6).

3.3.3 Experimental Results

We implemented the proposed layout decomposition algorithms in C++ and tested on a Linux machine with a 2.9 GHz CPU. We choose GUROBI [24] as the ILP solver and CSDP [33] as the SDP solver. The benchmarks in [1, 7] are used as our test cases. We scale down the Metal1 layer to 20 nm half pitch. The minimum feature width w_m is set to 20 nm. The minimum spacing s_m between two features is set to 20. From Fig. 3.23 we can see that when the minimum coloring distance $min_s = 2{\cdot}s_m{+}w_m = 60$ nm, even one dimension regular patterns can be K_5 structures, which are not four-colorable or planar [27]. 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. For larger min_s, there are too many native conflicts in layouts, as the benchmarks are not multiple patterning friendly.

We first compare different color assignment algorithms for quadruple patterning, and the results are listed in Table 3.3. "**ILP**," "**SDP+Backtrack**," "**SDP+Greedy**" and "**Linear**" denote ILP formulation, SDP followed by backtrack mapping

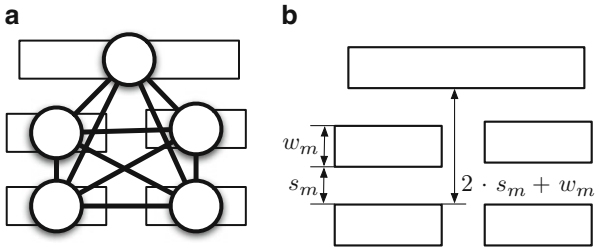

Fig. 3.23 $min_s = 2 \cdot s_m + w_m$ may cause K_5 structure

(Sect. 3.3.2), SDP followed by greedy mapping, and linear color assignment (Sect. 3.3.2), respectively. Here we implement an ILP formulation extended from the triple patterning work [1]. In SDP+Greedy, a greedy mapping from [1] is applied. All the graph division techniques, including GH-tree based division, are applied. The columns "cn#" and "st#" denote the number of conflicts and stitches, respectively. Column "CPU(s)" is the time color assignment takes in seconds.

From Table 3.3 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) the runtime for ILP increases dramatically such that none of them can be finished in 1 h. Compared with ILP, SDP+Backtrack can achieve near-optimal solutions, i.e., in every case the conflict number is optimal, while two more stitches are introduced in only one case. SDP+Greedy method can achieve 2× speedup against SDP+Backtrack. But the performance of SDP+Greedy worsens for complex designs hundreds of additional conflicts are reported. The linear color assignment can achieve around 200× speed-up 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 the best of our 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 3.4 evaluates the six most dense cases. We can see that compared with SDP+Backtrack, SDP+Greedy can achieve around 8× speed-up, but 15 % more conflicts are reported. In terms of runtime, linear color assignment can achieve 500× and 60× speed-up, 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 the number of conflicts. However, when a layout is very complex or involves many native conflicts, linear color

Table 3.3 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 3.4 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

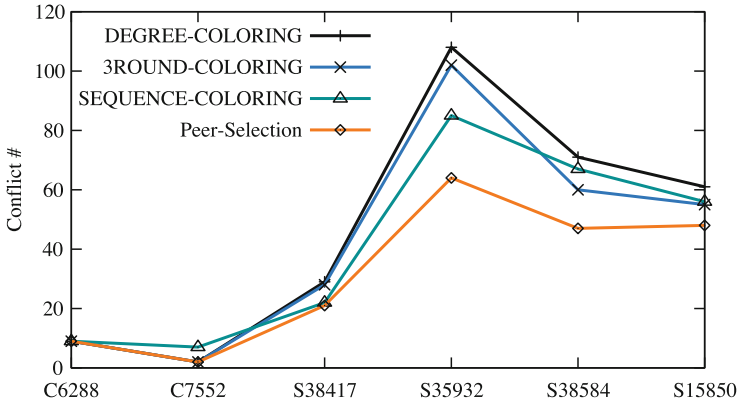

Fig. 3.24 Comparison on conflict number

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.

Lastly, Figs. 3.24 and 3.25 compare the performance of different vertex orders, in terms of the number of conflicts and stitches. Here *SEQUENCE-COLORING*, *DEGREE-COLORING*, and *3ROUND-COLORING* denote the coloring through the three different respective orders. *Peer selection* is the method proposed in our linear color assignment. From Fig. 3.24 we can clearly see that peer selection can achieve fewer conflicts than any single vertex order. This is because, for each test case, the whole decomposition graph is divided into several components. For each component one specific order may dominate and then be selected by peer selection. Therefore, for the whole layout, peer selection would be better than any single vertex ordering rule.

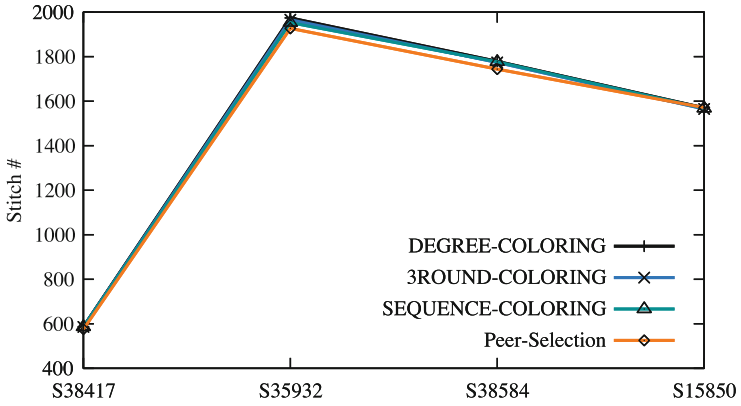

Fig. 3.25 Comparison on stitch number

3.4 Summary

In this section, we proposed an improved framework and algorithms to solve the LELE-EC layout decomposition problem. New end-cut candidates are generated, taking into consideration potential hotspots. The layout decomposition is formulated as an ILP. The experimental results show the effectiveness of our algorithms. It should be noted that our framework is very generic: 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 may be some faster solutions. Since end-cutting can provide better printability than the traditional LELELE process, we expect to see future work on the LELE-EC layout decomposition and LELE-EC aware design.

In this section we proposed the layout decomposition framework for quadruple patterning and beyond. Experimental evaluations have demonstrated that our algorithm is effective and efficient for obtaining a high quality solution. As technology nodes continue to be scaled to the sub-10 nm range, MPL may become an increasingly promising manufacturing solution. We believe this work will stimulate future research into this field, facilitating the advancement of MPL technology.

References

1. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2011)
2. Ausschnitt, C., Dasari, P.: Multi-patterning overlay control. In: Proceedings of SPIE, vol. 6924 (2008)

3. Lin, B.J.: Lithography till the end of Moore's law. In: ACM International Symposium on Physical Design (ISPD), pp. 1–2 (2012)
4. Banerjee, S., Li, Z., Nassif, S.R.: ICCAD-2013 CAD contest in mask optimization and benchmark suite. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 271–274 (2013)
5. Mack, C.: Fundamental Principles of Optical Lithography: The Science of Microfabrication. Wiley, New York (2008)
6. Cork, C., Madre, J.-C., Barnes, L.: Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns. In: Proceedings of SPIE, vol. 7028 (2008)
7. Fang, S.-Y., Chen, W.-Y., Chang, Y.-W.: A novel layout decomposition algorithm for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 1185–1190 (2012)
8. Tian, H., Zhang, H., Ma, Q., Xiao, Z., Wong, M.D.F.: A polynomial time triple patterning algorithm for cell based row-structure layout. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 57–64 (2012)
9. Kuang, J., Young, E.F.: An efficient layout decomposition approach for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 69:1–69:6 (2013)
10. Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., Pan, D.Z.: A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 163–169 (2013)
11. Zhang, Y., Luk, W.-S., Zhou, H., Yan, C., Zeng, X.: Layout decomposition with pairwise coloring for multiple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 170–177 (2013)
12. Yu, B., Pan, D.Z.: Layout decomposition for quadruple patterning lithography and beyond. In: ACM/IEEE Design Automation Conference (DAC), pp. 53:1–53:6 (2014)
13. Ma, Q., Zhang, H., Wong, M.D.F.: Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology. In: ACM/IEEE Design Automation Conference (DAC), pp. 591–596 (2012)
14. Lin, Y.-H., Yu, B., Pan, D.Z., Li, Y.-L.: TRIAD: a triple patterning lithography aware detailed router. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 123–129 (2012)
15. Tian, H., Du, Y., Zhang, H., Xiao, Z., Wong, M.D.F.: Constrained pattern assignment for standard cell based triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 178–185 (2013)
16. Yu, B., Xu, X., Gao, J.-R., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 349–356 (2013)
17. Kahng, A.B., Park, C.-H., Xu, X., Yao, H.: Layout decomposition for double patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 465–472 (2008)
18. Sun, J., Lu, Y., Zhou, H., Zeng, X.: Post-routing layer assignment for double patterning. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 793–798 (2011)
19. Cormen, T.T., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
20. Yu, B., Gao, J.-R., Pan, D.Z.: Triple patterning lithography (TPL) layout decomposition using end-cutting. In: Proceedings of SPIE, vol. 8684 (2013)
21. Xu, Y., Chu, C.: A matching based decomposer for double patterning lithography. In: ACM International Symposium on Physical Design (ISPD), pp. 121–126 (2010)
22. Yuan, K., Yang, J.-S., Pan, D.Z.: Double patterning layout decomposition for simultaneous conflict and stitch minimization. In: ACM International Symposium on Physical Design (ISPD), pp. 107–114 (2009)
23. Yang, J.-S., Lu, K., Cho, M., Yuan, K., Pan, D.Z.: A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 637–644 (2010)

24. Gurobi Optimization Inc.: Gurobi optimizer reference manual. http://www.gurobi.com (2014)
25. Karger, D., Motwani, R., Sudan, M.: Approximate graph coloring by semidefinite programming. J. ACM **45**, 246–265 (1998)
26. Appel, K., Haken, W.: Every planar map is four colorable. Part I: discharging. Ill. J. Math. **21**(3), 429–490 (1977)
27. Kuratowski, C.: Sur le probleme des courbes gauches en topologie. Fundam. Math. **15**(1), 271–283 (1930)
28. Robertson, N., Sanders, D.P., Seymour, P., Thomas, R.: Efficiently four-coloring planar graphs. In: ACM Symposium on Theory of computing (STOC), pp. 571–575 (1996)
29. Tang, X., Cho, M.: Optimal layout decomposition for double patterning technology. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 9–13 (2011)
30. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. Soc. Ind. Appl. Math. **9**(4), 551–570 (1961)
31. Gusfield, D.: Very simple methods for all pairs network flow analysis. SIAM J. Comput. **19**(1), 143–155 (1990)
32. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in networks with power estimation. Sov. Math. Dokl **11**(5), 1277–1280 (1970)
33. Borchers, B.: CSDP, a C library for semidefinite programming. Optim. Methods Softw. **11**, 613–623 (1999)

Chapter 4
Standard Cell Compliance and Placement Co-Optimization

4.1 Introduction

The TPL layout decomposition problem with conflict and stitch minimization has been studied extensively in the past few years [1–9], including the early work presented in Chaps. 2 and 3. However, most existing work suffers from one or more of the following drawbacks. (1) Because the TPL layout decomposition problem is NP-hard [3], most of the decomposers are based on approximation or heuristic methods, possibly leading to extra conflicts being reported. (2) For each design, since the library only contains a fixed number of standard cells, layout decomposition would contain numerous 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 a 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.

It has been observed that most hard-to-decompose patterns originate from the contact and M1 layers. Figure 4.1 shows two common native TPL conflicts in the contact and M1 layers, respectively. As shown in Fig. 4.1a, contact layout within the standard cell may generate some four-clique patterns, which cannot be decomposed. Meanwhile, if placement techniques are not TPL-friendly, some boundary metals may introduce native conflicts (see Fig. 4.1b). Since redesigning indecomposable patterns in the final layout requires high ECO efforts, generating TPL-friendly layouts, especially in the early design stage, becomes especially urgent. Through these two examples, we can see that TPL constraints should be considered in both the standard cell design and placement stages, so that we can avoid indecomposable patterns in the final layout.

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_4

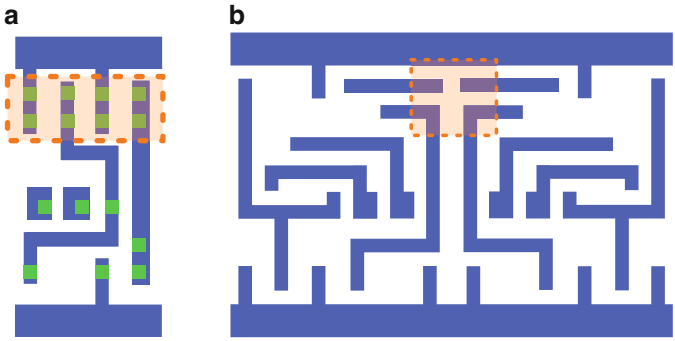

Fig. 4.1 Two native conflicts (*in red boxes*) from (**a**) contact layer within a standard cell; (**b**) M1 layer between adjacent standard cells

There exist several placement studies of different manufacturing process targets [10–15]. Liebmann et al. [10] proposed some guidelines to enable DPL-friendly standard cell design and placement. Taghavi et al. [15] presented a set of algorithms to disperse local congestion. Recently, Ma et al. [16] and Lin et al. [17] proposed TPL aware detailed routing schemes. However, to the best our knowledge, no previous work has addressed TPL compliance at either the standard cell or placement levels.

In this section, we present a systematic framework for seamlessly integrating TPL constraints into the early design stages, handling 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 time-consuming conventional 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 solutions.
- We present the first systematic study of TPL-aware ordered single row (TPL-OSR) placement, where cell placement and color assignment can be solved simultaneously.
- We propose a linear dynamic programming algorithm to solve TPL-aware single row placement with maximum displacement while also achieving a good trade-off in terms of runtime and solution quality.
- Our framework seamlessly integrates decomposition in each key step, removing the need for additional layout decomposition.
- Experimental results show that our framework can achieve zero conflicts while effectively reduce the stitch number.

4.2 Preliminaries

4.2.1 Row Structure Layout

Our framework assumes a row-structure layout where cells in each row are at the same height, and power and ground rails go from the very left to the very right (see Fig. 4.2a). A similar assumption was made in the row-based TPL layout decomposition in [5] as well. The minimum metal feature width 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 as d_{row}. If we further analyze the layout patterns in the library, we observe that the width of a power or ground rail is twice the width of a metal wire within standard cells [18]. Under the row-structure layout, we have the following lemma.

Lemma 4.1. *There is no coloring conflict between two M1 wires or contacts from different rows.*

Proof. For TPL, the layout will be decomposed into three masks, which means layout features within the minimum coloring distance are assigned three colors to

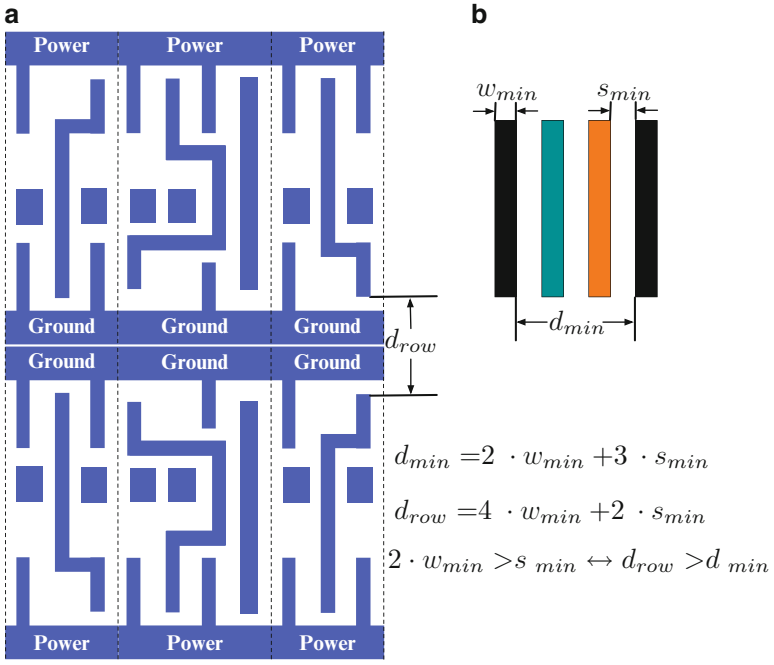

Fig. 4.2 (**a**) Minimum spacing between M1 wires among different rows. (**b**) Minimum spacing between M1 wires with the same color

increase the pitch between neighboring features. Then, we can see from Fig. 4.2 that 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 the M1 layer. For the same reason, we can achieve a similar conclusion for the contact layer.

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*). The decomposed results for each row will then 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.

4.2.2 Overall Design Flow

The overall flow of our proposed framework is illustrated in Fig. 4.3, 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

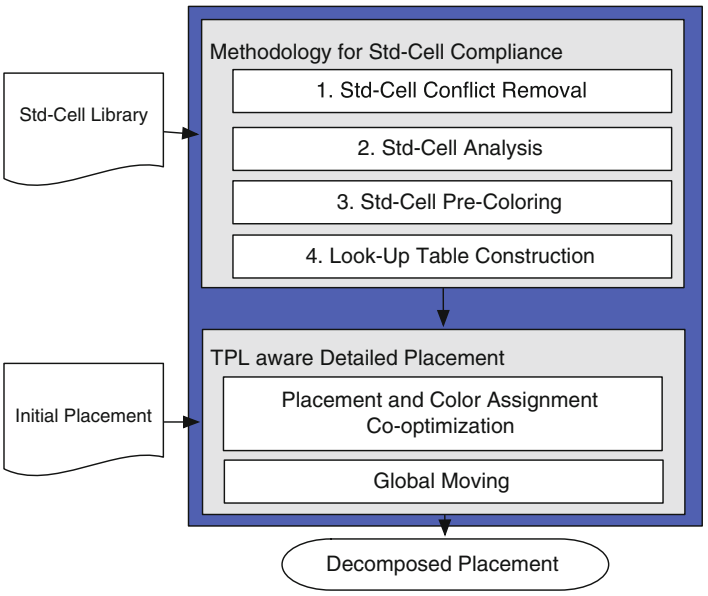

Fig. 4.3 Overall flow of the methodologies for standard cell compliance and detailed placement

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, a 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 Sects. 4.4.1 and 4.4.2), and global moving (see Sect. 4.4.3).

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 a decomposed layout that resolves cell placement and color assignment simultaneously.

4.3 Standard Cell Compliance

Without considering TPL in standard cell design, the cell library may involve several cells with native TPL conflicts (see Fig. 4.1a for one example). The inner native TPL conflict cannot be resolved through either cell shift or layout decomposition. Additionally, one cell may be applied many times in a single design, implying that each inner native conflict may cause hundreds of coloring conflicts in the final layout. To achieve a 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.

4.3.1 Native TPL Conflict Removal

An example of native TPL conflict is illustrated in Fig. 4.4, 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 a TPL-friendly layout [19]. 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 conflicts. As shown in Fig. 4.4, with slight modification to the original layout, we can choose either to move contacts connected with power/ground rails or to shift contacts on the signal paths of the cell. We call these two options case 1 and case 2, respectively, both of which will lead to a TPL-friendly standard cell layout. Note that although conventional cell migration techniques [20–22] 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.

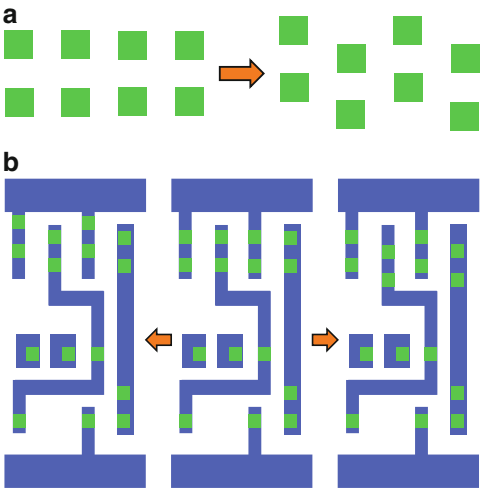

Fig. 4.4 Contact layout modification to hexagonal packing. (**a**) The principle for contact shifting; (**b**) demonstration of two options for contact shifting, with original layout in the *middle*, case 1 on the *left* and case 2 on the *right*

4.3.2 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 due to layout modification.

A Nangate 45 nm Open Cell Library [18] has been scaled to 16 nm technology node. After native TPL conflict detection and layout modification, we carry out the standard cell level timing analysis. Calibre xRC [23] 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 directly result from layout modification. We use SPICE simulation to characterize different types of gates, which is based on the 16 nm PTM model [24]. Then, we can get the propagation delay of each gate, which is the average of the rising and falling delays. We pick the six most commonly used cells to measure the relative change in the propagation delay due to layout modification (see Fig. 4.5). It is clear that, for both case 1 and case 2, the timing impact will be within 0.5 % of the original propagation delay of the gates, which can be 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.

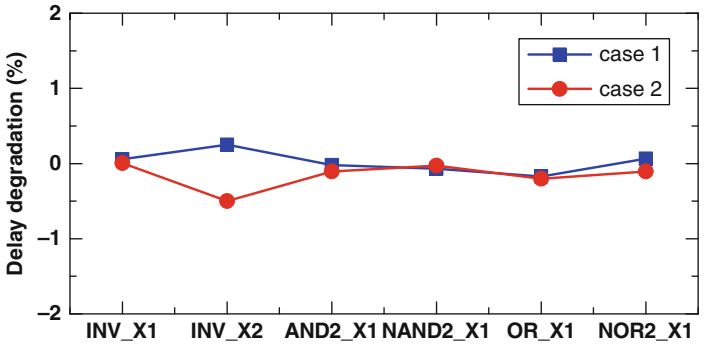

Fig. 4.5 The timing impact from layout modification for different types of gates, including case 1 and case 2

4.3.3 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.

Problem Formulation

Given the input standard cell layout, all the stitch candidates are captured through wire projection [6]. One feature in the layout is divided into two adjacent parts if one stitch candidate is introduced. Then a *constraint graph (CG)* is constructed to represent all input features and all of 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. There is a stitch edge iff two untouched vertices are within the minimum coloring distance d_{min}. For example, given an input layout shown in Fig. 4.6a, five stitch candidates are generated through wire projection. The constraint graph is illustrated in Fig. 4.6b, where the conflict edges and the stitch edges are shown as solid edges and dash edges, respectively. Note that we forbid the stitch on small features, e.g., contact, due to printability issues. 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, unlike in conventional layout decomposition, pre-coloring could have more than one solution for each cell. For some complex cell structures, if we exhaustively enumerate all possible colorings, there may be thousands of solutions. A large solution size would impact the performance of our

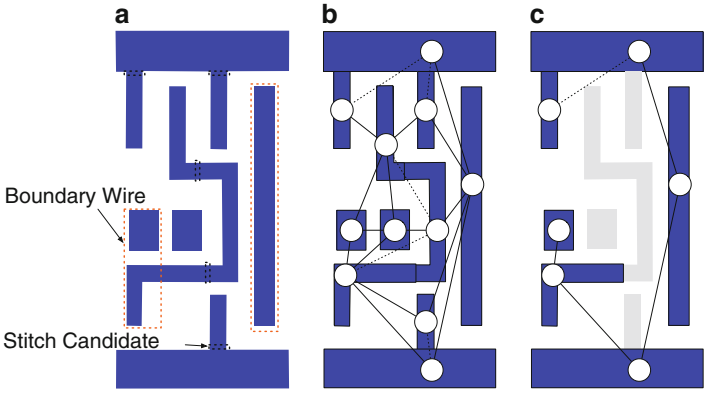

Fig. 4.6 Constraint graph construction and simplification. (**a**) Input layout and all stitch candidates. (**b**) Constraint graph (CG) where *solid edges* are conflict edges and *dash edges* are stitch edges. (**c**) The simplified constraint graph (SCG) after removing immune features

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

Definition 4.1 (Immune Feature). In one standard cell, an immune feature is an inside feature that does not conflict with any outside feature.

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

Definition 4.2 (Redundant Coloring Solutions). If two coloring solutions are only different at the immune features, these two solutions are redundant with respect to each other.

Problem 4.1 (Standard Cell Pre-coloring). Given the input standard cell layout and the maximum allowed stitch number, *maxS*, construct the CG. The 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, we temporarily remove these features to avoid redundant coloring solutions. We denote the remaining graph as a *simplified constraint graph (SCG)*. For example, for the constraint graph in Fig. 4.6b, the corresponding SCG is shown in Fig. 4.6c. Our standard cell pre-coloring algorithm consist of two stages: coloring solution enumeration on SCG and solution verification on CG.

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**

SCG Solution Enumeration

In the first step, given a SCG, we enumerate all possible coloring solutions. Our enumeration is based on a backtracking algorithm [25], 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 a color c, no conflicts are introduced, and the total stitch number does not exceed *maxS*. It should be mentioned that, since all power/ground rails are assigned a default color, the colors of the corresponding vertices are assigned before the backtracking process. For example, given the SCG shown in Fig. 4.6c, if no stitch is allowed, there are eight solutions (see Fig. 4.7).

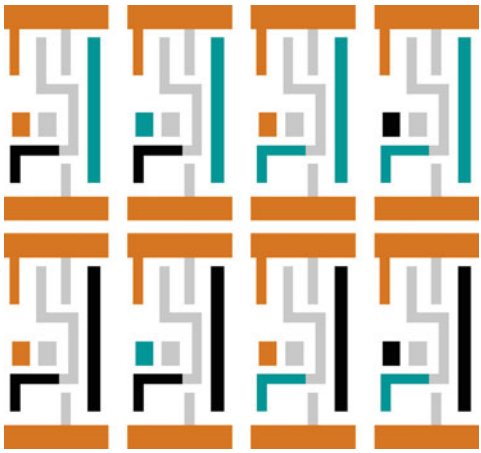

Fig. 4.7 AND2X1 cell example: eight enumerated solutions for SCG

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** ▷ $s_i[t] = -1$
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**

CG Solution Verification

Until now, we have enumerated all coloring solutions for a SCG. However, not all of the SCG solutions can achieve a legal layout decomposition in the initial constraint graph (CG). Therefore, in the second step, CG solution verification is performed on each generated solution. Since a SCG is a subset of a CG, the verification can be viewed as layout decomposition with pre-colored features on SCG. If a coloring solution with stitch number less than *maxS* for the whole CG can be found, it will be stored as one pre-coloring solution. The CG solution verification is based on the **branch-and-bound** algorithm [25], 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 problems, i.e., only one solution is generated. (2) The branch-and-bound algorithm introduces a bounding function to prune sub-optimal nodes in the 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 12. Given a set of SCG coloring solutions $S' = \{s'_1, s'_2 \ldots s'_n\}$, we first generate the corresponding

CG coloring solutions $S = \{s_1, s_2, \ldots, s_n\}$ (line 1). Then we iteratively check each coloring solution s_i (lines 2–6). For a coloring solution s_i, if vertex t belongs to the SCG, $s_i[t]$ should already have been assigned a legal color. If t does not belong to SCG, $s_i[t] \leftarrow -1$. The BRANCH-AND-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 a legal color in the SCG, we skip t and travel to the next vertex. Otherwise, a legal color would be assigned to t before traveling to the next vertex. Unlike in an exhaustive search, the search space can be effectively reduced through the pruning process (lines 11–12). The function LOWER-BOUND() returns the lower bound by calculating the current stitch number. Note that if a conflict is found, then the function returns a large value. Before checking a legal color of vertex t, we calculate its lower bound first. If LOWER-BOUND() is larger than *minCost*, we will not branch from t, since all of the children solutions will be of higher cost than *minCost*. The traversal will assign all vertices legal colors, stored in s_i. Afterwards, if $minCost \leq maxS$, then s_i is one of the pre-coloring solutions (lines 5–6).

Note that, although other layout decomposition techniques, like integer linear programming (ILP), may be modified to act as the verification engine, our branch-and-bound based method is easy to implement and effective for the standard cell level problem size. Even for the most complex cell, SCG solution enumeration and CG solution verification can be finished in under 5 s. For the SCG solutions in Fig. 4.7, four solutions are verified and assigned final colorings (see Fig. 4.8). These four solutions would be the final coloring solutions for this standard cell, and are provided as a supplement to the library.

4.3.4 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 look-up table. For example, assume that two cells, c_i and c_j, are assigned the pth and qth coloring solutions, respectively. Then we store into the look-up table a value LUT(i, p, j, q), which is the minimum distance required when c_i is to the left of c_j. If two colored cells can legally be abutted to each other, the corresponding value would be 0. Otherwise, the value would be the site number

Fig. 4.8 AND2X1 cell example: in CG 4 verified solutions are stored as final coloring solutions

required to keep the two cells decomposable. Meanwhile, for each cell, the stitch numbers in the different coloring solutions are also stored. Note that during the look-up table construction, we consider cell flipping and store related values as well.

4.4 TPL Aware Detailed Placement

4.4.1 TPL Aware Ordered Single Row Placement

We first solve a single row placement, where we determine the orders of all cells on the row. When the TPL process is not considered, this row based design problem is known as the OSR problem, which has been well studied [26–29]. Here we revisit the OSR problem with the TPL process consideration. For convenience, Table 4.1 lists the notations used in this section.

Problem Formulation

We consider a single input row as m ordered sites $R = \{r_1, r_2, \ldots, r_m\}$, and an n input 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 pth 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 have 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:

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

Table 4.1 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 pth color solution
$\text{LUT}(i, p, j, q)$	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

where LUT(i, p, j, q) is the minimum distance required between (i, p) & (j, q). Based on these notations, we define the *TPL-OSR* problem as follows.

Problem 4.2 (TPL Aware Ordered Single Row Problem). Given a single row placement, we seek a legal placement and cell color assignment such 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 the cell placement, but it also needs to assign appropriate coloring solutions to 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 the 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, we cannot calculate the required site number before the coloring assignment.

In addition, note 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 picking up compatible coloring solutions, TPL-OSR can seek to flip cells (see Fig. 4.9a) or shift cells (see Fig. 4.9b).

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, we construct a directed acyclic graph $G = (V, E)$. The graph G has vertex set V and edge set E. $V = \{\{0, \ldots, m\} \times \{0, \ldots, N\}, t\}$, where $N = \sum_{i=1}^{n} v_i$. The vertex in the first row and 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 for one cell. Without loss of generality, we label each row as $r(i, p)$, if it is related to cell c_i with the pth coloring solution. The edge set E is composed of three sets of edges: horizontal edges E_h, ending edges E_e, and diagonal edges E_d.

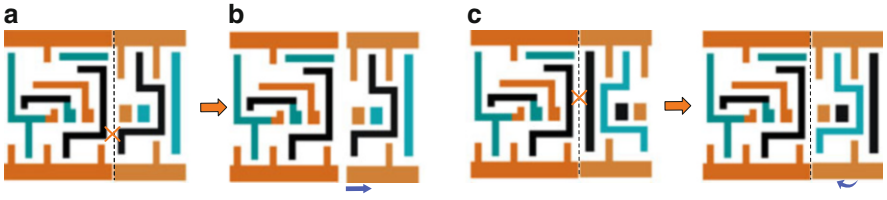

Fig. 4.9 Two techniques for removing conflicts during placement. (**a**) Flip the cell; (**b**) shift the cell

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

We denote each edge by its start and end point. A legal TPL-OSR solution corresponds to finding a directed path from vertex s to vertex t. Sometimes one row cannot handle insertion of all the cells, so we introduce ending edges. With these ending edges, the graph model is guaranteed to contain a 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 have zero cost. (2) For ending edge $\{(r(i, p), m) \rightarrow t\}$, it is labeled with cost $(n - i) \cdot M$, where M is a large number. (3) For diagonal edge $\{(r(i, p), k) \rightarrow (r(j, q), k + w(c_j) + \text{LUT}(i, p, j, q))\}$, it is labelled with the cost calculated 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. 4.10. Note that for clarity, we do not show the diagonal edges.

One example of the graph model is illustrated in Fig. 4.11, 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 the 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. 4.11b–d, where each figure shows a different part of the 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. Figure 4.12 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.

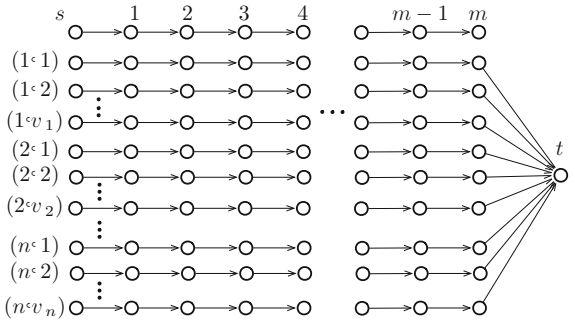

Fig. 4.10 Graph model for the TPL-OSR problem (only the horizontal edges and ending edges are showed)

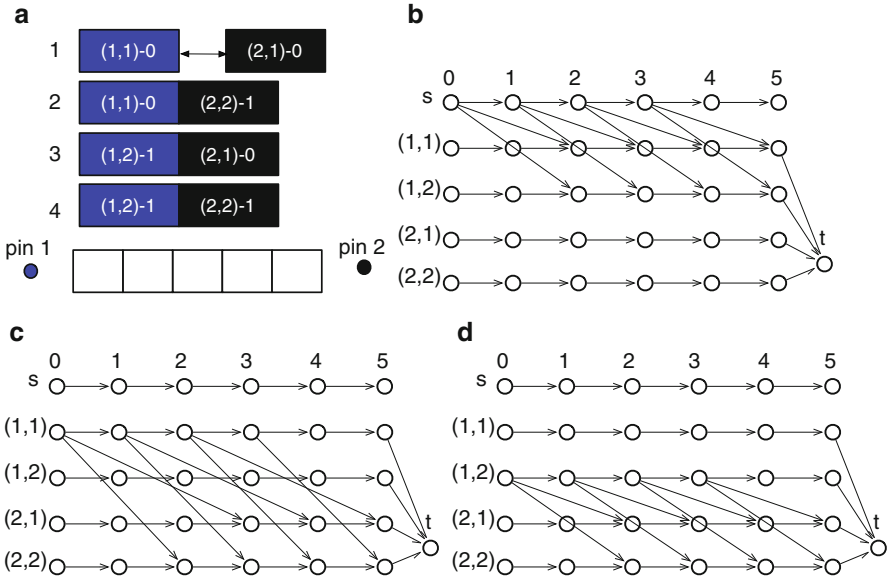

Fig. 4.11 Example for the TPL-OSR problem. (**a**) Two cells with different coloring solutions to be placed into a 5 sites row; graph models with diagonal edges (**b**) from s vertex to first cell; (**c**) from c1_1 to second cell; (**d**) from c1_2 to second cell

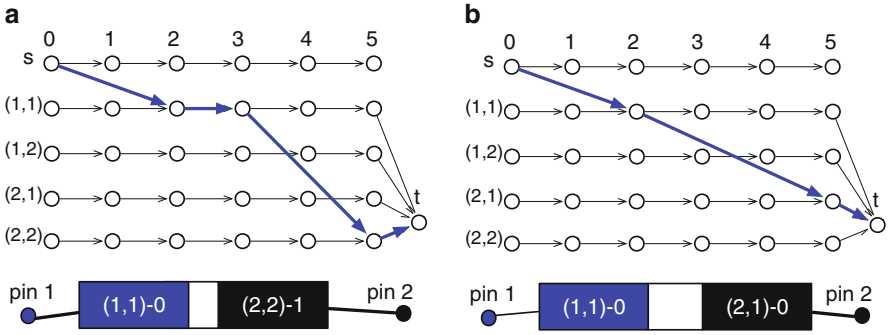

Fig. 4.12 Shortest path solutions on the graph model with (**a**) 1 stitch and (**b**) 0 stitch

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

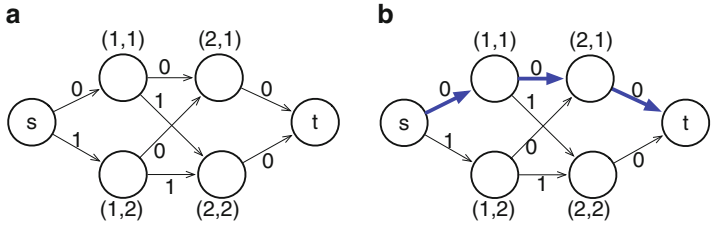

Fig. 4.13 (**a**) The first stage to solve color assignment. In this example edge cost only considers the stitch number minimization. (**b**) One shortest path solution corresponds to a color assignment solution

Two-Stage Graph Model

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

To solve the example in Fig. 4.11, the first stage graph model is illustrated in Fig. 4.13a, 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.

Our second stage handles cell placement, and we consider the previous color assignment solutions here. That is, if in previous color assignment cells, c_{i-1} and c_i are assigned its pth and qth 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 [29] can be directly applied here. The second stage graph model for the example in Fig. 4.11 is illustrated in Fig. 4.14. Note that since all cells have been assigned a coloring solution, the graph size is much smaller than that in Fig. 4.11. As shown in Fig. 4.14b, 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 cannot achieve an optimal solution of the TPL-OSR problem, applying the two-stage graph model can reduce the complexity from $O(mnK)$ to $O(nK + mn)$.

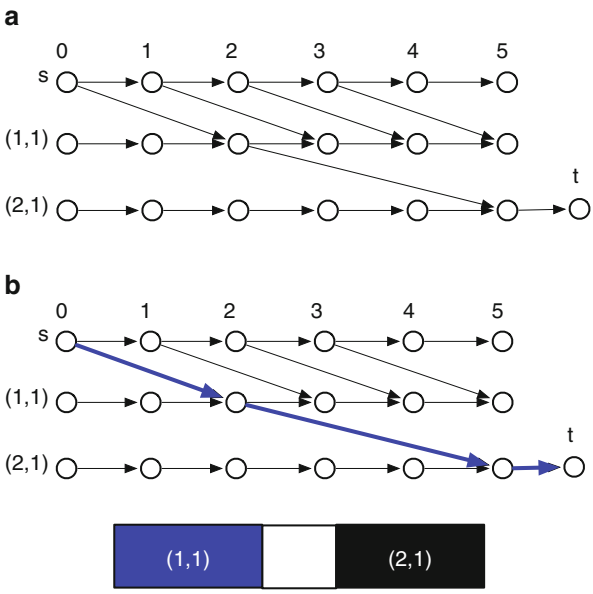

Fig. 4.14 (**a**) The second stage to solve detailed placement. (**b**) One shortest path solution corresponds to a cell placement

4.4.2 TPL-OSR with Maximum Displacement

Here we consider another single row placement problem similar to the TPL-OSR, where we determine the initial cell orders. Unlike in TPL-OSR, each cell is forbidden from moving more than a distance M from its original location. This 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 solving the color assignment and cell placement separately, its solution qualities may not be good enough. For the new problem, we are able to propose a fast and high performance optimization algorithm. **Second**, from a 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.

Problem Formulation

The problem we solve is finding new locations for all cells that preserve their relative order. Meanwhile, each cell has a maximum moving distance, M, from its original location. In other words, each cell c_i has $2M + 1$ possible new 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:

Table 4.2 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

Problem 4.3 (TPL-OSR with Maximum Displacement). Given a single row placement, we seek cell displacement values $d(i)$, with $|d(i)| < M$, and color assignments such that the HPWL of all nets and the total stitch number are minimized.

Linear Dynamic Programming Algorithm

Inspired by Taghavi et al. [15], our algorithm is based on linear dynamic programming, which means the optimal solution can be found in linear time. The general idea is to 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 for the optimal placement and color assignment simultaneously for c_i. For convenience, Table 4.2 lists some additional notations used in this 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 (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 wirelength 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.

Unlike in the method described in [15], 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

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**

inferior to $t[i][d2][a]$. Then $r[i][d1][a]$ is assigned 1 to denote inferiority (line 16). Therefore, one can exit early when checking the r value (lines 5–7). Lines 18–24 compute the end case of the last cell in the row, and the solution is recovered at the end (lines 25–28).

Theorem 4.1. *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. Note that the unified graph model in Sect. 4.4.1 can also be modified to solve the problem here. However, the runtime complexity using the unified graph model is $O(nmK)$. Since m is usually larger than n, the complexity of a unified graph model is quadratic and may be slower than linear dynamic programming.

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;

4.4.3 Overall Placement Scheme

In this section, we present our overall scheme for the design level TPL-aware detailed placement. Algorithm 14 summarizes the overall flow. Initially, 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 will be solved earlier. For each row row_i, we carry out single row TPL-aware detailed placement as introduced in Sects. 4.4.1 and 4.4.2, to solve color assignment and cell placement simultaneously. Note that it may be the case that in one row, we cannot assign all cells legal positions, due to the extra sites required to resolve coloring conflicts.

If the 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 and remove some local triple patterning conflicts. For each cell we define its "optimal region" as the placement site where the HPWL is optimal [30]. Note that one cell can only be moved to *FREE* rows. Since some cells in the middle of a row may be moved, we need to solve the OSR problem to rearrange the cell positions [29]. Note that since all cells on the row have been assigned colors, cell widths should be updated to preserve extra space for coloring conflicts (line 8–9) . After solving a row 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).

4.5 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 a 3.0 GHz CPU. We use as our initial standard cell library the Nangate 45 nm library [18], scaled down to a 16 nm technology node. We apply standard cell compliance and pre-coloring on the scaled library. During standard cell pre-coloring, each cell's maximum allowed stitch number, *maxS*, is set to 2. We use Design Compiler [31] to synthesize OpenSPARC T1 designs based on the modified cell library. For each benchmark, we perform placement with Cadence SOC Encounter [32] 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 4.3. Column "*K*" is the maximum cell pre-coloring solution number among all standard cell types, which is related to the look-up 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

Table 4.3 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	14,793	123	146	919
top-80	74	14,793	115	157	860
top-85	74	14,793	112	158	832

Table 4.4 Comparisons with traditional flow

	Conventional flow		Our flow		
Bench	CN#	ST#	CN#	ST#	ΔST 0#
alu-70	461	8476	0	1013	−89.5 %
alu-80	428	10,862	0	1011	−91.2 %
alu-85	493	5559	0	1006	−82.2 %
byp-70	1168	43,730	0	2743	−96.7 %
byp-80	1251	35,973	0	2889	−95.6 %
byp-85	1342	38,576	0	3136	−95.6 %
div-70	821	19,492	0	2119	−91.9 %
div-80	860	18,928	0	2090	−90.8 %
div-85	982	22,070	0	2080	−91.9 %
ecc-70	368	7801	0	247	−96.4 %
ecc-80	362	10,083	0	274	−97.1 %
ecc-85	372	9990	0	369	−96.8 %
efc-70	222	7589	0	1005	−90.0 %
efc-80	225	13,006	0	1008	−93.5 %
efc-85	258	5260	0	1005	−83.7 %
ctl-70	354	15,356	0	573	−97.8 %
ctl-80	380	15,158	0	561	−97.4 %
ctl-85	414	14,873	0	556	−97.1 %
top-70	3708	58,953	0	8069	−90.5 %
top-80	3976	60,736	0	8120	−89.8 %
top-85	4265	70,316	0	8710	−90.9 %
Average	1081	23,466	**0**	1672.5	**−92.7 %**

size for 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.

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 4.4 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 [4] is used as our layout decomposer. Column "**Our flow**" is our 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 that for each flow, the standard cell inner native conflicts have been removed through use of our compliance techniques (see Sect. 4.3). In other words, the conflicts can only theoretically happen on the boundaries between standard cells.

On the one hand, we can see that in the conventional design flow, even when each standard cell is TPL-friendly, more than 1000 conflicts are reported on average in the final decomposed layout. Meanwhile, over 20,000 stitches are introduced on average for each case. Due to the large number of conflicts and stitches, substantial effort may be required to manually modify or migrate the layout to resolve the conflicts. On the other hand, through considering TPL constraints in the early design stages, our proposed TPL-aware design flow can guarantee **zero** conflicts. 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 %, compared with traditional flow.

In Sects. 4.4.1 and 4.4.2, we proposed several algorithms for solving TPL-aware single row detailed placement. In the second experiment, we analyzed the performance of the proposed algorithms and related speed-up techniques in Table 4.5. Column "**GREEDY**" is a greedy detailed placement algorithm [14] whose implementation is used as our baseline. Although the work in [14] 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. Column "**TPLPlacer-MDP**" applies the linear dynamic programming method (see Sect. 4.4.2) 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 wirelength difference taken before and after our TPL aware placement, where HPWL is applied to calculate the total wirelength. 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 s. However, in 9 out of 21 cases it cannot find legal placement solutions. For each illegal result "N/A" is labeled in the table. These illegal solutions result because 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 a greedy method as well, it loses the global view necessary to minimize the stitch number. We can observe that more stitches are reported for those cases where it finds legal results.

Next, we compare two TPL-OSR algorithms: "TPLPlacer" and "TPLPlacer-2Stage." 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 a 100× speed-up against "TPLPlacer." However, "TPLPlacer-2Stage" has a 19 % higher stitch number. This may be because under the two-stage graph model, placement and color assignment are optimized separately, so this speed-up technique may lose some optimality in terms of stitch number.

Table 4.5 Comparisons of detailed placement algorithms

	GREEDY [14]			TPLPlacer			TPLPlacer-2Stage			TPLPlacer-MDP		
Bench	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 %	4.8	
alu-80	N/A	N/A	N/A	−0.6 %	948	24.8	−0.62 %	1099	0.43	−0.67 %	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 %	1.01	
ecc-85	N/A	N/A	N/A	−0.49 %	312	5.79	−0.48 %	344	0.13	−0.55 %	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**

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

For test case ctl-70, Fig. 4.15 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 Figs. 4.15a, b are very similar, which means that speed-up technique TPLPlacer-MDP can achieve a very comparable result with TPLPlacer. However, another speed-up technique, TPLPlacer-2Stage, may involve more final stitches (see Fig. 4.15c).

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 greatly impacts the performance of detailed placement. Therefore, to demonstrate the robustness of "TPLPlacer-MDP," it would be interesting to analyze the performance with different M settings. Figure 4.16 gives such an analysis for test cases alu_70, alu_80 and alu_85.

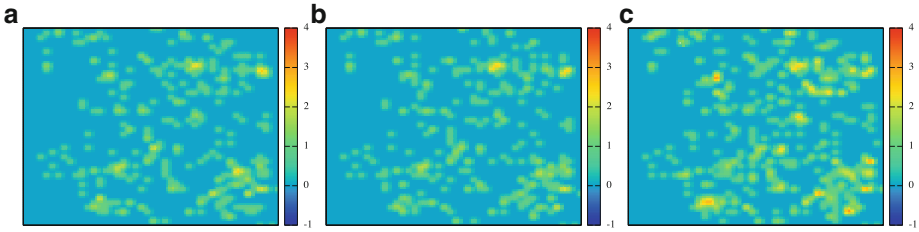

Fig. 4.15 For test case ctl-70, three different stitch density maps through different detailed placement algorithms. (**a**) 335 stitches through TPLPlacer; (**b**) 330 stitches through TPLPlacer-MDP; (**c**) 491 stitches through TPLPlacer-2Stage

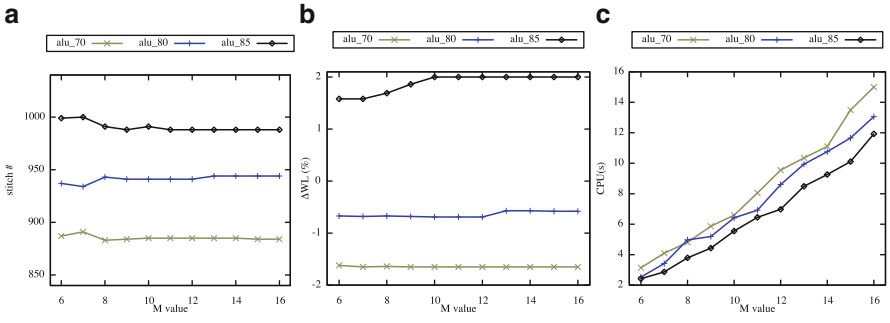

Fig. 4.16 TPLPlacer-MDP performance analyses with different M values for alu design cases. (**a**) Impact on stitch numbers. (**b**) Impact on wire-length improvements (ΔWL). (**c**) Impact on runtimes

From Figs. 4.16a, 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. 4.16c 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.

4.6 Summary

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

References

1. Cork, C., Madre, J.-C., Barnes, L.: Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns. In: Proceedings of SPIE, vol. 7028 (2008)
2. Ghaida, R.S., Agarwal, K.B., Liebmann, L.W., Nassif, S.R., Gupta, P.: A novel methodology for triple/multiple-patterning layout decomposition. In: Proceedings of SPIE, vol. 8327 (2012)
3. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2011)
4. Fang, S.-Y., Chen, W.-Y., Chang, Y.-W.: A novel layout decomposition algorithm for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 1185–1190 (2012)
5. Tian, H., Zhang, H., Ma, Q., Xiao, Z., Wong, M.D.F.: A polynomial time triple patterning algorithm for cell based row-structure layout. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 57–64 (2012)
6. Kuang, J., Young, E.F.: An efficient layout decomposition approach for triple patterning lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 69:1–69:6 (2013)
7. Yu, B., Gao, J.-R., Pan, D.Z.: Triple patterning lithography (TPL) layout decomposition using end-cutting. In: Proceedings of SPIE, vol. 8684 (2013)
8. Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., Pan, D.Z.: A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 163–169 (2013)

9. Zhang, Y., Luk, W.-S., Zhou, H., Yan, C., Zeng, X.: Layout decomposition with pairwise coloring for multiple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 170–177 (2013)
10. Liebmann, L., Pietromonaco, D., Graf, M.: Decomposition-aware standard cell design flows to enable double-patterning technology. In: Proceedings of SPIE, vol. 7974 (2011)
11. Chen, T.-C., Cho, M., Pan, D.Z., Chang, Y.-W.: Metal-density-driven placement for CMP variation and routability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **27**(12), 2145–2155 (2008)
12. Hu, S., Shah, P., Hu, J.: Pattern sensitive placement perturbation for manufacturability. IEEE Trans. Very Large Scale Integr. VLSI Syst. **18**(6), 1002–1006 (2010)
13. Gupta, M., Jeong, K., Kahng, A.B.: Timing yield-aware color reassignment and detailed placement perturbation for bimodal cd distribution in double patterning lithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(8), 1229–1242 (2010)
14. Gao, J.-R., Yu, B., Huang, R., Pan, D.Z.: Self-aligned double patterning friendly configuration for standard cell library considering placement. In: Proceedings of SPIE, vol. 8684 (2013)
15. Taghavi, T., Alpert, C., Huber, A., Li, Z., Nam, G.-J., Ramji, S.: New placement prediction and mitigation techniques for local routing congestion. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 621–624 (2010)
16. Ma, Q., Zhang, H., Wong, M.D.F.: Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology. In: ACM/IEEE Design Automation Conference (DAC), pp. 591–596 (2012)
17. Lin, Y.-H., Yu, B., Pan, D.Z., Li, Y.-L.: TRIAD: a triple patterning lithography aware detailed router. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 123–129 (2012)
18. NanGate FreePDK45 Generic Open Cell Library. http://www.si2.org/openeda.si2.org/projects/nangatelib (2008)
19. Lucas, K., Cork, C., Yu, B., Luk-Pat, G., Painter, B., Pan, D.Z.: Implications of triple patterning for 14 nm node design and patterning. In: Proceedings of SPIE, vol. 8327 (2012)
20. Yuan, K., Pan, D.Z.: WISDOM: wire spreading enhanced decomposition of masks in double patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 32–38 (2010)
21. Fang, S.-Y., Chen, S.-Y., Chang, Y.-W.: Native-conflict and stitch-aware wire perturbation for double patterning technology. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **31**(5), 703–716 (2012)
22. Ghaida, R.S., Agarwal, K.B., Nassif, S.R., Yuan, X., Liebmann, L.W., Gupta, P.: Layout decomposition and legalization for double-patterning technology. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(2), 202–215 (2013)
23. Mentor Graphics.: Calibre verification user's manual (2008)
24. Predictive Technology Model ver. 2.1. http://ptm.asu.edu (2008)
25. Neapolitan, R., Naimipour, K.: Foundations of Algorithms. Jones & Bartlett Publishers, New Delhi (2010)
26. Vygen, J.: Algorithms for detailed placement of standard cells. In: IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE), pp. 321–324 (1998)
27. Kahng, A.B., Tucker, P., Zelikovsky, A.: Optimization of linear placements for wirelength minimization with free sites. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 241–244 (1999)
28. Brenner, U., Vygen, J.: Faster optimal single-row placement with fixed ordering. In: IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE), pp. 117–121 (2000)
29. Kahng, A.B., Reda, S., Wang, Q.: Architecture and details of a high quality, large-scale analytical placer. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 891–898 (2005)
30. Goto, S.: An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. IEEE Trans. Circuits Syst. **28**(1), 12–18 (1981)
31. Synopsys IC Compiler. http://www.synopsys.com (2013)
32. Cadence SOC Encounter. http://www.cadence.com (2013)

Chapter 5
Design for Manufacturability with E-Beam Lithography

5.1 Introduction

As the minimum feature size continues to scale to sub-22 nm, conventional 193 nm optical photolithography technology is reaching its printability limit. In the near future, multiple patterning lithography (MPL) has become one of the viable lithography techniques for 22 nm and 14 nm logic nodes [1–4]. In the long run, i.e., for the logic nodes beyond 14 nm, extreme ultra violet (EUV), directed self-assembly (DSA), and electric beam lithography (EBL) are promising candidates as next generation lithography technologies [5]. Currently, both EUV and DSA suffer from some technical barriers: EUV technique is delayed due to tremendous technical issues such as lack of power sources, resists, and defect-free masks [6, 7]; DSA has the potential only to generate contact or via layers [8].

EBL system, on the other hand, has been developed for several decades [9]. Compared with the traditional lithographic methodologies, EBL has several advantages. (1) Electron beam can easily be focused into nanometer diameter with charged particle beam, thus can avoid suffering from 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 a great flexibility for fast turnaround times or 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.

Conventional EBL system applies variable shaped beam (VSB) technique [10]. In the VSB mode, the entire layout is decomposed into a set of rectangles, through a process called layout fracturing. Each rectangle is shot into resist by one electron beam. The printing process of VSB mode is illustrated in Fig. 5.1a. At first an electrical gun generates the initial beam, which becomes uniform through a shaping aperture. Then a second aperture finalizes the target shape with a limited maximum

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_5

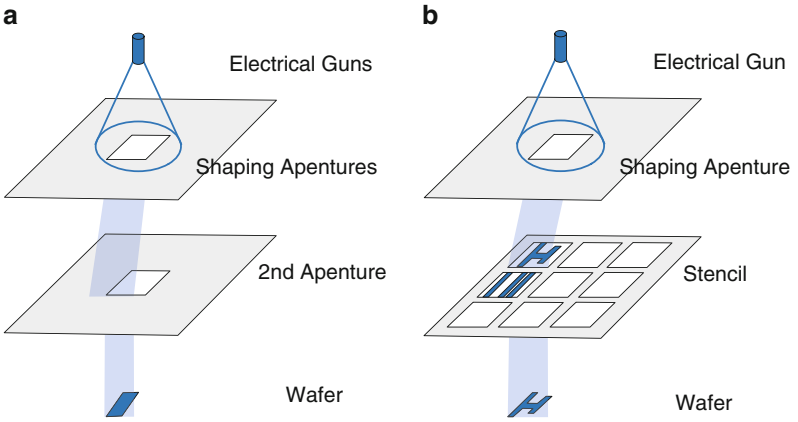

Fig. 5.1 Printing process of conventional EBL system: (**a**) VSB mode; (**b**) CP mode

size. Since each pattern needs to be fractured into pieces of rectangles and printed one by one, the VSB mode suffers from serious throughput problem.

One improved technique in EBL system is called character projection (CP) [10], where the second aperture is replaced by a *stencil*. As shown in Fig. 5.1b, in CP mode 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 the VSB mode. In this way the CP mode can improve the throughput significantly. In addition, CP exposure has a good critical dimension control stability compared with VSB [11]. 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 the stencil. Those patterns not contained by stencil are still required to be written by VSB [12]. Thus one emerging challenge in CP mode is how to pack the characters into the stencil to effectively improve the throughput.

Many previous works dealt with the design optimization for EBL system. For VSB mode, [13–16] considered EBL as a complementary lithography technique to print via/cut patterns or additional patterns after multiple patterning mask processes. Fang et al. [17] integrated EBL constraints into an early routing stage to avoid stitching line-induced bad patterns for parallel EBL system, and [18, 19] solved the subfield scheduling problem to reduce the critical dimension distortion. Kahng et al. [20], Ma et al. [21], Yu et al. [22], and Chan et al. [23] proposed a set of layout/mask fracturing approaches to reduce the VSB shot number. Besides, several works solved the design challenges under CP technique. Before stencil manufacturing, all design steps should consider the character projection, i.e., character aware library building [24], technology mapping [25], and character sizing problem [26]. Besides, [27, 28] proposed several character design methods for both via layers and interconnect layers to achieve stencil area-efficiency. After stencil

manufacturing, characters are stamped to practical layout patterns to minimize the electron beam shot number, especially for irregular routing or via layout [27, 29]. Besides, [30] proposed a structured routing architecture for high throughput CP mode.

However, even with decades of development, the key limitation of the EBL system has been and still is the low throughput. For example, at the 22 nm logic node and beyond, due to the design shrink and the inverse lithography techniques or optical proximity correction (OPC), the total electron beam shot number each mask could be larger than 1T [31]. To overcome the throughput issue, in this chapter we will present two methodologies. In Chap. 5.2 we solve the L-shape based layout fracturing problem, which can improve the throughput of VSB mode. In Chap. 5.3 we handle the overlapping aware stencil planning (OSP) problem, which benefits the CP mode.

5.2 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 VSB.

As the minimum feature size decreases, the number of rectangles in the layout steadily increases. Highly complex OPC causes both the writing time and data volume to increase as well. The cost, which scales with both writing time and data volume, increases as a result. Low throughput thus remains the bottleneck for EBL writing.

To overcome this manufacturing problem, several optimization methods have been proposed to reduce the EBL writing time to a reasonable level [32–34]. These optimization methods contain both hardware improvement and software speed-up, e.g., jog alignment, multi-resolution writing, character projection, and L-shape shot. One of these, the L-shape shot strategy, is a very simple yet effective approach to reduce the e-beam mask writing time, thus reducing the mask manufacturing cost and improving the throughput [32, 33]. This technique can also be applied to reduce the cost of the lithographic process. Conventional EBL writing is based on rectangular VSB shots. 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. The printing process of the L-shape shot, an improved technique, is illustrated in Fig. 5.2. To take advantage of this new printing process, a new fracturing methodology is needed to provide the L-shape in the fractured layout. One additional aperture, the third aperture, is employed to create L-shape shots. This strategy can potentially reduce the EBL writing time or cost by 50 % if all rectangles are combined into L-shapes. For example, in Fig. 5.3, instead of four rectangles, using L-shape fracturing only requires two L-shape shots.

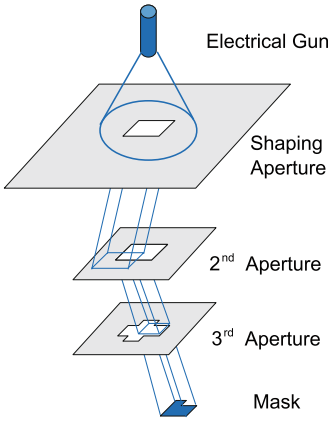

Fig. 5.2 L-shape writing process with one additional aperture

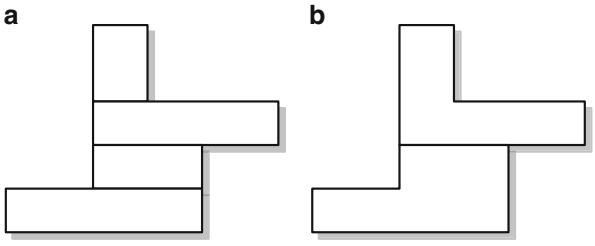

Fig. 5.3 Examples of polygon fracturing. (**a**) Rectangular shots with 4 shot number. (**b**) L-shape shots with two shot number

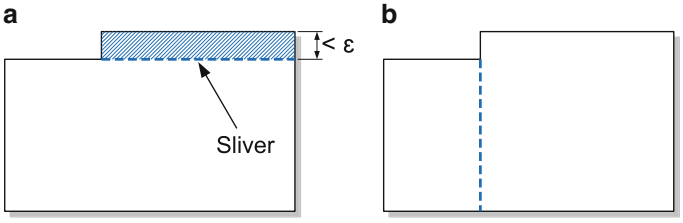

Fig. 5.4 (**a**) Fracturing with one sliver. (**b**) Fracturing without any slivers

Note that the layout fracturing problem is different from the general polygon decomposition problem in geometrical science. Taking into account yield 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 layout fracturing, sliver minimization is an important objective [35]. As shown in Fig. 5.4, two fractured layouts can achieve the same shot number: 2. However, because of the sliver, the fractured result shown in Fig. 5.4a is worse than that shown in Fig. 5.4b. It should be noted that the layout in Fig. 5.4 can be written in one L-shaped shot without a sliver.

Several papers have studied the layout fracturing problem for traditional rectangular layouts [20, 21, 35–37]. Kahng et al. proposed an integer linear programming (ILP) formulation and some speed-up techniques based on matching [20, 35]. Recently, Ma et al. [21] presented a heuristic algorithm that generates rectangular

shots and further reduces the number of slivers. The L-shape fracturing problem is newer than the rectangular fracturing problem, so there is only limited work so far, mostly describing methodology. No systematic algorithm has been proposed so far. Sahouria and Bowhill [32] reported initial results showing that L-shape fracturing can save about an additional 38 % of the shot count, but no algorithmic details were provided. For the general decomposition problem of polygons into L-shapes, several heuristic methods have been proposed [38, 39]. 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 of EBL L-shape fracturing that considers sliver minimization. We propose two algorithms for the L-shape fracturing problem. The first, called RM, takes rectangles generated by any previous fracturing framework and merges them into L-shapes. We then use a maximum-weighted matching algorithm to find the optimal merging solution, simultaneously minimizing the shot count and the number of slivers. To further overcome the intrinsic limitations of rectangular merging, we propose a second fracturing algorithm called DLF. Through effectively detecting and taking advantage of some special cuts, DLF can directly fracture the layout into a set of L-shapes in $O(n^2 \log n)$ time. Experimental results show that our algorithms are very promising for both shot count reduction and sliver minimization. DLF can even achieve a significant speed-up compared with previous state-of-the-art rectangular fracturing algorithms [21].

5.2.1 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 5.1 (Concave Vertex). The concave vertex of a polygon is one at which the internal angle is 270°.

Let c be the number of concave vertices in P; [40] gave the relationship between n and c: $n = 2c + 4$. If the number of concave vertices c is odd, polygon P is called an *odd polygon*; otherwise, P is an *even polygon*.

Definition 5.2 (Cut). A cut of a polygon P is a horizontal or vertical line segment that has at least one endpoint 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 an odd number of concave vertices lying on one side or the other, then the cut is called an *odd-cut*. If an cut is both an odd-cut and chord,

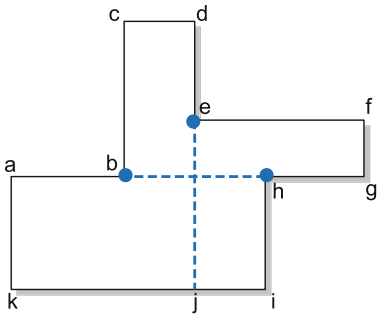

Fig. 5.5 Concepts of concave vertices and cuts

it is called an *odd-chord*. These concepts are illustrated in Fig. 5.5, where vertices b, e, h are concave vertices, edges $\bar{bh}, \bar{ej}$ are odd-cuts, and edge $\bar{bh}$ is a chord. Note that $\bar{bh}$ is also an odd-chord.

Definition 5.3 (L-Shape). An L-shape is a polygon whose shape is in the form of the letter "L".

An L-shape can also be 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 6, i.e., $n = 6$. There must then only be one concave vertex, i.e., $c = 1$.

Definition 5.4 (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 5.1 (L-Shape Based Layout Fracturing). Given an input layout specified by polygons, our goal is to fracture it into a set of L-shapes and/or rectangles. We also wish to minimize both the number of shots and the silver length of fractured shots.

5.2.2 Rectangular Merging (RM) Algorithm

Given the rectangles generated by any rectangular fracturing methodology, we propose an algorithm called RM that will merge them into a set of L-shapes. The key idea is that two rectangles that share a common coordinate can be combined into one L-shape. Although this idea is straightforward, the benefit is obvious: the previous rectangular fracturing algorithms can then be re-used. The RM algorithm is also used as a baseline in comparison with our other algorithm, DLF, which will be described in Sect. 5.2.3.

Given the input rectangles, the RM algorithm can find the optimal L-shape merging solution. The shot count and sliver lengths are simultaneously minimized.

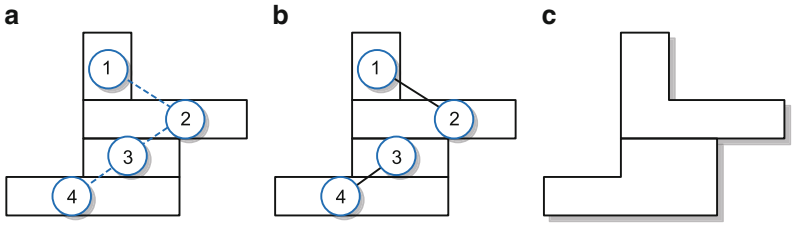

Fig. 5.6 Example of RM algorithm. (**a**) Graph construction. (**b**) Maximum matching result. (**c**) Corresponding rectangular merging

We first 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. Figure 5.6 shows an example where four rectangles are generated after rectangular fracturing. The constructed merging graph G is illustrated in Fig. 5.6a, where the three edges show that there are three ways to generate L-shapes. L-shape merging can thus 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 endpoint. For example, rectangle 2 can only belong to one L-shape. Only one edge can then be chosen between edges $\bar{12}$ and $\bar{23}$.

By utilizing the merging graph, the best edge selection can be solved by finding a maximum matching. The rectangular merging can therefore be formulated as a maximum matching problem. In the case of Fig. 5.6, the result of the maximum matching is illustrated in Fig. 5.6b, and the corresponding L-shape fracturing result is shown in Fig. 5.6c.

To take sliver minimization into account, we assign weights to the edges to represent whether the merging would remove one sliver. For example, if there is still one sliver even after two rectangles v_i and v_j are merged into one L-shape, we assign a smaller weight to edge e_{ij}. Otherwise, a larger weight is assigned. The rectangular merging can thus be formulated as maximum weighted matching. Even in general graphs, the maximum weighted matching can be solved in $O(nm \log n)$ time [41], where n is the number of vertices and m the number of edges in G.

5.2.3 *Direct L-Shape Fracturing (DLF) Algorithm*

Although the RM algorithm described above can provide the optimal merging solution for a given set of rectangles, it may suffer from several limitations. The polygon is first fractured into rectangles and then merged. This strategy, however, has some redundant operations. In Fig. 5.3, instead of a complex rectangle generation, one cut is sufficient for the L-shape fracturing. The rectangular fracturing may also ignore some internal features of L-shape fracturing, which could affect overall

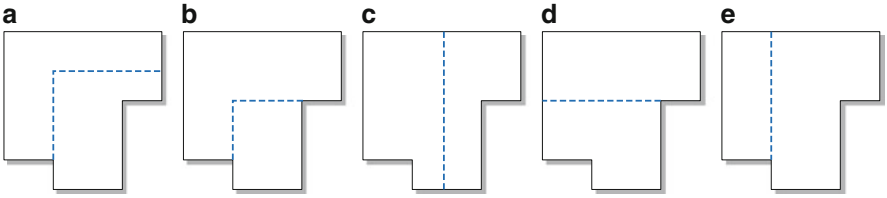

Fig. 5.7 Five fracturing solutions with the same shot count

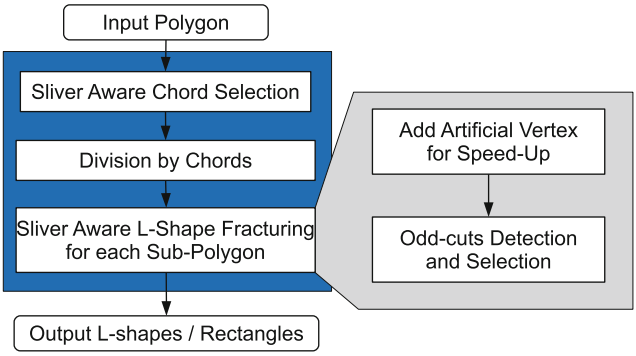

Fig. 5.8 Overall flow of DLF algorithm

performance. To overcome both of these limitations, we propose a novel algorithm called DLF to directly fracture a 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. 5.7, the input polygon has at least five different fracturing solutions with two shots.

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 for an appropriate odd-cut to decompose the polygon further. It was shown in [40] that an odd-cut always exists and that $\lfloor c/2 \rfloor + 1$ "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 5.1. *A polygon with c concave vertices can be decomposed into N L-shapes with an upper bound $N_{up} = \lfloor c/2 \rfloor + 1$.*

Figure 5.8 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 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. We differentiate between chords and cuts during polygon fracturing because chords are special cuts whose endpoints were both concave points in the original polygon. This helps us to design more efficient algorithms for odd cut/chord detection.

Sliver Aware Chord Selection

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

Lemma 5.2. *Decomposing a polygon by cutting through a chord does not increase the L-shape upper bound number N_{up}.*

Proof. Cut a polygon along a 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 5.1 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 [20, 21]. For L-shape fracturing, we will specifically select odd-chords, since they can reduce the number of L-shapes.

Lemma 5.3. *Decomposing an 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 5.2. 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 5.3 provides a guideline to select chords. An example is illustrated in Fig. 5.9, which contains two chords $\bar{bh}$ and $\bar{hk}$. Since the number of concave vertices to both sides of chord $\bar{bh}$ are odd (1), $\bar{bh}$ is an odd-chord. Cut along $\bar{bh}$, as shown in Fig. 5.9a, can achieve two L-shots. However, a cut along another chord $\bar{hk}$, which is not an odd-chord, would create three shots. Note that, in an odd polygon, although all chords are odd, cutting along them may not reduce N_{up}, but N_{up} is *guaranteed* not to increase.

For any even polygon P, we propose the following odd-chord search procedure. Each vertex v_i is assigned 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 the current vertex v_i.

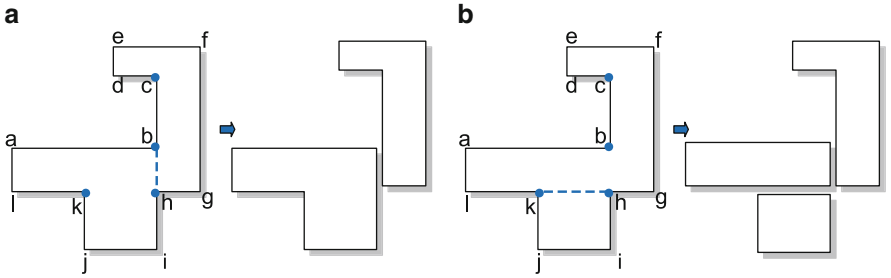

Fig. 5.9 Examples to illustrate Lemma 5.3. (**a**) Cut along odd-chord $\bar{bh}$ results in two L-shape shots. (**b**) Cut along chord $\bar{hk}$ would cause one more shot

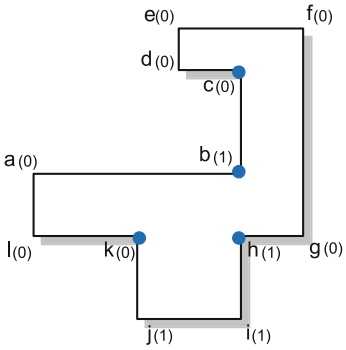

Fig. 5.10 To detect odd-chords in even polygon, each vertex is associated with one Boolean parity

Otherwise p_j is assigned to p_i. This parity assignment can be completed during one clockwise traversal in $O(n)$ time. An example of a parity assignment starting from $a(0)$ is shown in Fig. 5.10, where each vertex is associated with one parity.

Theorem 5.1. *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 5.1. For each concave vertex v_i, a plane sweep is applied to search for any chord containing v_i. The plane sweep for each vertex can be finished in $O(\log n)$, and the number of vertices is $O(n)$. Therefore, in an even polygon, odd-chords detection can be completed in $O(n \log n)$ 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. 5.10, chords $\bar{bh}$ and $\bar{hk}$ cannot be selected simultaneously. The relationship among the chords can be represented as a bipartite graph [20], where the vertices in the 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. This can be reduced to the maximum matching problem,

which can be performed in polynomial time. It should be noted that if the input polygon is an even polygon, because of Theorem 5.1, we preferentially choose odd-chords. The bipartite graph is thus modified by assigning weights to the edges. Sliver minimization is also 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 a sliver, so we will discard this candidate.

Sliver Aware L-Shape Fracturing

After chord selection, the input polygon P is decomposed into m sub-polygons (denoted as $P_1, P_2, \ldots, P_m$). For each polygon P_i, we will recursively fracture it until we are only left with L-shapes and/or rectangles. Our fracturing algorithm is based on odd-cut selection: we pick an odd-cut at every step and decompose the polygon into two pieces through this odd-cut. This iteratively fractures the polygon into several L-shapes. Note that our fracturing algorithm considers sliver minimization, i.e., we try to minimize the sliver length during fracturing.

The first question is how to detect all of 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 and assign each vertex v_i 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 for both horizontal and vertical cuts originating 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, we can check if it is an odd-cut in constant time using the following Theorem 5.2.

Theorem 5.2. *In an odd polygon, a cut* $(a, \bar{bc})$ *is an odd-cut if and only if the following condition is satisfied:*

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

The detailed proof is omitted due to space constraints. An example of odd-cut detection is shown in Fig. 5.11. There are three concave vertices in the odd polygon: v_b, v_f, and v_i. Start from each concave vertex and search all six cuts. Applying Theorem 5.2, we find two odd-cuts $(b, \bar{fg})$ and $(i, \bar{cd})$.

Algorithm 15 describes our L-shape fracturing in detail. Given the input polygon P, if it is already an L-shape or rectangle, then the fracturing is complete. Otherwise, we find all odd-cuts (line 5). We choose one odd-cut cc from the set of all detected odd-cuts, and we cut the P into two pieces P_1 and P_2 (lines 10–11). Lastly, we recursively apply L-shape fracturing to P_1 and $P2$ (lines 12–13).

Algorithm 15 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($P2$);

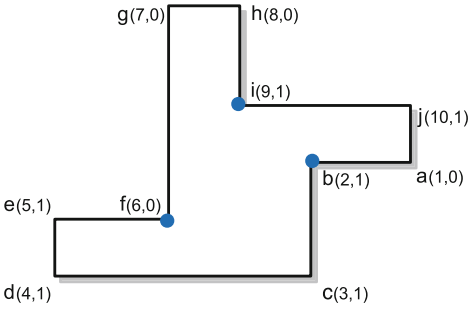

Fig. 5.11 Odd-cut detection using order number and parity

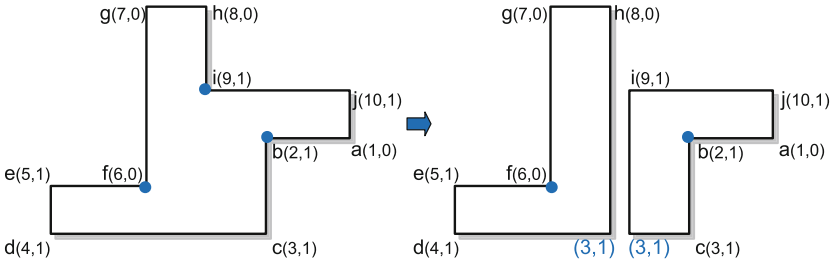

Fig. 5.12 Only one vertex and four edges need to be updated during polygon decomposition

Note that during the polygon decomposition, we do not need to recalculate the order number and parity for each vertex. Instead, when a polygon is divided into two parts, we only update one vertex and four edges, maintaining all other information. If polygon P is cut using odd-cut $(a, \bar{bc})$, a new vertex 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 $\bar{dc}$. Two edges $\bar{ad}$ and $\bar{da}$ are then inserted. The update method is simple and easy to implement. An example of such an update is shown in Fig. 5.12.

Sliver minimization is integrated into our L-shape fracturing algorithm. In Algorithm 15, when picking one cut from all detected odd-cuts, we try to avoid

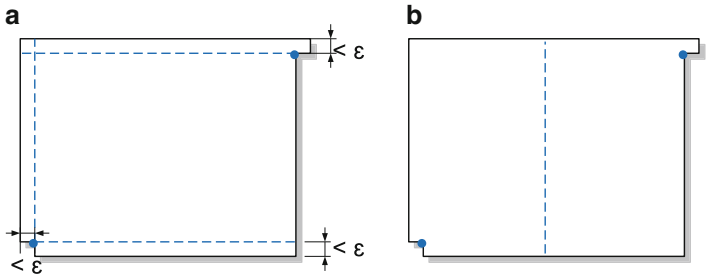

Fig. 5.13 Auxiliary cut generation. (**a**) Here every odd-cut would cause sliver. (**b**) Decompose through on auxiliary cut can avoid sliver

creating any slivers. For example, as illustrated in Fig. 5.13a, there are three odd-cuts, but all of them would create a sliver. Instead of selecting one of these, we generate an auxiliary cut in the middle (see Fig. 5.13b). The resulting polygon can be fractured without introducing a sliver. In general, if there are several odd-cuts that do not cause slivers, we pick the cut using the following rules: (1) We prefer the cut that partitions the polygon into two balanced sub-polygons; (2) If the polygon is more horizontal than vertical, we prefer a vertical cut, and vice versa.

Given a polygon with n vertices, finding all concave vertices takes $O(n)$ time. For each concave vertex v_i, searching for cut that starts there requires $O(\log n)$ time. Due to Theorem 5.2, checking whether a cut is an odd-cut can be performed in $O(1)$, thus finding all odd-cuts can be finished in $O(n \log n)$ time. Note that given a polygon with c concave vertices, if no auxiliary cut is generated, the L-shape fracturing can be completed using $\lfloor c/2 \rfloor$ odd-cuts. When auxiliary cuts are applied, there are at most $c - 1$ cuts made that fracture the input polygon. This leads to the following theorem.

Theorem 5.3. *The sliver aware L-shape generation can find a set of L-shapes in* $O(n^2 \log n)$ *time.*

Note that if our objective is only to minimize the shot number, no auxiliary cuts would be introduced. Thus, at most $\lfloor c/2 \rfloor + 1$ L-shapes would be generated. In other words, the shot number would be bounded by the theoretical upper bound N_{up}.

Speed-Up Technique

We observe that in practice during the execution of Algorithm 15, many odd-cuts do not intersect. This implies that multiple 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 number of concave

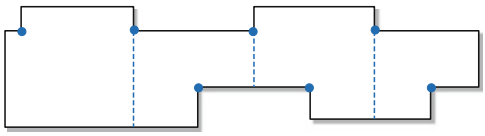

Fig. 5.14 Speed-up for odd polygon, where all three odd-cuts are compatible

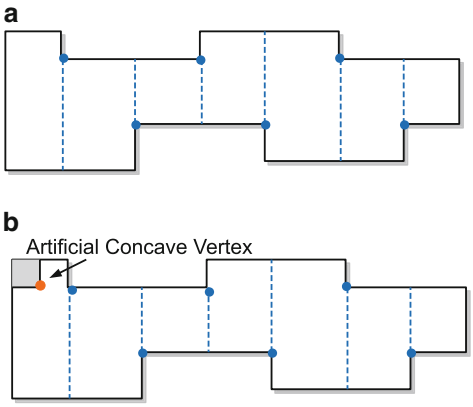

Fig. 5.15 Speed-up for even polygon. (**a**) All cuts are odd-cuts. (**b**) Introducing one artificial concave vertex translates the even polygon into an odd polygon

vertices on each side. Partitioning the polygon along such an odd-cut leaves all other remaining odd-cuts as odd-cuts. For example, Fig. 5.14a shows an odd polygon, where all three odd-cuts are compatible, and can be chosen simultaneously. Through fracturing the polygon along the three odd-cuts, the L-shape fracturing problem is resolved in one step.

However, this speed-up technique cannot be directly applied to an even polygon, since it may actually increase the shot number. This is because an even polygon does not guarantee that odd-cuts will remain odd-cuts in the resulting sub-polygons. For example, in Fig. 5.15a, 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, we introduce one artificial concave vertex for each even-polygon. This artificial concave vertex converts the polygon into an odd polygon. As shown in Fig. 5.15b, in the resulting odd polygon, all compatible odd-cuts can be used for fracturing without increasing the shot number. Because of Lemma 5.4, this translation is guaranteed not to increase the total shot number.

Lemma 5.4. *Introducing one artificial concave vertex to an even polygon does not increase the L-shape upper bound* N_{up}.

When employing this speed-up technique, for most cases, the odd-cut detection only needs to be performed once, so the DLF algorithm can generally be completed in $O(n \log n)$ time in practice.

5.2.4 Experimental Results

We implemented our two L-shape fracturing algorithms, RM and DLF, in C++. Since RM needs a rectangular fracturing method to generate the initial rectangles, we implemented a state-of-the-art algorithm proposed in [21]. Based on the generated rectangles, the RM algorithm was applied to merge them into a set of L-shapes. LEDA package [42] was adopted for the maximum weighted matching algorithm.

The experiments are performed on an Intel Xeon 3.0 GHz Linux machine with a 32 GB RAM. ISCAS 85&89 benchmarks are scaled down to ta 28 nm logic node, followed by accurate lithographic simulations performed on 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 [43] was adopted for interfacing.

Table 5.1 shows the results of our DLF algorithm in comparison with the approaches in [21] and the RM algorithm. Since the framework [21] is adopted to provide the input rectangles, the RM algorithm is denoted as "[21]+RM." Column "poly#" lists the number of polygons for each test circuit. All fracturing methods are evaluated with the sliver parameter $\epsilon = 5$ nm. 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 [21] to 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 increase is reasonable: the RM algorithm introduces a 41 % increases in the runtime. We then compare our DLF algorithm with the other two methods. We can see that DLF performs optimally, in terms of runtime and performance. Compared with traditional sliver-aware rectangular fracturing [21], it can achieve approximately a 9× speed-up. The shot number and sliver length are also significantly reduced (39 % and 82 %, respectively). Even compared with the RM algorithm, DLF is better in terms of performance: its shot number and the sliver length are lower by 3.2 % and 67 %, respectively.

In order to evaluate the scalability of our algorithm, we summarize all the runtimes from Table 5.1 and display in Fig. 5.16. Here the X-axis denotes the number of polygons (e.g., the problem size), while the Y-axis shows the runtime. We can see that DLF algorithm scales better than both [21] and RM algorithm.

5.3 OSP for MCC System

To overcome the throughput limitation, recently multi-column cell (MCC) system is proposed as an extension to CP technique [44, 45]. 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

Table 5.1 Runtime and performance comparisons

Circuits	Poly#	[21]			[21]+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	13,397	96.0	16.9	8325	45.0	19.5	8112	11.8	2.6
C880	2411	17,586	160.5	24.93	11,020	84.4	29.7	10,653	28.6	3.8
C1355	3262	23,283	185.2	29.44	14,555	87.8	33.6	13,936	24.8	5.1
C1908	5125	35,657	333.6	48.78	22,352	181.1	57.3	21,540	88.0	7.68
C2670	7933	56,619	525.4	84.11	35,424	274.4	96.9	34,102	114.8	11.89
C3540	10,189	74,632	668.5	114.33	46,617	360.0	133.7	44,901	129.8	15.98
C5315	14,603	108,761	950.4	176.89	67,795	488.9	200.8	65,222	190.2	23.85
C6288	14,575	103,148	819.2	175.65	64,987	382.0	201.0	62,416	86.1	22.64
C7552	21,253	151,643	1334.6	242.77	94,902	717.6	280.0	91,157	290.7	32.02
S1488	4611	37,126	303.7	55.03	22,984	146.2	64.7	22,099	31.6	8.14
S38417	67,696	454,307	4040.2	727.1	285,049	2293.0	1020	275,054	729	88.5
S35932	26,267	163,956	1470.4	228.02	103,960	808.3	256.4	100,629	284	34.85
S38584	168,319	1,096,363	10,045.2	2268.6	690,054	5777.0	3565.2	666,906	1801.7	216.39
S15850	34,660	231,681	2012.8	329.99	145,745	1085.1	414.6	140,879	320	44.7
Avg.	–	171,670	1533.0	302.1	107,876	850.3	425.6	104,121	275.9	34.7
Ratio	–	1	1	1	0.63	0.55	1.41 1	**0.61**	**0.18**	**0.11**

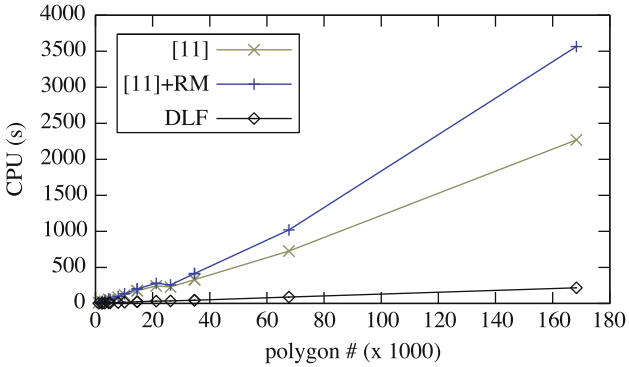

Fig. 5.16 Comparison on algorithm scalability

to achieve better throughput. In modern MCC system, there are more than 1300 character projections (CPs) [46]. Since one CP is associated with one stencil, there are more than 1300 stencils in total. The manufacturing of stencil is similar to mask manufacturing. If each stencil is different, then the stencil preparation process could be very time consuming and expensive. Due to the design complexity and cost consideration, different CPs share one stencil design. One example of MCC printing process is illustrated in Fig. 5.17, 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. The whole writing time of the MCC system is determined by the maximum one among 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.

Stencil planning is one of the most challenging problems in CP mode, and has earned more and more attentions. When blank overlapping is not considered, this problem equals to a character selection problem, which can be solved through an integer linear programming (ILP) formulation to maximize the system throughput [47]. When the characters can be overlapped to save more stencil space, the corresponding stencil planning is referred to as *overlapping-aware stencil planning* (OSP). As suggested in [34], the OSP problem can be divided into two types: one dimensional (1D) problem and two dimensional (2D) problem.

In the one dimensional OSP problem, the standard cells with same height are selected into the stencil. As shown in Fig. 5.18a, 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. Yuan et al. [34] proposed a set of heuristics, and the single row reordering was formulated as minimum cost Hamiltonian path. Kuang and Young [48] proposed an integrated framework to solve all the sub-problems effectively: character selection, row distribution, single-row

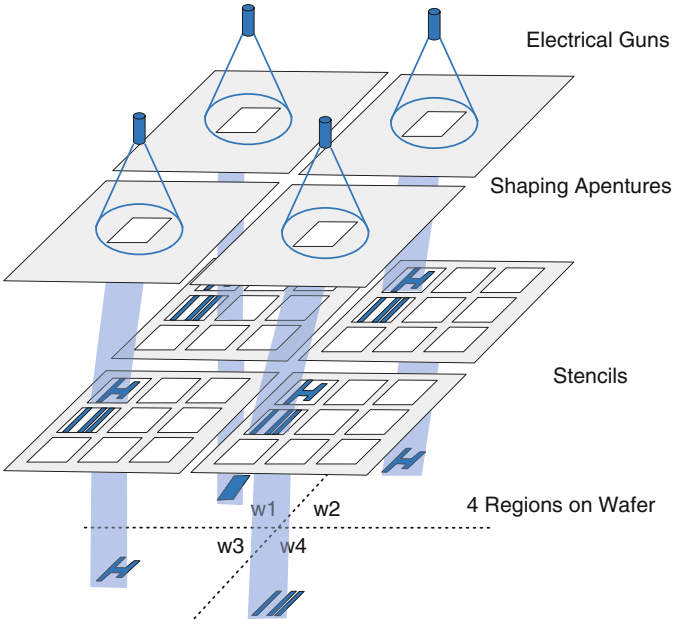

Fig. 5.17 Printing process of MCC system, where four CPs are bundled

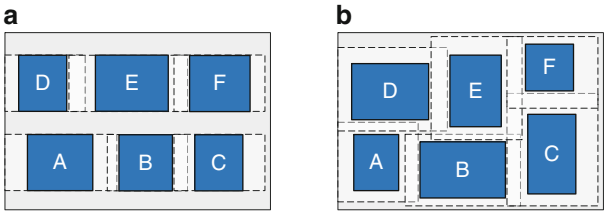

Fig. 5.18 Two types of OSP problem: (**a**) one dimensional problem; (**b**) two dimensional problem

ordering, and inter-row swapping. Guo et al. [49] proved that single row reordering can be optimally solved in polynomial time. In addition, [50, 51] assumed that the pattern position in each character can be shifted, and integrated the character redesign into OSP problem.

In the two dimensional OSP problem, the blank 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 (see Fig. 5.18b for an example). Yuan et al. [34] solved the problem through a modified floorplanning engine, while [52] further speed-up the engine through clustering technique. Kuang and Young [53] proposed a set of fast and effective deterministic methods to solve this problem.

Compared with conventional EBL system, MCC system introduces two main challenges in OSP problem. (1) 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 time of all regions. However, in conventional EBL system the objective is simply to minimize the wafer writing time. (2) The stencil for an MCC system can contain more than 4000 characters, and 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 a powerful tool, E-BLOW, to overcome both the challenges. 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 the best of our knowledge, this is the first mathematical formulation for both 1D-OSP and 2D-OSP.
- We propose a simplified formulation for 1D-OSP, and prove 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 speed up 2D-OSP solution.

5.3.1 Problem Formulation

In this section, we give the problem formulation. For convenience, Table 5.2 lists the notations used in this paper. Note that in this section, we denote $[n]$ as a set of integers $\{1, 2, \ldots, n\}$.

In an MCC system with K CPs, the whole wafer is divided into K regions, and each region is written by one particular CP. We assume cell extraction [12] has been resolved first. In other words, a set of n character $C = \{c_1, \ldots, c_n\}$ has already been given to the MCC system. For each character $c_i \in C$, its width is w_i. Meanwhile, the writing time through VSB mode and CP mode are a_i and 1, respectively. The stencil

Table 5.2 Notations used in this paper

Term	Meaning	Term	Meaning
W	Width of each stencil row	K	Number of regions
T_k	System writing time on region k	T	Total writing time of all regions
m	Number of rows	n	Number of characters
w	Width of each character	C	Set of characters $C = \{c_1, \ldots, c_n\}$
a_i	Character c_i's writing time in VSB mode	x_i	x-position of character c_i
sl_i	Left blank of character c_i	sr_i	Right blank of character c_i
s_i	$\lceil (sl_i + sr_i)/2 \rceil$	b_{ij}	0–1 variable, $b_{ij} = 1$ if c_i is on row j
t_{ik}	Repeating times of character c_i in region k	o_{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

is divided into m rows, and the width of each row is W. For each row j, b_{ij} indicates whether character c_i is selected on row j:

$$b_{ij} = \begin{cases} 1, & \text{candidate } c_i \text{ is selected on row } j \\ 0, & \text{otherwise} \end{cases}$$

Different regions have different layout patterns, thus the throughputs would also be different. For region k ($k \in [K]$), character $c_i \in C$ repeats t_{ik} times. If c_i is prepared on stencil, the total writing time of character c_i on region r_c is $t_{ik} \cdot 1$. Otherwise, c_i should be printed through VSB, and its writing time would be $t_{ik} \cdot a_i$. Therefore, for region k ($k \in [K]$) the writing time T_k is as follows:

$$T_k = \sum_{i \in [n]} t_{ik} \cdot a_i - \sum_{i \in [n]} \sum_{j \in [m]} b_{ij} \cdot t_{ik} \cdot (a_i - 1) \tag{5.1}$$

where the first term is the writing time using VSB mode, while the second term is the writing time improvement through CP mode. Therefore, the total writing time of the MCC system is formulated as follows:

$$T = \max_{k \in [K]} \{T_k\} \tag{5.2}$$

Based on the notations above, we define OSP for MCC system problem as follows.

Problem 5.2 (OSP for MCC System). In an MCC system, given a stencil with m row, and each row is with width W. A set of character C is also given. We select a subset of characters in C, and place them on the stencil. The objective is to minimize the MCC system writing time T expressed by Eq. (5.2), while the placement of all characters is bounded by the outline of stencil.

For convenience, we use the term OSP to refer to OSP for MCC system in the rest of this chapter. Note that when region number K is 1, then MCC system equals to conventional EBL system. Therefore, our proposed methodologies can also handle the OSP problem for conventional EBL system.

5.3.2 *Proof of NP-Hardness*

In this section we prove that both 1D-OSP and 2D-OSP problems are NP-hard. Guo et al. [49] proves that a sub-problem of 1D-OSP, single row ordering, can be optimally solved in polynomial time. Mak and Chu [51] proves that when an additional problem, character re-design, is integrated, the extended 1D-OSP problem is NP-hard. Although we know that a sub-problem of 1D-OSP is in P [49], while an extended problem of 2D-OSP is NP-hard [51], the complexity of 1D-OSP

itself is still an open question. This is the first work proving the complexity of 1D-OSP problem. In addition, we will show that even a simpler version of 1D-OSP problem, where there is only one row, is NP-hard as well.

BSS Problem Is NP-Complete

To facilitate the proof, we first define a bounded subset sum (BSS) problem as follows.

Problem 5.3 (Bounded Subset Sum). Given a list of n numbers $\{x_1, \ldots, x_n\}$ and a number s, where $\forall i \in [n]\ 2 \cdot x_i > x_{\max}(\triangleq \max_{i \in [n]} |x_i|)$, decide if there is a subset of the numbers that sums up to s.

For example, given three numbers $1100, 1200, 1413$ and $T = 2300$, we can find a subset $\{1100, 1200\}$ such that $1100 + 1200 = 2300$. Additionally, we have the assumption that $t > c \cdot x_{\max}$, where c is some constant. Otherwise it be solved in $O(n^c)$ time. Besides, without the bounded constraint $\forall i \in [n]\ 2 \cdot x_i > x_{\max}$, the BSS problem becomes a **Subset sum** problem, which is in NP-complete [54]. For simplicity of later explanation, let S denote the set of n numbers. Note that, we can assume that all the numbers are integer numbers.

Lemma 5.5. *BSS problem is in NP.*

Proof. It is easy to see that BSS problem is in NP. Given a subset of integer numbers $S' \in S$, we can add them up and verify that their sum is s in polynomial time.

We further prove BSS problem is NP-hard through showing that it can be reduced from 3SAT problem, which is NP-complete [54].

Lemma 5.6. *3SAT* $\leq_p$ *BSS.*

Proof. In 3SAT problem, we are given m clauses $\{C_1, C_2, \ldots, C_m\}$ over n variables $\{y_1, y_2, \ldots, y_n\}$. Besides, there are three literals in each clause, which is the OR of some number of literals. Equation (5.3) gives one example of 3SAT, where $n = 4$ and $m = 2$:

$$(y_1 \vee \bar{y}_3 \vee \bar{y}_4) \wedge (\bar{y}_1 \vee y_2 \vee \bar{y}_4) \tag{5.3}$$

Without loss of generality, we can have the following assumptions:

1. No clause contains both variable y_i and $\bar{y}_i$. Otherwise, any such clause is always true and we can just eliminate them from the formula.
2. Each variable y_i appears in at least one clause. Otherwise, we can just assign any arbitrary value to the variable y_i.

To convert a 3SAT instance to a BSS instance, we create two integer numbers in set S for each variable y_i and three integer numbers in S for each clause C_j. All the

numbers in set S and s are in base 10. Besides, $10^{n+2m} < y_i < 2 \cdot 10^{n+2m}$, so that the bounded constraints are satisfied. All the details regarding S and s are defined as follows:

- In the set S, all integer numbers are with $n + 2m + 1$ digits, and the first digit is always 1.
- In the set S, we construct two integer numbers t_i and f_i for the variable y_i. For both of the values, the n digits after the first "1" serve to indicate the corresponding variable in S. That is, the ith digit in these n digits is set to 1 and all others are 0. For the next m digits, the jth digit is set to 1 if the clause C_j contains the respective literal. The last m digits are always 0.
- In the set S, we also construct three integer numbers c_{j1}, c_{j2}, and c_{j3} for each clause C_j. In c_{jk} where $k = \{1, 2, 3\}$, the first n digits after the first "1" are 0, and in the next m digits all are 0 except the jth index setting to k. The last m digits are all 0 except the jth index setting to 1.
- $T = (n + m) \cdot 10^{n+2m} + s_0$, where s_0 is an integer number with $n + 2m$ digits. The first n digits of s_0 are 1, in the next m digits all are 4, and in the last m digits all are 1.

Based on the above rules, given the 3SAT instance in Eq. (5.3), the constructed set S and target s are shown in Fig. 5.19. Note that the highest digit achievable is 9, meaning that no digit will carry over and interfere with other digits.

Claim. The 3SAT instance has a satisfying truth assignment iff the constructed BSS instance has a subset that adds up to s.

Proof of ⇒ Part of Claim If the 3SAT instance has a satisfying assignment, we can pick a subset containing all t_i for which y_i is set to true and f_i for which y_i is set to false. We should then be able to achieve s by picking the necessary c_{jk} to get 4's

Fig. 5.19 The constructed BSS instance for the given 3SAT instance in (5.3)

		y_1	y_2	y_3	y_4	C_1	C_2	A_1	A_2
t_1	= 1	1	0	0	0	1	0	0	0
f_1	= 1	1	0	0	0	0	1	0	0
t_2	= 1	0	1	0	0	0	1	0	0
f_2	= 1	0	1	0	0	0	0	0	0
t_3	= 1	0	0	1	0	0	0	0	0
f_3	= 1	0	0	1	0	1	0	0	0
t_4	= 1	0	0	0	1	0	0	0	0
f_4	= 1	0	0	0	1	1	1	0	0
c_{11}	= 1	0	0	0	0	1	0	1	0
c_{12}	= 1	0	0	0	0	2	0	1	0
c_{13}	= 1	0	0	0	0	3	0	1	0
c_{21}	= 1	0	0	0	0	0	1	0	1
c_{22}	= 1	0	0	0	0	0	2	0	1
c_{23}	= 1	0	0	0	0	0	3	0	1
s	= 6	1	1	1	1	4	4	1	1
s_0	=	1	1	1	1	4	4	1	1

in the s. Due to the last m "1" in s, for each $j \in [m]$ only one would be selected from $\{c_{j1}, c_{j2}, c_{j3}\}$. Besides, we can see totally $n + m$ numbers would be selected from S.

Proof of $\Leftarrow$ Part of Claim If there is a subset $S' \in S$ that adds up to s, we will show that it corresponds to a satisfying assignment in the 3SAT instance. S' must include exactly one of t_i and f_i, otherwise the ith digit value of s_0 cannot be satisfied. If $t_i \in S'$, in the 3SAT we set y_i to true; otherwise we set it to false. Similarly, S' must include exactly one of c_{j1}, c_{j2}, and c_{j3}, otherwise the last m digits of s cannot be satisfied. Therefore, all clauses in the 3SAT are satisfied and 3SAT has a satisfying assignment.

For instance, given a satisfying assignment of Eq. (5.3): $\langle y_1 = 0, y_2 = 1, y_3 = 0, y_4 = 0\rangle$, the corresponding subset S' is $\{f_1 = 110000100, t_2 = 101000100, f_3 = 100101000, f_4 = 100011100, c_{12} = 100002010, c_{21} = 100000101\}$. We set $s = (m+n) \cdot 10^{n+2m} + s_0$, where $s_0 = 11114411$, and then $s = 611114411$. We can see that $f_1 + t_2 + f_3 + f_4 + s_{12} + s_{21} = s$.

Combining Lemmas 5.5 and 5.6, we can achieve the following theorem.

Theorem 5.4. *BSS problem is NP-complete.*

OSP Problem Is NP-Hard

In the following, we will show that even a simpler version of 1D-OSP problem is NP-hard. In the simpler version, there is only one row in the stencil, and each character $c_i \in C$ is with the same length w. Besides, for each character, its left blank and right blank are symmetric.

Definition 5.5 (Minimum Packing). Given a subset of characters $C' \in C$, its minimum packing is the packing with the minimum stencil length.

Lemma 5.7. *Given a set of character $C = \{c_1, c_2, \ldots, c_n\}$ placed on a single row stencil. If for each character $c_i \in C$, both of its left and right blanks are s_i, then the minimum packing is with the following stencil length*

$$n \cdot w - \sum_{i \in [n]} s_i + \max_{i \in [n]} \{s_i\} \tag{5.4}$$

Proof. Without loss of generality, we assume that $s_1 \geq s_2 \geq \cdots \geq s_n$. We prove by recursion that in an minimum length packing, the overlapping blank is $f(n) = \sum_{i=2}^{n} s_i$. If there are only two characters, it is trivial that $f(2) = s_2$. We assume that when $p = n - 1$, the maximum overlapping blank $f(n-1) = \sum_{i=2}^{n-1} s_i$. For the last character c_n, the maximum sharing blank value is s_n. Since for any $i < n$, $s_i \geq s_n$, we can simply insert it at either the left end or the right end, and find the incremental overlapping blank s_n. Thus $f(n) = f(n-1) + s_n = \sum_{i=2}^{n} s_i$. Because the maximum overlapping blank for all characters is $\sum_{i=2}^{n} s_i$, we can see the minimum packing length is as in Eq. (5.4).

Lemma 5.8. *BSS* $\leq_p$ *1D-OSP.*

Proof. Given an instance of BSS with s and $S = \{x_1, x_2, \ldots, x_n\}$, we construct an 1D-OSP instance as follows:

- The stencil length is set to $M + s$, where $M = \max_{i \in [n]} \{x_i\}$.
- For each $x_i \in S'$, in 1D-OSP there is a character c_i, whose width is M and both left and right blanks are $M - x_i$. Since $x_i > M/2$, the sum of left blank and right blank is less or equal to M.
- We introduce an additional character c_0, whose width size is M, and both left and right blanks are $M - \min_{i \in [n]} \{x_i\}$.
- The VSB writing time of character c_0 is set to $\sum_{i \in [n]} x_i$, while the VSB writing time for each character c_i is set to x_i. The CP writing times are set to 0 for all characters.
- There is only one region, and each character c_i repeats one time in the region.

For instance, given initial set $S = \{1100, 1200, 2000\}$ and $s = 2300$, the constructed 1D-OSP instance is shown in Fig. 5.20.

We will show the BSS instance $S = \{x_1, x_2, \ldots, x_n\}$ has a subset that adds up to s if and only if the constructed 1D-OSP instance has minimum packing length $M + s$ and total writing time smaller than $\sum x_i$.

($\Rightarrow$ part) After solving the BSS problem, a set of items S' are selected that they add up to s. For each $x_i \in S'$, the corresponding character c_i is also selected into the stencil. Besides, since the system writing time for c_0 is $\sum x_i$, it is trivial to see that in the 1D-OSP instance the c_0 must be selected. Due to the Lemma 5.7, the minimum total packing length is:

$$(n+1) \cdot M - \sum_{i \in S'} (M - x_i) = M + \sum_{i \in S'} x_i = M + s$$

Meanwhile, the minimum total writing time in the 1D-OSP is $\sum_{i \in [n]} x_i - s$.

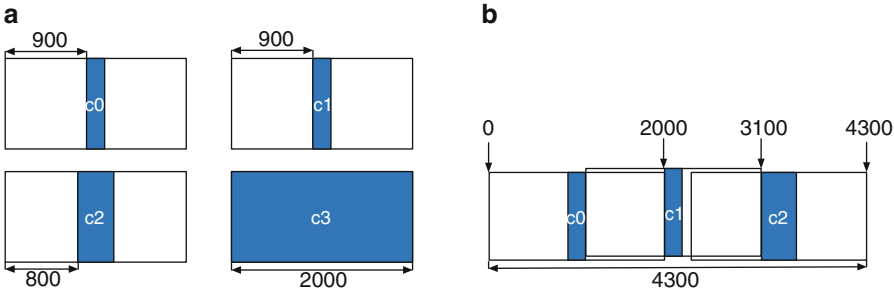

Fig. 5.20 (**a**) 1D-OSP instance for the BSS instance $S = \{1100, 1200, 2000\}$ and $s = 2300$. (**b**) The minimum packing is with stencil length $M + s = 2000 + 2300 = 4300$

($\Leftarrow$ part) We start from an 1D-OSP instance with minimum packing length $M+s$ and total writing time smaller than $\sum x_i$, where a set of character $C' \in C$ is selected. Since the total writing time must be smaller than $\sum x_i$, character $c_0 \in C'$. For all characters in set $c_i \in C'$ except c_0, we select x_i into the subset $S' \in S$, which adds up to s.

Theorem 5.5. *1D-OSP is NP-hard.*

Proof. Directly from Lemma 5.8 and Theorem 5.4.

1D-OSP problem is a special case of 2D-OSP problem, when all the characters share the same height. Therefore, from Theorem 5.5 we can see that 2D-OSP problem is NP-hard as well.

5.3.3 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 [34]. Different from two-step heuristic proposed in [34], we show that these two problems can be solved simultaneously through a unified ILP formulation (5.5).

$$\min \quad T \tag{5.5}$$

$$\text{s.t} \quad T \geq \sum_{i\in[n]} t_{ik}\cdot a_i - \sum_{i\in[n]}\sum_{j\in[m]} b_{ij}\cdot t_{ik}(a_i-1) \qquad \forall k \in [K] \tag{5.5a}$$

$$0 \leq x_i \leq W - w \qquad \forall i \in [n] \tag{5.5b}$$

$$\sum_{j\in[m]} b_{ij} \leq 1 \qquad \forall i \in [n] \tag{5.5c}$$

$$x_i + w_{ii'} - x_{i'} \leq W(2 + p_{ii'} - b_{ij} - b_{i'j}) \qquad \forall i, i' \in [n], j \in [m] \tag{5.5d}$$

$$x_{i'} + w_{i'i} - x_i \leq W(3 - p_{ii'} - b_{ij} - b_{i'j}) \qquad \forall i, i' \in [n], j \in [m] \tag{5.5e}$$

$$b_{ij}, b_{i'j}, p_{ii'} \in \{0,1\} \qquad \forall i, i' \in [n], j \in [m] \tag{5.5f}$$

In formulation (5.5), W is the stencil width and 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 row j, $b_{ik} = 1$. Otherwise, $b_{ij} = 0$.

Constraint (5.5a) is derived from Eqs. (5.1) and (5.2). Constraint (5.5b) is for the x-position of each character. Constraint (5.5c) is to make sure one character can be inserted into at most one row. Constraints (5.5d), (5.5e) are used to check position relationship between c_i and c_i'. Here $w_{ii'} = w_i - o_{ii'}$ and $w_{i'i} = w_{i'} - o^h_{i'i}$, where $o_{ii'}$ is the overlapping when candidates c_i and c_i' are packed together. Only when

$b_{ij} = b_{i'j} = 1$, i.e., both character c_i and character $c_{i'}$ are assigned to row j, one of the two constraints (5.5d), (5.5e) will be active. Besides, all the $p_{ii'}$ values are self-consistent. For example, for any three characters c_1, c_2, c_3 being assigned to row j, i.e., $b_{1j} = b_{2j} = b_{3j} = 1$, if c_1 is on the left of c_2 ($p_{12} = 0$) and c_2 is on the left of c_3 ($p_{23} = 0$), then c_1 should be on the left of c_3 ($p_{13} = 0$). Similarly, if c_1 is on the right of c_2 ($p_{12} = 1$) and c_2 is on the right of c_3 ($p_{23} = 1$), then c_1 should be on the right of c_3 ($p_{13} = 1$) as well.

Since ILP is a well-known NP-hard problem, directly solving it may suffer from long runtime penalty. One straightforward speed-up method is to relax the ILP into the corresponding linear programming (LP) through replacing constraints (5.5f) by the following:

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

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, $\sum_j b_{ij} = 1$ and all the $p_{ii'}$ 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 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.

To overcome the limitation of above rounding, we propose a novel successive rounding framework, E-BLOW, to search for near-optimal solution in reasonable runtime. As shown in Fig. 5.21, the overall flow includes five parts: Simplified ILP formulation, Successive Rounding, Fast ILP Convergence, Refinement, and Post-Insertion. In Sect. 5.3.3 the simplified formulation will be discussed, and its LP rounding lower bound will be proved. In Sect. 5.3.3 the details of successive rounding would be introduced. In Sect. 5.3.3 the refinement process is proposed. At last, to further improve the performance, in Sect. 5.3.3 the post-swap and post-insertion techniques are discussed.

Simplified ILP Formulation

As discussed above, solving the ILP formulation (5.5) is very time-consuming, and the related LP relaxation may be bad in performance. To overcome the limitations of (5.5), in this section we introduce a simplified ILP formulation, whose LP relaxation can provide good lower bound. The simplified formulation is based on a *symmetrical blank* (S-Blank) assumption: the blanks of each character are symmetric, 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 $c_{i'}$, their blanks s_i and $s_{i'}$ can be different.

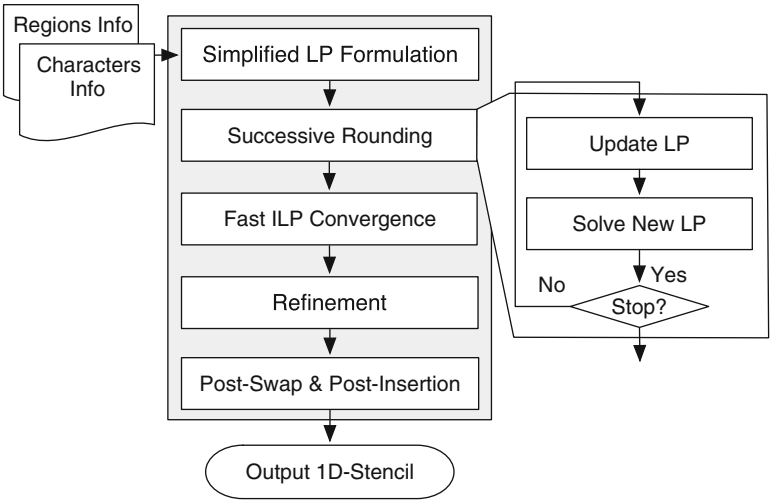

Fig. 5.21 E-BLOW overall flow for 1D-OSP problem

At first glance the S-Blank assumption may lose optimality. However, it provides several practical and theoretical benefits. (1) In [34] 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 [34], 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 Sect. 5.3.3).

The simplified ILP formulation is shown in Formula (5.6).

$$\max \sum_{i\in[n]}\sum_{j\in[m]} b_{ij}\cdot p_i \tag{5.6}$$

$$\text{s.t.} \sum_{i\in[n]}(w-s_i)\cdot b_{ij} \le W - B_j \qquad \forall j\in[m] \tag{5.6a}$$

$$B_j \ge s_i \cdot b_{ij} \qquad \forall i\in[n], j\in[m] \tag{5.6b}$$

$$\sum_{j\in[m]} b_{ij} \le 1 \qquad \forall i\in[n] \tag{5.6c}$$

$$b_{ij} = 0 \text{ or } 1 \qquad \forall i\in[n], j\in[m] \tag{5.6d}$$

In the objective function of Formula (5.6), each character c_i is associated with one profit value p_i. The p_i value is to evaluate the overall system writing time improvement if character c_i is selected. Through assigning each character c_i with one particular profit value, we can simplify the complex constraint (5.5a). More details regarding the profit value setting would be discussed in Sect. 5.3.3. Besides, due to Lemma 5.7, constraint (5.6a) and constraint (5.6b) are for row width calculation, where (5.6b) is to linearize max operation. Here B_j can be viewed as the maximum blank space of all the characters on row j. Constraint (5.6c) 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 (5.5), the variable number in (5.6) can be reduced dramatically.

In our implementation, we set blank value s_i to $\lceil (sl_i + sr_i)/2 \rceil$, where sl_i and sr_i are c_i's left blank and right blank, respectively. Note that here the ceiling function is used to make sure that under the S-Blank assumption, each blank is still integral. Although this setting may lose some optimality, E-BLOW provides post-stage to compensate the inaccuracy through incremental character insertion.

Now we will show that the LP relaxation of (5.6) has reasonable lower bound. To explain this, let us first look at a similar formulation (5.7) as follows:

$$\max \sum_{i\in[n]} \sum_{j\in[m]} b_{ij} \cdot p_i \tag{5.7}$$

$$\text{s.t.} \sum_{i\in[n]} (w - s_i) \cdot b_{ij} \leq W - s_{\max} \qquad \forall j \in [m] \tag{5.7a}$$

$$(5.6c) - (5.6d)$$

where s_{max} is the maximum horizontal blank length of every character, i.e.,

$$s_{max} = \max_{i\in[n]} \{s_i\}$$

Program (5.7) is a multiple knapsack problem [55]. A multiple knapsack is similar to a knapsack problem, with the difference that there are multiple knapsacks. In formulation (5.7), each p_i can be rephrased as $(w_i - s_i) \times ratio_i$.

Lemma 5.9. *If each $ratio_i$ is the same, the multiple knapsack problem (5.7) can find a 0.5-approximation algorithm using LP rounding method.*

For brevity we omit the proof, but detailed explanations can be found in [56]. When all $ratio_i$ are the same, Formulation (5.7) can be approximated to a max-flow problem. In addition, if we denote α as $\min\{ratio_i\}/\max\{ratio_i\}$, we can achieve the following Lemma:

Algorithm 16 SuccRounding ( )

Require: ILP Formulation (5.6)

```
1: Set all b_ij as unsolved;
2: repeat
3:     Update p_i for all unsolved b_ij;
4:     Solve relaxed LP of (5.6);
5:     repeat
6:         b_pq ← max{b_ij};
7:         for all b_ij ≥ b_pq × β do
8:             if c_i can be assigned to row r_j then
9:                 b_ij = 1 and set it as solved;
10:                Update capacity of row j;
11:            end if
12:        end for
13:    until cannot find b_pq
14: until
```

Lemma 5.10. *The LP rounding solution of (5.7) can be a* 0.5α*-approximation to optimal solution of (5.7).*

Proof. First we introduce a modified formulation to program (5.7), where each p_i is set to $\min\{p_i\}$. In other words, in the modified formulation, each $ratio_i$ is the same. Let OPT and OPT' be the optimal values of (5.7) and the modified formulation, respectively. Let APR' be the corresponding LP rounding result in the modified formulation. According to Lemma 5.9, $APR' \geq 0.5 \cdot OPT'$. Since $\min\{p_i\} \geq p_i \cdot \alpha$, we can get $OPT' \geq \alpha \cdot OPT$. In summary, $APR' \geq 0.5 \cdot OPT' \geq 0.5\alpha \cdot OPT$.

The difference between (5.6) and (5.7) is the right side values at (5.6a) and (5.7a). Blank spacing is relatively small when compared with the row length, so we can get that $W - s_{\max} \approx W - B_j$. Then we can expect that program (5.6) has a reasonable rounding performance.

Successive Rounding

In this section we propose a successive rounding algorithm to solve program (5.6) iteratively. Successive rounding uses a simple iterative scheme in which fractional variables are rounded one after the other until an integral solution is found [57]. The ILP formulation (5.6) becomes an LP if we relax the discrete constraint to a continuous constraint as: $0 \leq b_{ij} \leq 1$.

The details of successive rounding is shown in Algorithm 16. At first we set all b_{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 b_{ij}, and store in b_{pq} (line 6). Then we find all b_{ij} that are closest to the maximum value b_{pq}, i.e., $b_{ij} \geq b_{pq} \times \beta$. In our implementation, β is set to 0.9. For each selected variables b_{ij}, we try to pack c_i into row j, and set b_{ij} as *solved*. Note that when one character c_i is assigned to one row, all b_{ij} would be set as solved. Therefore, the variable number in

updated LP formulation would continue to decrease. This procedure repeats until no appropriate b_{ij} can be found. One key step of Algorithm 16 is the p_i update (line 3). For each character c_i, we set its p_i as follows:

$$p_i = \sum_{k \in [K]} \frac{t_k}{t_{\max}} \cdot (a_i - 1) \cdot t_{ik} \tag{5.8}$$

where t_k is current writing time of region k, and $t_{\max} = \max_{k \in [K]}\{t_c\}$. Through applying the p_i, the region k with longer writing time would be considered more during the LP formulation. During successive rounding, if character c_i is not assigned to any row, p_i would continue to be updated, so that the total writing time of the whole MCC system can be minimized.

Fast ILP Convergence

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. 5.22 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 b_{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, and call fast ILP convergence

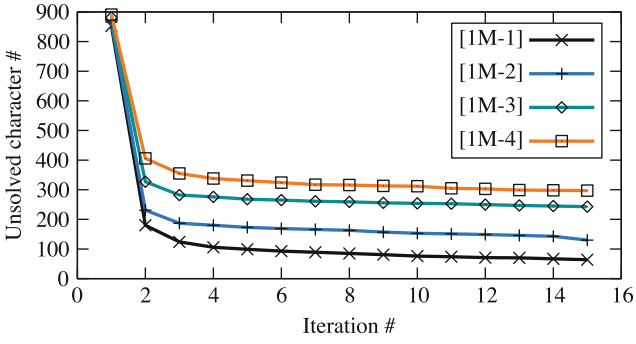

Fig. 5.22 Unsolved character number along the LP iterations for test cases 1M-1, 1M-2, 1M-3, and 1M-4

Algorithm 17 Fast ILP Convergence ( )

Require: Solutions of relaxed LP (5.6);
1: **for all** b_{ij} in relaxed LP solutions **do**
2: **if** $b_{ij} < \delta^-$ **then**
3: Set b_{ij} as solved;
4: **end if**
5: **if** $b_{ij} > \delta^+$ **then**
6: Assign c_i to row j;
7: Set b_{ij} as solved;
8: **end if**
9: **end for**
10: Solve ILP formulation (5.6) for all unsolved b_{ij};
11: **if** $b_{ij} = 1$ **then**
12: Assign c_i to row j;
13: **end if**

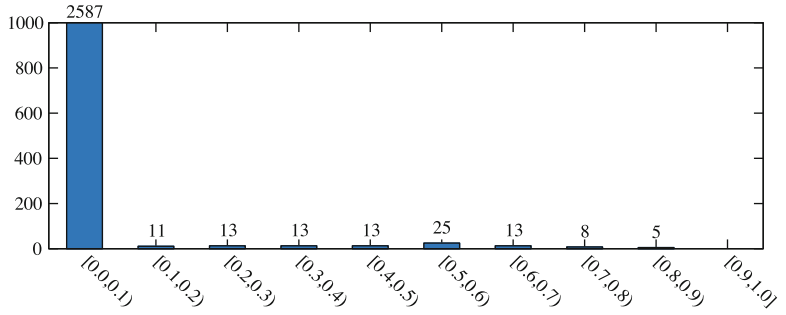

Fig. 5.23 For test case 1M-1, solution distribution in last LP, where most of values are close to 0

to assign all left characters. Note that in [48] an ILP formulation with similar idea was also applied. The details of the ILP convergence is shown in Algorithm 17. The inputs are the solutions of successive LP rounding. Besides, δ^- and δ^+ are two user-defined parameters. First we check each b_{ij} (lines 1–9). If $b_{ij} < \delta^-$, then we assume that character c_i would not be assigned to row j, and set b_{ij} as solved. Similarly, if $b_{ij} > \delta^+$, we assign c_i to row j and set b_{ij} as solved. For those unsolved b_{ij} we build up ILP formulation (5.6) 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 17, typically the variable number is small. Figure 5.23 illustrates the solution distribution in last LP formulation. We can see that most of the values are close to 0. In our implementation δ^- and δ^+ are set to 0.1 and 0.9, respectively. For this case, although the LP formulation contains more than 2500 variables, our fast ILP formulation results in only 101 binary variables.

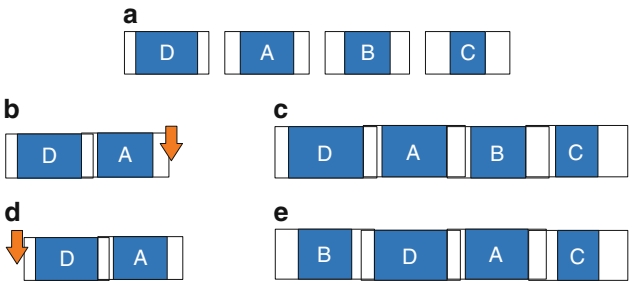

Fig. 5.24 Greedy based single row ordering. (**a**) At first all candidates are sorted by blank space. (**c**) One possible ordering solution where each candidate chooses the right end position. (**e**) Another possible ordering solution

Refinement

Refinement is a stage to solve the single row ordering problem [34], which adjusts the relative locations of input p characters to minimize the total width. Under the S-Blank assumption, because of Lemma 5.7, 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.

One example of the greedy approach is illustrated in Fig. 5.24, where four character candidates A, B, C, and D are to be ordered. In Fig. 5.24a, they are sorted decreasingly by blank 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. 5.24b), another is at the left side of A (Fig. 5.24d). Given different choices of candidate B, Fig. 5.24c, 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.

For the asymmetrical cases, the optimality 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 18. 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

Algorithm 18 Refine(k)

Require: k characters $\{c_1, \ldots, c_k\}$;
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** size of $S \geq \gamma$ **then**
11: Prune inferior solutions in S;
12: **end if**
13: **end if**

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 the 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 γ is set to 20.

Post-swap and Post-insertion

After refinement, a post-insertion stage is applied to further insert more characters into stencil. Different from the greedy insertion approach in [34] that new characters can only be inserted into one row's right end. We consider to insert characters into the middle part of rows. Generally speaking, the character with higher profit value (5.8) 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 [41].

Figure 5.25 illustrates one example of the character insertion. As shown in Fig. 5.25a, there are two rows (row 1 and 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

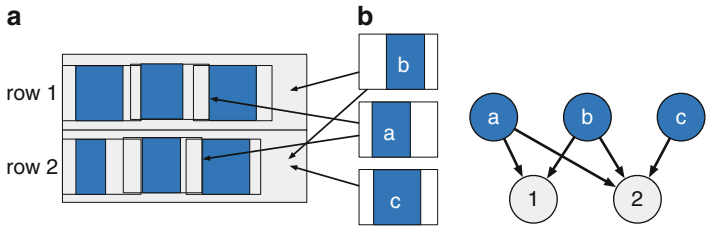

Fig. 5.25 Example of maximum weighted matching based post character insertion. (**a**) Three additional characters a, b, c and two rows. (**b**) Corresponding bipartite graph to represent the relationships among characters and rows

relationships among characters and rows (see Fig. 5.25b). Each edge is associated with a cost as character's profit. By utilizing 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.

5.3.4 *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 as 2D-OSP problem. In [34] 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.

Table 5.3 Notations used in 2D-ILP formulation

$W(H)$	Width (height) constraint of stencil
$w_i(h_i)$	Width (height) of candidate c_i
$o_{ij}^h(o_{ij}^v)$	Horizontal (vertical) overlap between c_i and c_j
$w_{ij}(h_{ij})$	$w_{ij} = w_i - o_{ij}^h$, $h_{ij} = h_i - o_{ij}^v$
a_i	0–1 variable, $a_i = 1$ if c_i is on stencil

ILP Formulation

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 also be formulated as an ILP formulation (5.9). For convenience, Table 5.3 lists some notations used in the ILP formulation. The formulation is motivated by Sutanthavibul et al. [58], but the difference is that our formulation can optimize both placement constraints and character selection, simultaneously.

$$\min T_{\text{total}} \tag{5.9}$$

$$\text{s.t. } T_{\text{total}} \geq T_c^{\text{VSB}} - \sum_{i=1}^{n} R_{ic} \cdot a_i, \quad \forall c \in P \tag{5.9a}$$

$$x_i + w_{ij} \leq x_j + W(2 + p_{ij} + q_{ij} - a_i - a_j) \tag{5.9b}$$

$$x_i - w_{ji} \geq x_j - W(3 + p_{ij} - q_{ij} - a_i - a_j) \tag{5.9c}$$

$$y_i + h_{ij} \leq y_j + H(3 - p_{ij} + q_{ij} - a_i - a_j) \tag{5.9d}$$

$$y_i - h_{ji} \geq y_j - H(4 - p_{ij} - q_{ij} - a_i - a_j) \tag{5.9e}$$

$$0 \leq x_i + w_i \leq W, \quad 0 \leq y_i + h_i \leq H \tag{5.9f}$$

$$p_{ij}, q_{ij}, a_i : \text{0–1 variable } \forall i, j \tag{5.9g}$$

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 (5.9b)–(5.9e) are not active. Besides, it is easy to 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 (5.9b)–(5.9e) are active. For example, with $(a_i, a_j, p_{ij}, q_{ij})$ = (1,1,1,1), only the constraint (5.9e) applies, which allows character c_i to be anywhere above character c_j. The other three constraints (5.9b)–(5.9d) are always satisfied for any permitted values of (x_i, y_i) and (x_j, y_j).

Program (5.9) can be relaxed to linear programming (LP) by replacing constraint (5.9g) 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

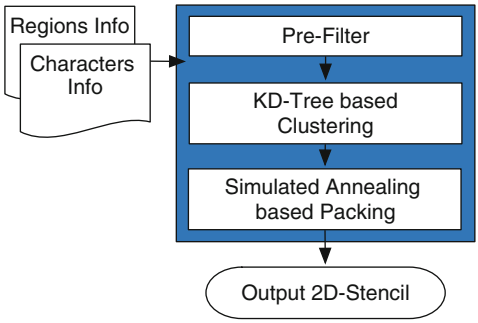

Fig. 5.26 E-BLOW overall flow for 2D-OSP

determined. Therefore we can see that LP rounding method cannot be effectively applied to program (5.9).

Clustering Based Simulated Annealing

To deal with all these limitations of ILP formulation, a fast packing framework is proposed (see Fig. 5.26). Given the input character candidates, the pre-filter process is first applied to remove characters with bad profit [defined in (5.8)]. 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 works and applications in VLSI [59–61]. However, previous methodologies cannot be directly applied here. First, traditional clustering is based on netlist, which provides all the 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 19. 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 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:

$$\begin{cases} \max\{|w_i - w_j|/w_j, |h_i - h_j|/h_j\} \le bound \\ \max\{|sh_i - sh_j|/sh_j, |sv_i - sv_j|/sv_j\} \le bound \\ |profit_i - profit_j|/profit_j \le bound \end{cases} \tag{5.10}$$

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 sizes, blanks, and profits are considered.

Algorithm 19 KD-Tree Based Clustering

Require: set of candidates C^C.

```
repeat
    Sort all candidates by profit_i;
    Set each candidates c_i to unclustered;
    for all unclustered candidate c_i do
        Search all similar character pairs (c_i, c_j);
        Cluster (c_i, c_j), label them as clustered;
    end for
    Update candidate information;
until No characters can be merged
```

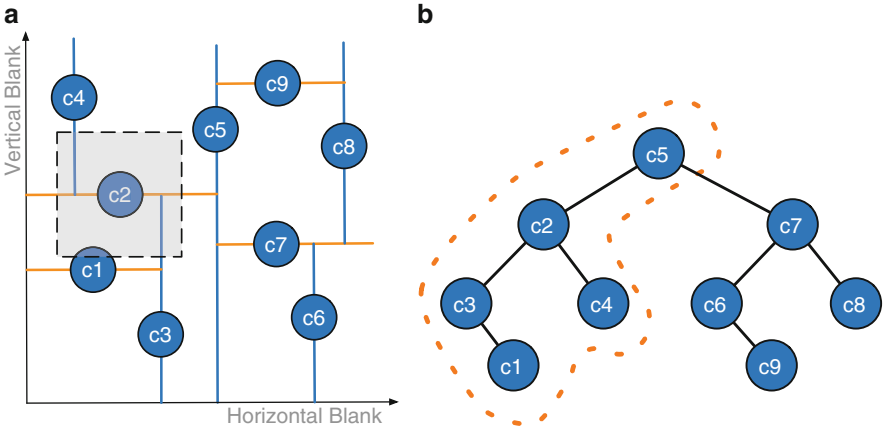

Fig. 5.27 KD-tree based region searching

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 [62] is used to speed-up the process of finding available pair (c_i, c_j). It provides fast $O(\log n)$ region searching operations which keep the time for insertion and deletion small: insertion, $O(\log n)$; deletion of the root, $O(n(k-1)/k)$; deletion of a random node, $O(\log n)$. Using KD-Tree, the complexity of the Algorithm 19 can be reduced to $O(n \log n)$. For instance, given nine character candidates $\{c_1, \ldots, c_9\}$, the corresponding KD-Tree is shown in Fig. 5.27a. For the sake of convenience, here characters are distributed only based on horizontal and vertical spaces. The edges of KD-Tree are labeled as well. To search for candidates with similar space with c_2 (see the shaded region of Fig. 5.27a), 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 $O(\log n)$. Particularly, all candidates scanned $(c_1 - c_5)$ are illustrated in Fig. 5.27b.

In [34], 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 [63] was adopted as simulated annealing engine, and Sequence Pair [64] was used as a topology representation. In E-BLOW we apply a simulated annealing based framework similar to that in [34]. To demonstrate the effectiveness of our pre-filter and clustering methodologies, E-BLOW uses the same parameters.

5.3.5 Experimental Results

E-BLOW is implemented in C++ programming language and executed on a Linux machine with two 3.0 GHz CPU and 32 GB Memory. GUROBI [65] is used to solve ILP/LP. The benchmark suite from [34] is 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 $2000 \times 2000\,\mu m$. The size and the blank width of each character are similar to those in [34].

For 1D-OSP, we compare E-BLOW with a greedy method in [34], a heuristic framework in [34], and algorithms in [48]. We have obtained the programs of Yuan et al. [34] and executed them in our machine. The results of Kuang and Young [48] are directly from their paper. Tables 5.4 and 5.5 list the performance comparison results and the runtime comparison results, respectively. 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. We can see that E-BLOW achieves better performance than both greedy method and heuristic method in [34]. Compared with E-BLOW, the greedy method has 32 % more system writing time, while [34] 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× faster than the work in [34].

E-BLOW is further compared with one recent 1D-OSP solver [48] in Table 5.4. 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 [48]. 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. In Table 5.5, although the linear programming

Table 5.4 Performance comparison for 1D-OSP

	Char #	CP #	Greedy in [34]		[34]		[48]		E-BLOW	
			T	Char#	T	Char#	T	Char#	T	Char#
1D-1	1000	1	64,891	912	50,809	926	**19,095**	940	19,479	940
1D-2	1000	1	99,381	884	93,465	854	35,295	864	**34,974**	866
1D-3	1000	1	165,480	748	152,376	749	69,301	757	**67,209**	766
1D-4	1000	1	193,881	691	193,494	687	**92,523**	703	93,766	703
1M-1	1000	10	63,811	912	53,333	926	39,026	938	**36,800**	945
1M-2	1000	10	104,877	884	95,963	854	77,997	864	**75,303**	874
1M-3	1000	10	172,834	748	156,700	749	138,256	758	**132,773**	774
1M-4	1000	10	200,498	691	196,686	687	176,228	698	**173,620**	711
1M-5	4000	10	274,992	3604	255,208	3629	204,114	3660	**201,492**	3681
1M-6	4000	10	437,088	3341	417,456	3346	357,829	3382	**348,007**	3420
1M-7	4000	10	650,419	3000	644,288	2986	568,339	3016	**559,655**	3070
1M-8	4000	10	820,013	2756	809,721	2734	731,483	2760	**721,149**	2818
Avg.	–	–	270,680.4	1597.6	259,958.3	1594.0	209,123.8	1611.7	205,352.3	1630.7
Ratio	–	–	**1.32**	0.98	**1.27**	0.98	**1.02**	0.99	**1.0**	1.0

Table 5.5 Runtime comparison for 1D-OSP

	Char #	CP #	Greedy in [34] CPU(s)	[34] CPU(s)	[48] CPU(s)	E-BLOW CPU(s)
1D-1	1000	1	0.1	13.5	0.005	1.9
1D-2	1000	1	0.1	11.8	0.005	1.6
1D-3	1000	1	0.1	9.13	0.005	1.6
1D-4	1000	1	0.1	7.7	0.005	4.6
1M-1	1000	10	0.1	13.5	0.01	3.2
1M-2	1000	10	0.1	11.8	0.01	3.0
1M-3	1000	10	0.1	9.2	0.56	7.8
1M-4	1000	10	0.1	7.7	0.36	8.8
1M-5	4000	10	1.0	1477.3	0.03	34.0
1M-6	4000	10	1.0	1182	0.03	45.0
1M-7	4000	10	1.0	876	0.59	43.4
1M-8	4000	10	1.0	730.7	0.42	49.5
Avg.	–	–	0.4	362.5	0.17	16.6
Ratio	–	–	0.03	21.9	0.01	1.0

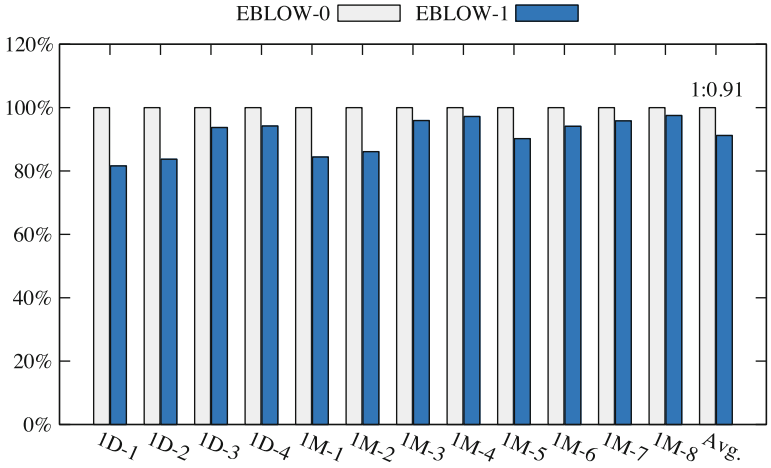

Fig. 5.28 The comparison of e-beam system writing times between E-BLOW-0 and E-BLOW-1

solvers are more expensive than the deterministic heuristics [48], the runtime of E-BLOW is reasonable that each case can be finished in 20 s 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. Figures 5.28 and 5.29 compare E-BLOW-0 and E-BLOW-1, in terms of system writing time and runtime, respectively. From Fig. 5.28 we can see that applying fast ILP convergence and post-insertion can effectively E-Beam system throughput, that is, averagely 9 %

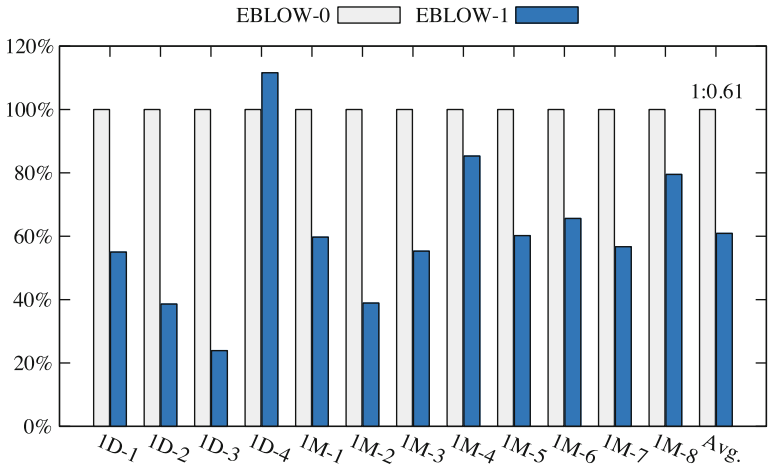

Fig. 5.29 The comparison of runtime between E-BLOW-0 and E-BLOW-1

system writing time reduction can be achieved. In addition, Fig. 5.29 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 5.6 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 5.4. 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 [34] puts 2 % characters onto stencil, it gets 15 % more system 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 [34]. 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 [34], E-BLOW can achieve a better trade-off between runtime and system throughput.

We further compare the E-BLOW with the ILP formulations (5.5) and (5.9). 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 to 40 × 40 μm. 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 5.7, where column "candidate#" is the number

Table 5.6 Result comparison for 2D-OSP

	Char #	CP #	Greedy in [34]			[34]			E-BLOW		
			T	Char #	CPU(s)	T	Char #	CPU(s)	T	Char #	CPU(s)
2D-1	1000	1	159,654	734	2.1	107,876	826	329.6	105,723	789	65.5
2D-2	1000	1	269,940	576	2.4	166,524	741	278.1	170,934	657	52.5
2D-3	1000	1	290,068	551	2.6	210,496	686	296.7	178,777	663	56.4
2D-4	1000	1	327,890	499	2.7	240,971	632	301.7	179,981	605	54.7
2M-1	1000	1	168,279	734	2.1	122,017	811	313.7	91,193	777	58.6
2M-2	1000	1	283,702	576	2.4	187,235	728	286.1	163,327	661	48.7
2M-3	1000	1	298,813	551	2.6	235,788	653	289.0	162,648	659	52.3
2M-4	1000	1	338,610	499	2.7	270,384	605	285.6	195,469	590	53.3
2M-5	4000	10	824,060	2704	19.0	700,414	2913	3891.0	687,287	2853	59.0
2M-6	4000	10	1,044,161	2388	20.2	898,530	2624	4245.0	717,236	2721	60.7
2M-7	4000	10	1,264,748	2101	21.9	1,064,789	2410	3925.5	921,867	2409	57.1
2M-8	4000	10	1,331,457	2011	22.8	1,176,700	2259	4550.0	1,104,724	2119	57.7
Avg.	–	–	550,115	1218.1	8.3	448,477	1324	1582.7	389,930.5	1291.9	56.375
Ratio	–	–	**1.41**	0.94	**0.15**	**1.15**	1.02	**28.1**	**1.0**	1.0	**1.0**

Table 5.7 ILP v.s. E-BLOW

		ILP				E-BLOW		
	Candidate#	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

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 5.7 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 s. Although ILP formulation can achieve optimal results, it is very slow that a case with 14 character candidates (1T-5) cannot be solved in 1 h. Next, let us 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 1 h. 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.

5.4 Summary

In this chapter we have introduced different types of EBL system: VSB mode and CP mode, and proposed a set of optimization techniques to overcome the throughput limitation.

For VSB mode EBL system, we developed L-shape based layout fracturing for VSB shot number and sliver minimization. The rectangular merging (RM)-based algorithm is optimal for a given set of rectangular fractures. However, we show

that the direct L-shape fracturing (DLF) algorithm has superior performance by directly decomposing the original layouts into a set of L-shapes. DLF obtained the best results in all metrics: shot count, sliver length, and runtime, compared to the previous state-of-the-art rectangular fracturing with RM.

For CP mode, we explore an extended MCC system, and discussed the corresponding OSP problem in the 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.

References

1. Kahng, A.B., Park, C.-H., Xu, X., Yao, H.: Layout decomposition for double patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 465–472 (2008)
2. Zhang, H., Du, Y., Wong, M.D., Topaloglu, R.: Self-aligned double patterning decomposition for overlay minimization and hot spot detection. In: ACM/IEEE Design Automation Conference (DAC), pp. 71–76 (2011)
3. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2011)
4. Yu, B., Pan, D.Z.: Layout decomposition for quadruple patterning lithography and beyond. In: ACM/IEEE Design Automation Conference (DAC), pp. 53:1–53:6 (2014)
5. Pan, D.Z., Yu, B., Gao, J.-R.: Design for manufacturing with emerging nanolithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(10), 1453–1472 (2013)
6. Arisawa, Y., Aoyama, H., Uno, T., Tanaka, T.: EUV flare correction for the half-pitch 22nm node. In: Proceedings of SPIE, vol. 7636 (2010)
7. Zhang, H., Du, Y., Wong, M.D.F., Deng, Y., Mangat, P.: Layout small-angle rotation and shift for EUV defect mitigation. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 43–49 (2012)
8. Chang, L.-W., Bao, X., Chris, B., Philip Wong, H.-S.: Experimental demonstration of aperiodic patterns of directed self-assembly by block copolymer lithography for random logic circuit layout. In: IEEE International Electron Devices Meeting (IEDM), pp. 33.2.1–33.2.4 (2010)
9. Pfeiffer, H.C.: New prospects for electron beams as tools for semiconductor lithography. In: Proceedings of SPIE, vol. 7378 (2009)
10. Fujimura, A.: Design for e-beam: design insights for direct-write maskless lithography. In: Proceedings of SPIE, vol. 7823 (2010)
11. Maruyama, T., Takakuwa, M., Kojima, Y., Takahashi, Y., Yamada, K., Kon, J., Miyajima, M., Shimizu, A., Machida, Y., Hoshino, H., Takita, H., Sugatani, S., Tsuchikawa, H.: EBDW technology for EB shuttle at 65nm node and beyond. In: Proceedings of SPIE, vol. 6921 (2008)
12. Manakli, S., Komami, H., Takizawa, M., Mitsuhashi, T., Pain, L.: Cell projection use in maskless lithography for 45nm & 32nm logic nodes. In: Proceedings of SPIE, vol. 7271 (2009)

13. Du, Y., Zhang, H., Wong, M.D.F., Chao, K.-Y.: Hybrid lithography optimization with e-beam and immersion processes for 16nm 1D gridded design. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 707–712 (2012)
14. Gao, J.-R., Yu, B., Pan, D.Z.: Self-aligned double patterning layout decomposition with complementary e-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 143–148 (2014)
15. Ding, Y., Chu, C., Mak, W.-K.: Throughput optimization for SADP and e-beam based manufacturing of 1D layout. In: ACM/IEEE Design Automation Conference (DAC), pp. 51:1–51:6 (2014)
16. Yang, Y., Luk, W.-S., Zhou, H., Yan, C., Zeng, X., Zhou, D.: Layout decomposition co-optimization for hybrid e-beam and multiple patterning lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 652–657 (2015)
17. Fang, S.-Y., Liu, I.-J., Chang, Y.-W.: Stitch-aware routing for multiple e-beam lithography. In: ACM/IEEE Design Automation Conference (DAC), pp. 25:1–25:6 (2013)
18. Babin, S., Kahng, A.B., Mandoiu, I.I., Muddu, S.: Resist heating dependence on subfield scheduling in 50kV electron beam maskmaking. In: Proceedings of SPIE, vol. 5130 (2003)
19. Fang, S.-Y., Chen, W.-Y., Chang, Y.-W.: Graph-based subfield scheduling for electron-beam photomask fabrication. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(2), 189–201 (2013)
20. Kahng, A.B., Xu, X., Zelikovsky, A.: Fast yield-driven fracture for variable shaped-beam mask writing. In: Proceedings of SPIE, vol. 6283 (2006)
21. Ma, X., Jiang, S., Zakhor, A.: A cost-driven fracture heuristics to minimize sliver length. In: Proceedings of SPIE, vol. 7973 (2011)
22. Yu, B., Gao, J.-R., Pan, D.Z.: L-shape based layout fracturing for e-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 249–254 (2013)
23. Chan, T.B., Gupta, P., Han, K., Kagalwalla, A.A., Kahng, A.B., Sahouria, E.: Benchmarking of mask fracturing heuristics. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 246–253 (2014)
24. Fujino, T., Kajiya, Y., Yoshikawa, M.: Character-build standard-cell layout technique for high-throughput character-projection EB lithography. In: Proceedings of SPIE, vol. 5853 (2005)
25. Sugihara, M., Takata, T., Nakamura, K., Inanami, R., Hayashi, H., Kishimoto, K., Hasebe, T., Kawano, Y., Matsunaga, Y., Murakami, K., Okumurae, K.: Technology mapping technique for throughput enhancement of character projection equipment. In: Proceedings of SPIE, vol. 6151 (2007)
26. Sugihara, M., Takata, T., Nakamura, K., Inanami, R., Inanami, R., Hayashi, H., Kishimoto, K., Hasebe, T., Kawano, Y., Matsunaga, Y., Murakami, K., Okumura, K.: A character size optimization technique for throughput enhancement of character projection lithography. In: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2561–2564 (2006)
27. Du, P., Zhao, W., Weng, S.-H., Cheng, C.-K., Graham, R.: Character design and stamp algorithms for character projection electron-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 725–730 (2012)
28. Ikeno, R., Maruyama, T., Iizuka, T., Komatsu, S., Ikeda, M., Asada, K.: High-throughput electron beam direct writing of VIA layers by character projection using character sets based on one-dimensional VIA arrays with area-efficient stencil design. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 255–260 (2013)
29. Minh, H.P.D., Iizuka, T., Ikeda, M., Asada, K.: Shot minimization for throughput improvement of character projection electron beam direct writing. In: Proceedings of SPIE, vol. 6921 (2006)
30. Ikeno, R., Maruyama, T., Komatsu, S., Iizuka, T., Ikeda, M., Asada, K.: A structured routing architecture and its design methodology suitable for high-throughput electron beam direct writing with character projection. In: ACM International Symposium on Physical Design (ISPD), pp. 69–76 (2013)
31. Lee, S.H., Choi, J., Kim, H.B., Kim, B.G., Cho, H.-K.: The requirements for the future e-beam mask writer: statistical analysis of pattern accuracy. In: Proceedings of SPIE, vol. 8166 (2011)

32. Sahouria, E., Bowhill, A.: Generalization of shot definition for variable shaped e-beam machines for write time reduction. In: Proceedings of SPIE, vol. 7823 (2010)
33. Elayat, A., Lin, T., Sahouria, E., Schulze, S.F.: Assessment and comparison of different approaches for mask write time reduction. In: Proceedings of SPIE, vol. 8166 (2011)
34. Yuan, K., Yu, B., Pan, D.Z.: E-beam lithography stencil planning and optimization with overlapped characters. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **31**(2), 167–179 (2012)
35. Kahng, A.B., Xu, X., Zelikovsky, A.: Yield-and cost-driven fracturing for variable shaped-beam mask writing. In: Proceedings of SPIE, vol. 5567 (2004)
36. Dillon, B., Norris, T.: Case study: the impact of VSB fracturing. In: Proceedings of SPIE, vol. 7028 (2008)
37. Jiang, S., Ma, X., Zakhor, A.: A recursive cost-based approach to fracturing. In: Proceedings of SPIE, vol. 7973 (2011)
38. Edelsbrunner, H., O'Rourke, J., Welzl, E.: Stationing guards in rectilinear art galleries. Comput. Vis. Graph. Image Process. **28**, 167–176 (1984)
39. Lopez, M.A., Mehta, D.P.: Efficient decomposition of polygons into L-shapes with application to VLSI layouts. ACM Trans. Des. Autom. Electron. Syst. **1**(3), 371–395 (1996)
40. O'Rourke, J.: An alternate proof of the rectilinear art gallery theorem. J. Geom. **21**, 118–130 (1983)
41. Galil, Z.: Efficient algorithms for finding maximum matching in graphs. ACM Comput. Surv. **18**(1), 23–38 (1986)
42. Mehlhorn, K., Naher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
43. Guiney, M., Leavitt, E.: An introduction to OpenAccess: an open source data model and API for IC design. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 434–436 (2006)
44. Yasuda, H., Haraguchi, T., Yamada, A.: A proposal for an MCC (multi-column cell with lotus root lens) system to be used as a mask-making e-beam tool. In: Proceedings of SPIE, vol. 5567 (2004)
45. Maruyama, T., Machida, Y., Sugatani, S., Takita, H., Hoshino, H., Hino, T., Ito, M., Yamada, A., Iizuka, T., Komatsue, S., Ikeda, M., Asada, K.: CP element based design for 14nm node EBDW high volume manufacturing. In: Proceedings of SPIE, vol. 8323 (2012)
46. Shoji, M., Inoue, T., Yamabe, M.: Extraction and utilization of the repeating patterns for CP writing in mask making. In: Proceedings of SPIE, vol. 7748 (2010)
47. Sugihara, M., Takata, T., Nakamura, K., Inanami, R., Hayashi, H., Kishimoto, K., Hasebe, T., Kawano, Y., Matsunaga, Y., Murakami, K., Okumura, K.: Cell library development methodology for throughput enhancement of character projection equipment. IEICE Trans. Electron. **E89-C**, 377–383 (2006)
48. Kuang, J., Young, E.F.: A highly-efficient row-structure stencil planning approach for e-beam lithography with overlapped characters. In: ACM International Symposium on Physical Design (ISPD), pp. 109–116 (2014)
49. Guo, D., Du, Y., Wong, M.D.: Polynomial time optimal algorithm for stencil row planning in e-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 658–664 (2015)
50. Chu, C., Mak, W.-K.: Flexible packed stencil design with multiple shaping apertures for e-beam lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 137–142 (2014)
51. Mak, W.-K., Chu, C.: E-beam lithography character and stencil co-optimization. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(5), 741–751 (2014)
52. Yu, B., Yuan, K., Gao, J.-R., Pan, D.Z.: E-BLOW: e-beam lithography overlapping aware stencil planning for MCC system. In: ACM/IEEE Design Automation Conference (DAC), pp. 70:1–70:7 (2013)
53. Kuang, J., Young, E.F.: Overlapping-aware throughput-driven stencil planning for e-beam lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 254–261 (2014)

54. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)
55. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York (1990)
56. Dawande, M., Kalagnanam, J., Keskinocak, P., Salman, F., Ravi, R.: Approximation algorithms for the multiple knapsack problem with assignment restrictions. J. Comb. Optim. **4**, 171–186 (2000)
57. Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.: Progress in linear programming-based algorithms for integer programming: an exposition. INFORMS J. Comput. **12**(1), 2–23 (2000)
58. Sutanthavibul, S., Shragowitz, E., Rosen, J.: An analytical approach to floorplan design and optimization. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **10**(6), 761–769 (1991)
59. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: application in VLSI domain. In: ACM/IEEE Design Automation Conference (DAC), pp. 526–529 (1997)
60. Nam, G.-J., Reda, S., Alpert, C., Villarrubia, P., Kahng, A.: A fast hierarchical quadratic placement algorithm. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **25**(4), 678–691 (2006)
61. Yan, J.Z., Chu, C., Mak, W.-K.: SafeChoice: a novel clustering algorithm for wirelength-driven placement. In: ACM International Symposium on Physical Design (ISPD), pp. 185–192 (2010)
62. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**, 509–517 (1975)
63. Adya, S.N., Markov, I.L.: Fixed-outline floorplanning: enabling hierarchical design. IEEE Trans. Very Large Scale Integr. Syst. **11**(6), 1120–1135 (2003)
64. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: VLSI module placement based on rectangle-packing by the sequence-pair. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **12**, 1518–1524 (1996)
65. Gurobi Optimization Inc.: Gurobi optimizer reference manual. http://www.gurobi.com (2014)

Chapter 6
Conclusions and Future Works

In this book, we have proposed a set of algorithms/methodologies to resolve issues in modern design for manufacturability (DFM) problems with advanced lithography. The methodologies proposed are flexible that they can be easily extended to handle other emerging lithography challenges in layout design stages. Our major contributions include:

- In Chaps. 2 and 3, we tackled the challenge of layout decompositions for different patterning techniques. In Sect. 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 Sect. 2.3 we proposed a high performance layout decomposer providing more balanced density. In Sect. 3.2 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 ILP. In Sect. 3.3 we extend to general multiple patterning problems. The proposed decomposer consists of holistic algorithmic processes, such as SDP 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 Chap. 4, 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 pre-coloring 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

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0_6

with maximum displacement, which can achieve good trade-off in terms of runtime and performance.

- In Chap. 5, we study the DFM with E-Beam lithography. In Sect. 5.3 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 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 Sect. 5.2 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 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 [1, 2]. Recently, [3] proposed an SDP based formulation. However, both methods may suffer from long runtime penalty, especially to whole chip level decomposition cases. Therefore, how to achieve better runtime and solution quality balance is an open question.
- We have integrated triple patterning constraints into early design stages [4, 5]. Recently, this research direction has earned more and more attention [6–10]. 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), extreme ultra violet (EUV), and nanoimprint lithography (NIL) are other promising candidates for next generation lithography techniques. For DSA, it has demonstrated its potential to generate dense contacts for cut patterns [11]. A lot of works have been done on the investigation of DSA contact layer fabrication and DSA aware physical design [12–15]. For EUV, there are lots of DFM research works to overcome the defect issues (e.g., [16–20]). However, we still need to further study and evaluate NIL as an option for future manufacturing.
- Although various resolution enhancement techniques are utilized, random geometrical configurations are still hard to implement due to lithography limitation [21]. Therefore, unidirectional layout styles have been proposed to improve the manufacturability and achieve manageable post-layout processing complexity

[22–24]. Recently, some works discussed how to handle unidirectional layout in standard cell design stages [25, 26]. However, although there are some attempts (e.g., [27–29]), how to handle the unidirectional design style beyond standard cell level is still an open problem.

References

1. Yu, B., Gao, J.-R., Pan, D.Z.: Triple patterning lithography (TPL) layout decomposition using end-cutting. In: Proceedings of SPIE, vol. 8684 (2013)
2. Yu, B., Roy, S., Gao, J.-R., Pan, D.Z.: Triple patterning lithography layout decomposition using end-cutting. J. Micro/Nanolithogr. MEMS MOEMS (JM3) **14**(1), 011002 (2015)
3. Kohira, Y., Matsui, T., Yokoyama, Y., Kodama, C., Takahashi, A., Nojima, S., Tanaka, S.: Fast mask assignment using positive semidefinite relaxation in LELECUT triple patterning lithography. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 665–670 (2015)
4. Yu, B., Xu, X., Gao, J.-R., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 349–356 (2013)
5. Yu, B., Xu, X., Gao, J.-R., Lin, Y., Li, Z., Alpert, C., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (TCAD) **34**(5), 726–739 (2015)
6. Kuang, J., Chow, W.-K., Young, E.F.Y.: Triple patterning lithography aware optimization for standard cell based design. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 108–115 (2014)
7. Chien, H.-A., Chen, Y.-H., Han, S.-Y., Lai, H.-Y., Wang, T.-C.: On refining row-based detailed placement for triple patterning lithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (TCAD) **34**(5), 778–793 (2015)
8. Lin, T., Chu, C.: TPL-aware displacement-driven detailed placement refinement with coloring constraints. In: ACM International Symposium on Physical Design (ISPD), pp. 75–80 (2015)
9. Tian, H., Du, Y., Zhang, H., Xiao, Z., Wong, M.D.F.: Triple patterning aware detailed placement with constrained pattern assignment. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 116–123 (2014)
10. Gao, J.-R., Yu, B., Huang, R., Pan, D.Z.: Self-aligned double patterning friendly configuration for standard cell library considering placement. In: Proceedings of SPIE, vol. 8684 (2013)
11. Yi, H., Bao, X.-Y., Zhang, J., Tiberio, R., Conway, J., Chang, L.-W., Mitra, S., Wong, H.-S.P.: Contact-hole patterning for random logic circuit using block copolymer directed self-assembly. In: Proceedings of SPIE, vol. 8323 (2012)
12. Xiao, Z., Du, Y., Wong, M.D., Zhang, H.: DSA template mask determination and cut redistribution for advanced 1D gridded design. In: Proceedings of SPIE, vol. 8880 (2013)
13. Du, Y., Guo, D., Wong, M.D.F., Yi, H., Wong, H.-S.P., Zhang, H., Ma, Q.: Block copolymer directed self-assembly (DSA) aware contact layer optimization for 10 nm 1D standard cell library. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 186–193 (2013)
14. Du, Y., Xiao, Z., Wong, M.D., Yi, H., Wong, H.-S.P.: DSA-aware detailed routing for via layer optimization. In: Proceedings of SPIE, vol. 9049 (2014)
15. Ou, J., Yu, B., Gao, J.-R., Pan, D.Z., Preil, M., Latypov, A.: Directed self-assembly based cut mask optimization for unidirectional design. In: ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 83–86 (2015)

16. Zhang, H., Du, Y., Wong, M.D.F., Tapalaglu, R.O.: Efficient pattern relocation for EUV blank defect mitigation. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 719–724 (2012)
17. Zhang, H., Du, Y., Wong, M.D.F., Deng, Y., Mangat, P.: Layout small-angle rotation and shift for EUV defect mitigation. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 43–49 (2012)
18. Du, Y., Zhang, H., Ma, Q., Wong, M.D.F.: Linear time algorithm to find all relocation positions for EUV defect mitigation. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 261–266 (2013)
19. Kagalwalla, A.A., Gupta, P., Hur, D.-H., Park, C.-H.: Defect-aware reticle floorplanning for EUV masks. In: Proceedings of SPIE, vol. 7479 (2011)
20. Fang, S.-Y., Chang, Y.-W.: Simultaneous flare level and flare variation minimization with dummification in EUVL. In: ACM/IEEE Design Automation Conference (DAC), pp. 1179–1184 (2012)
21. Pan, D.Z., Yu, B., Gao, J.-R.: Design for manufacturing with emerging nanolithography. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (TCAD) **32**(10), 1453–1472 (2013)
22. Jhaveri, T., Rovner, V., Liebmann, L., Pileggi, L., Strojwas, A.J., Hibbeler, J.D.: Co-optimization of circuits, layout and lithography for predictive technology scaling beyond gratings. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (TCAD) **29**(4), 509–527 (2010)
23. Du, Y., Zhang, H., Wong, M.D.F., Chao, K.-Y.: Hybrid lithography optimization with e-beam and immersion processes for 16nm 1D gridded design. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 707–712 (2012)
24. Liebmann, L., Chu, A., Gutwin, P.: The daunting complexity of scaling to 7 nm without EUV: pushing DTCO to the extreme. In: Proceedings of SPIE, vol. 9427 (2015)
25. Xu, X., Cline, B., Yeric, G., Yu, B., Pan, D.Z.: Self-aligned double patterning aware pin access and standard cell layout co-optimization. In: ACM International Symposium on Physical Design (ISPD), pp. 101–108 (2014)
26. Ye, W., Yu, B., Ban, Y.-C., Liebmann, L., Pan, D.Z.: Standard cell layout regularity and pin access optimization considering middle-of-line. In: ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 289–294 (2015)
27. Fang, S.-Y.: Cut mask optimization with wire planning in self-aligned multiple patterning full-chip routing. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 396–401 (2015)
28. Xu, X., Yu, B., Gao, J.-R., Hsu, C.-L., Pan, D.Z.: PARR: pin access planning and regular routing for self-aligned double patterning. In: ACM/IEEE Design Automation Conference (DAC), pp. 28:1–28:6 (2015)
29. Su, Y.-H., Chang, Y.-W.: Nanowire-aware routing considering high cut mask complexity. In: ACM/IEEE Design Automation Conference (DAC), pp. 138:1–138:6 (2015)

Index

B. Yu, D.Z. Pan, *Design for Manufacturability with Advanced Lithography*,
DOI 10.1007/978-3-319-20385-0