(c) 2019 DAIFENG GUO

# ALGORITHMS FOR DFM IN ELECTRONIC DESIGN AUTOMATION 

BY<br>DAIFENG GUO

## DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:
Professor Martin D. F. Wong, Chair
Professor Deming Chen
Associate Professor Shobha Vasudevan
Dr. Charles Chiang, Synopsys

## ABSTRACT

As the dimension of features in integrated circuits (IC) keeps shrinking to fulfill Moore's law, the manufacturing process has no choice but confronting the limit of physics at the expense of design flexibility. On the other hand, IC designs inevitably becomes more complex to meet the increasing demand of computational power. To close this gap, design for manufacturing (DFM) becomes the key to enable an easy and low-cost IC fabrication. Therefore, efficient electronic design automation (EDA) algorithms must be developed for DFM to address the design constraints and help the designers to better facilitate the manufacture process. As the core of manufacturing ICs, conventional lithography systems (193i) reach their limit for the 22 nm technology node and beyond. Consequently, several advanced lithography techniques are proposed, such as multiple patterning lithography (MPL), extreme ultra-violet lithography (EUV), electron beam (E-beam), and block copolymer directed selfassembly (DSA); however, DFM algorithms are essential for them to achieve better printability of a design. In this dissertation, we focus on analyzing the compatibility of designs and various advanced lithography techniques, and develop efficient algorithms to enable the manufacturing.

We first explore E-Beam, one of the promising candidates for IC fabrication beyond the 10 nm technology node. To address its low throughput issue, the character projection technique has been proposed, and its stencil planning can be optimized with an awareness of overlapping characters. 2D stencil planning is proved NP-Hard. With the assumption of standard cells, the 2D problem can be partitioned into 1D row ordering subproblems; however, it is also considered hard, and no efficient optimal solution has been provided so far. We propose a polynomial time optimal algorithm to solve the 1D row ordering problem, which serves as the major subroutine for the entire stencil planning problem. Technical proofs and experimental results verify that our algorithm is efficient and indeed optimal.

As the most popular and practical lithography technique, MPL utilizes multiple exposures to print a single layout and thus allows placement of features within the minimum distance. Therefore, a feasible decomposition of the layout is a must to adopt MPL, and it is usually formulated as a graph k-coloring problem, which is computationally difficult for $k>2$. We study the k-colorability of rectangular and diagonal grid graphs as induced subgraphs of a rectangular or diagonal grid respectively, since it has direct applications in printing contact/via layouts. It remains an open question on how hard it is to color grid graphs due to their regularity and sparsity. In this dissertation, we conduct a complete analysis of the k-coloring problems on rectangular and diagonal grid graphs, and particularly the NP-completeness of 3 -coloring on a diagonal grid graph is proved. In practice, we propose an exact 3-coloring algorithm for those graphs and conduct experiments to verify its performance and effectiveness. Besides, we also develop an efficient algorithm for model based MPL, because it is more expensive but accurate than the rule based decomposition.

As one of the alternative lithography techniques, block copolymer directed self-assembly (DSA) is studied. It has emerged as a low-cost, highthroughput option in the pursuit of alternatives to traditional optical lithography. However, issues of defectivity have hampered DSA's viability for large-scale patterning. Recent studies have shown the copolymer fill level to be a crucial factor in defectivity, as template overfill can result in malformed DSA structures and poor LCDU after etching. For this reason, the use of sub-DSA resolution assist features (SDRAFs) as a method of evening out template density has been demonstrated. In this dissertation, we propose an algorithm to place SDRAFs in random logic contact/via layouts. By adopting this SDRAF placement scheme, we can significantly improve the density unevenness and the resources used are also optimized. We also apply our knowledge in coloring grid graphs to the problem of group-and-coloring in DSA-MPL hybrid lithography. We derive a solution to group-3-coloring and prove the NP-completeness of grouping-2-coloring.

To my parents and grandparents, for their love and support.
To world peace.
To Champaign.

## ACKNOWLEDGMENTS

I would like to show my deepest gratitude to my adviser, Professor Martin D.F. Wong. You guided me into the area of electronic design automation and gave me insightful advices on research throughout my PhD career. Without your support and wisdom, this dissertation would not be possible. I am also very grateful to the rest of members in my doctoral committee, Professor Deming Chen, Professor Shobha Vasudevan and Doctor Chiang. Their valuable comments and suggestions raise this dissertation to another level. I want to express my thanks to everyone that helped me during my time at UIUC. Special thanks go to my colleagues Hongbo Zhang, Yuelin Du, Haitong Tian, Zigang Xiao, Chun-Xun Lin, Tsung-Wei Huang, Leslie Hwang, Tan Yan, Qiang Ma, Tin-Yin Lai, Guannan Guo, Iou-Jen Liu, Ting Yu, Deojkiin Joo, Fan Zhang, Maryann Tung, H.-S. Philip Wong and Yi He. Finally, words cannot fully express my love to Champaign, the town that I spent almost one-third of my life. You witnessed me growing up from a boy to a man.

## TABLE OF CONTENTS

CHAPTER 1 A POLYNOMIAL TIME OPTIMAL ALGORITHM FOR STENCIL ROW PLANNING IN E-BEAM LITHOGRAPHY ..... 1
1.1 Introduction ..... 1
1.2 Problem Formulation ..... 3
1.3 Algorithm ..... 4
1.4 Proof ..... 17
1.5 Experimental Results ..... 22
1.6 Conclusion ..... 23
CHAPTER 2 MODEL-BASED MULTIPLE PATTERNING LAY- OUT DECOMPOSITION ..... 24
2.1 Introduction ..... 24
2.2 Motivation ..... 27
2.3 Preprocessing ..... 28
2.4 Algorithm ..... 32
2.5 Experimental Result ..... 34
2.6 Conclusion ..... 35
CHAPTER 3 COLORING RECTANGULAR AND DIAGONAL GRID GRAPHS FOR MULTI-PATTERNING LITHOGRAPHY ..... 36
3.1 Introduction ..... 36
3.2 Problem Definition ..... 39
3.3 Coloring a Rectangular Grid Graph ..... 39
3.4 Coloring a Diagonal Grid Graph ..... 41
3.5 3-Coloring a Diagonal Grid Graph ..... 44
3.6 An Exact 3-Coloring Algorithm and Its Experiments ..... 53
3.7 Conclusion ..... 57
CHAPTER 4 GROUPING AND COLORING DIAGONAL GRID GRAPHS FOR DIRECTED SELF-ASSEMBLY LITHOGRAPHY ..... 58
4.1 Introduction ..... 58
4.2 Problem Definition ..... 60
4.3 Grouping-3-Coloring a Diagonal Graph ..... 61
4.4 Grouping-2-Coloring a Diagonal Graph ..... 62
4.5 Conclusion ..... 68
CHAPTER 5 DENSITY DRIVEN PLACEMENT OF SUB-DSA RESOLUTION ASSISTANT FEATURES (SDRAFS) FOR DI- RECTED SELF-ASSEMBLY LITHOGRAPHY ..... 69
5.1 Introduction ..... 69
5.2 Preliminary ..... 70
5.3 Algorithm ..... 73
5.4 Experimental Result ..... 76
5.5 Conclusion ..... 77
CHAPTER 6 DENSITY BALANCING AWARE MASK ASSIGN-
TACT LAYERS ..... 79
6.1 Introduction ..... 79
6.2 Background and Problem Formulation ..... 80
6.3 ILP Approach ..... 85
6.4 Experimental Results ..... 88
6.5 Concluding Remarks ..... 88
REFERENCES ..... 90

## CHAPTER 1

## A POLYNOMIAL TIME OPTIMAL ALGORITHM FOR STENCIL ROW PLANNING IN E-BEAM LITHOGRAPHY

### 1.1 Introduction

Integrated circuit (IC) fabrication continues according to Moore's law in achieving denser devices. Below the 28 nm technology node, conventional 193 nm immersion (193i) lithography with single exposure has reached its printability limit, which triggers some advanced lithography techniques such as double patterning lithography (DPL) [1] and triple patterning lithography (TPL) [2]. However, multiple patterning lithography (MPL) introduces new challenges such as decomposability, stitches and overlay, and the manufacturing cost increases exponentially with the number of masks. As a result, other promising candidates are also being explored for the next-generation lithography, including extreme ultraviolet lithography (EUVL) [3], directed self-assembly (DSA) [4] and electron beam lithography (EBL). Each of the advanced lithography techniques has its own advantages over others, but also faces great challenges due to different process limitations. EBL, for instance, is able to print extremely complicated and dense features, but faces one major challenge of low throughput.

The most intuitive version of EBL is electron beam direct write (EBDW), which shoots the desired patterns pixel by pixel, and thus has very low throughput. One essential improvement of EBL is the variable shaped beam (VSB) [5, 6], which can print an arbitrarily sized rectangle with one single shot. However, since current layout designs contain billions of rectangles, the throughput of VSB is still incapable of meeting the requirement. To further improve the throughput of EBL, Character Projection (CP) (later multicolumn cell (MCC)) has been proposed [7, 8], which is capable of printing an entire character (e.g. a standard cell) with one shot.

There are two major challenges in CP. First, how to design the set of pro-
jection characters; second, how to plan the stencil to pack as many characters as possible. The former problem is investigated by [8, 9]. For the latter problem, placement optimization should be performed based on the fact that the characters can overlap at the blank margins located at the character boundaries, as illustrated in Fig. 1.1. The blank margin is used to reserve some space for the scattered electrons after they pass through the aperture [7]. By sharing the blank area in Figs. 1.1(b) and (c), the characters occupy less stencil area than those in Fig. 1.1(a). Obviously, different placements of the characters result in different area occupation as illustrated by Figs. 1.1(b) and (c), because the shared blank margins in total are different among different placement solutions. For a given set of characters, it is a challenging problem to find their optimal placement, such that they occupy the smallest stencil area and leave more room to insert additional characters or features.


Figure 1.1: Comparison of stencil area occupation without and with blank margin sharing.

In the stencil planning problem, it is reasonable to assume that the characters are selected from standard cells or vertical slices of cells, which have the same heights. In addition, those standard cell characters also share very similar top and bottom blank margins, because a standard cell usually has power tracks on the top and bottom, and the distance that scattered electrons can travel outside the character is highly dependent on the pattern near its boundaries. With such assumptions, we do not need to consider the vertical placement constraints, and in consequence, the original character placement problem can be reduced as a row ordering problem, which has been proposed as the 1D overlapping aware stencil planning (OSP) problem in previous works. Several attempts have been made to solve this problem. However, Yuan et al. [10] formulated it as an NP-hard problem, and previous works [11, 10, 12] provided heuristic approaches and made a number of assumptions to guarantee their solution's quality and performance, for
instance, the difference between left and right blank margins is very small. Besides, Chu [13] use different assumptions of the process, i.e., the projection region belongs to a set of shapes and the character can locate anywhere in the projection region. All those assumptions are related to the EBL process and not only need to be proved by realistic litho-experimental results, but also make the problem much simpler. In this chapter, we neatly solve the general row ordering problem without any additional assumption by a polynomial time optimal algorithm and we prove its optimality rigorously. Consequently our algorithm can be adopted in various process conditions and used as the key subroutine for character selection and distribution in higher-level EBL stencil planning.

The rest of the chapter is organized as follows. Section 1.2 formulates the overall optimization problem. Then the polynomial time optimal algorithm is provided in Section 1.3. In Section 1.4, we prove the optimality of our algorithm and analyze its complexity. Experimental results are reported in Section 1.5, and finally, Section 1.6 concludes the chapter.

### 1.2 Problem Formulation

In this chapter, we target solving the 1 D row ordering problem for stencil planning in EBL. Given a set of $n$ characters $C=\left\{c_{1}, c_{2}, \ldots, c_{n}\right\}$, where each character $c_{i}$ has left blank margin $l_{i}$ and right blank margin $r_{i}$ as shown in Fig. 1.2 (a), we can generate a set of blank margin pairs associated with $C$, denoted by $C_{p}=\left\{\left(l_{1}, r_{1}\right),\left(l_{2}, r_{2}\right), \ldots,\left(l_{n}, r_{n}\right)\right\}$. By reorder$\operatorname{ing} C_{p}$, a sequence of pairs can be obtained, which is denoted by $S_{p}=$ $\left\{\left(l_{s_{1}}, r_{s_{1}}\right),\left(l_{s_{2}}, r_{s_{2}}\right), \ldots,\left(l_{s_{n}}, r_{s_{n}}\right)\right\}$. We define its cost by the total length of blank margins occupied by all characters after blank margin sharing, as described in Eq. 1.1:

$$
\begin{equation*}
\operatorname{Cost}_{S_{p}}=l_{s_{1}}+\sum_{i=1}^{n-1} \max \left(r_{s_{i}}, l_{s_{i+1}}\right)+r_{s_{n}} \tag{1.1}
\end{equation*}
$$

For example in Fig. 1.2(b), if we place the three characters in the order of $\left\{c_{i}, c_{k}, c_{j}\right\}$, the sequence cost would be $l_{i}+\max \left(r_{i}, l_{k}\right)+\max \left(r_{k}, l_{j}\right)+l_{j}$. On the other hand, if we reorder them to be $\left\{c_{j}, c_{i}, c_{k}\right\}$, the sequence cost can be reduced accordingly. Based on that, we define the row ordering problem.


Figure 1.2: Area saving by blank margin overlapping.

Row Ordering Problem (ROP): Given a set of blank margin pairs $C_{p}$, find its optimal order $S_{p}$, such that the sequence cost $\operatorname{Cost}_{S_{p}}$ is minimal.

### 1.3 Algorithm

In this section, we will illustrate and discuss our algorithm step by step. In Section 1.4, we will prove the optimality and the time efficiency.

First, we will give a lower bound of the Cost for all the possible solutions. Next, we will discuss the feasibility issue of the lower bound solution and some notation will be defined. Finally, we will solve the feasibility issue by presenting an minimum spanning tree-based algorithm.

### 1.3.1 From Order to Matching

We create a complete bipartite graph $G$, namely $K_{N, N}$ by making all left blank margins $r_{i}$ as indices in one set and all right margins $l_{i}$ as indices in the other set, and connect all possible $l_{i}$ and $r_{i}$ as shown in Fig. 1.3(a). The edge $\left(r_{x}, l_{y}\right)$ has the weight $e_{i}=\max \left(r_{x}, l_{y}\right)$, and means that the original pairs $\left(l_{x}, r_{x}\right)$ and $\left(l_{y}, r_{y}\right)$ can be connected in the order of $\left(l_{x}, r_{x}\right)\left(l_{y}, r_{y}\right)$. For an order of the pairs, there is a corresponding matching between the left and right blank margins $l_{i}$ and $r_{i}$ in the bipartite graph. As shown in Fig. 1.3(b), for an order $S_{p}=\left\{\left(l_{1}, r_{1}\right),\left(l_{2}, r_{2}\right), \ldots,\left(l_{n}, r_{n}\right)\right\}$, we connect the adjacent $r_{x}$ with $l_{x+1}$, and the edges not in the matching are not shown. For instance, for two pairs $\left(l_{1}, r_{1}\right)$ and $\left(l_{2}, r_{2}\right)$ which are ordered as $\left(l_{1}, r_{1}\right),\left(l_{2}, r_{2}\right)$, we create
an edge to connect $r_{1}$ and $l_{2}$. In this way, we have a matching as shown in Fig. 1.3(b), in which all numbers are connected by an edge except for $l_{1}$ and $r_{n}$. We call the matching with one edge less than the perfect matching as almost-perfect matching. If we add a dummy edge of $l_{1}$ and $r_{n}$, then we have a perfect matching of the graph. As a result, the optimization problem becomes the following.

Weighted Almost-Perfect Matching Problem (WAMP): Find an almost-perfect matching in $G$ by deleting one of the matching edges from a perfect matching in $G$, such that a set of edges $E_{s}$ of the matching is able to define an order of the given pairs with the lowest Cost, where Cost $=$ $\sum_{i=1}^{N-1} e_{i} \forall e_{i} \in E_{s}$.

Also, we use Cost* to represent the cost of a perfect matching, and Cost is the cost of almost-perfect matching. Note that, though an order has a corresponding matching, conversely an almost-perfect matching is not always able to define an order of the pairs. One counterexample is shown in Fig. 1.4(b) and will be illustrated in Section 1.3.2. The problem as well as the general WAMP will be addressed in the following sections.


Figure 1.3: Perfect and almost-perfect matching between two arrays.

### 1.3.2 Lower Bound by Sorting and Naive Matching

Intuitively matching two numbers with a big difference is not desirable because it means a waste of potential to save more area. This leads us to first sort the numbers. We sort the left blank margins $l_{i}$ and right blank margins $r_{i}$ independently in the descending order of their value, as illustrated in Fig. 1.4. For future convenience, we denote the sorted array of left components $l_{i}$ as $L$ and the sorted array of right components as $R$, as shown in Fig. 1.4(b). In order to match up numbers with the smallest differences, we adopt a naive matching strategy by connecting the $r_{i_{x}}$ and $l_{i_{x}}$ with the same index $x$ in the sorted array. Specifically, if any pair of entries $r_{x}$ and $l_{y}$ have the same array index after sorting, we connect them with an edge as illustrated in Fig. 1.4(b), meaning that the original pairs $\left(l_{x}, r_{x}\right)$ and $\left(l_{y}, r_{y}\right)$ are arranged in the order of $\left(l_{x}, r_{x}\right)\left(l_{y}, r_{y}\right)$. Once all left and right components are connected, we have a perfect matching for all the numbers, namely an assignment for their neighbors.


Figure 1.4: Sorting two arrays.
However, as mentioned before, perfectly matching $R$ and $L$ by the same index probably will not result in a valid ordered sequence of pairs, but one
or multiple cycles. For instance, if after sorting we have $R$ and $L$ as shown in Fig. 1.5(a), we will end up with one cycle of pairs corresponding to the perfect matching. On the other hand, if $R$ and $L$ are ordered as shown in Fig. 1.5(b), we will have three cycles.

So Cost* of perfect matching can also represent the cost of cycles, and Cost can represent the cost of almost-perfect matching or a sequence. On the other hand, by the naive matching strategy, we claim that this perfect matching with the same index will give us a lower bound of the Cost* ${ }^{*}$, which is $\operatorname{Cost}_{I D E A L}^{*}$. If the solution set of perfect matching is $\Omega$, then we have the following lemma.

Lemma 1. $\operatorname{Cost}_{I D E A L}^{*} \leq \operatorname{Cost}_{\omega}^{*}$ for all $\omega \in \Omega$
The proof will be given in Section 1.4. Then we discuss the cases of one and multiple cycles.

| R | L |
| :---: | :---: |
| $r_{3}$ | $\mathrm{I}_{7}$ |
| $r_{2}$ | $\mathrm{I}_{1}$ |
| $\mathrm{r}_{7}$ | $\mathrm{I}_{6}$ |
| $\mathrm{r}_{8}$ | $\mathrm{I}_{4}$ |
| $\mathrm{r}_{4}$ | $\mathrm{I}_{2}$ |
| $\mathrm{r}_{5}$ | $\mathrm{I}_{3}$ |
| $\mathrm{r}_{1}$ | $\mathrm{I}_{5}$ |
| $\mathrm{r}_{6}$ | $\mathrm{I}_{8}$ |



$$
\binom{\left(I_{8}, r_{8}\right)-\left(I_{4}, r_{4}\right)-\left(I_{2}, r_{2}\right)-\left(I_{1}, r_{1}\right)}{\left(r_{6}, I_{6}\right)-\left(r_{7}, I_{7}\right)-\left(r_{3}, I_{3}\right)-\left(r_{5}, I_{5}\right)}
$$

$$
\left(I_{8}, r_{8}\right)-\left(I_{1}, r_{1}\right)
$$

$$
\left(I_{4}, r_{4}\right)-\left(I_{3}, r_{3}\right)-\left(I_{2}, r_{2}\right)
$$

(b)

Figure 1.5: Examples of one and multiple cycles.

## One cycle

In the case of only one cycle, we can simply cut a cycle into a sequence. In other words, we need to delete one of the edges in $G$ in order to get an
almost-perfect matching from a perfect matching. Here, we use Cost* to represent the cost of a cycle or a perfect matching, and Cost is the cost of almost-perfect matching. Say we have a cycle $\beta$ that needs to be cut into a sequence $B$; its cost is defines as:

$$
\begin{align*}
\operatorname{Cost}_{\beta}^{*} & =\max \left(l_{\beta_{1}}, r_{\beta_{N}}\right)+\Sigma_{i=1}^{N-1} \max \left(r_{\beta_{i}}, l_{\beta_{i+1}}\right) \\
& =\operatorname{Cost}_{B}-\phi, \text { where } \phi \in L \bigcup R \tag{1.2}
\end{align*}
$$

To obtain the almost-perfect matching with the smallest cost increment $\phi$ from perfect matching, we pick the edge of the smallest number in the set of $L \bigcup R$ to delete, because deleting one edge means breaking one of the $N$ maximization braces in Eq. 1.2, and the smaller term in the brace would be $\phi$. So $\phi$ has to be the smallest number in $L \bigcup R$. This gets the almost-perfect matching and a valid sequence without losing any optimality. Since the smallest number is always in the bottom of the array $R$ and $L$, we just need to delete the last edge in the perfect matching. In the example of Fig. 1.5(a), we cut the edge $\left(r_{6}, l_{8}\right)$ and have the sequence as $\left(l_{8}, r_{8}\right)\left(l_{4}, r_{4}\right)\left(l_{2}, r_{2}\right)\left(l_{1}, r_{1}\right)\left(l_{5}, r_{5}\right)\left(l_{3}, r_{3}\right)\left(l_{7}, r_{7}\right)\left(l_{6}, r_{6}\right)$.

So by this method, we can always get the best almost-perfect matching with the smallest Cost based on a perfect matching. Then minimizing Cost is the same as minimizing Cost*. Additionally, in this case, there is only one cycle and it has the smallest Cost* already. As a result, we have the smallest Cost after deleting the last edge and obtain a valid order of pairs. Thus WAMP is solved in the case of one cycle.

## Multiple cycles

Clearly we cannot have a valid order of the pairs if we have multiple cycles. Solving it is the key part of our algorithm. The idea is to merge all cycles into one and then adopt the method in the case of one cycle to obtain a sequence. The difficulty is how to guarantee the optimality, which means having the smallest Cost* after merging. The algorithm dealing with this issue will be discussed in detail in the following sections.

To sum up, sorting and bipartite matching of numbers with the same indexes can give us an ideal case of ordering which has the smallest possible Cost, and can output an optimal solution if only one cycle is produced;
otherwise, the solution might not be valid.

### 1.3.3 Multiple Cycles Analysis

If we have multiple cycles after naive matching, the remaining problem would be how to get a feasible solution and guarantee the optimality at the same time. In this section, we will defines several notations used to address this issue in the Section 1.3.4. In order to make it clear, we use the example shown in Fig. 1.5(b) to illustrate them.

(a)
(b)

Figure 1.6: Different types of edge-switch.

Region
In the ideal case, we can divide the sorted array $R$ and $L$ into several regions such that one region represents one cycle. As shown in Fig. 1.5(b), region I represents Cycle I and similarly for regions II, III, and they are distinguished by different colors. Note that it is not necessarily true that one region is formed by consecutive matched pairs. It can consist of multiple intervals of consecutive matched pairs, i.e. region III.

Table 1.1: $\Delta$ Cost

| $\Delta$ Cost |  | Edge-switch |  |
| :---: | :---: | :---: | :---: |
|  | Type 1 | Type 2 |  |
| Relation | Type 1,2 | $\min \left(r_{i_{u}}, l_{i_{v}}\right)$ <br> $-\max \left(r_{i_{x}}, l_{i_{y}}\right)$ | $-\min \left(r_{i_{u}}, l_{i_{v}}\right)$ <br> $+\max \left(r_{i_{x}}, l_{i_{y}}\right)$ |
|  | Type 3 | 0 | 0 |

## Boundary

We use $B$ to represent the boundary between two adjacent regions in the arrays. As shown in Fig. 1.5(b), $B_{i}^{\mathrm{IIII}}$ denotes the $i^{\text {th }}$ boundary between regions I and II.

Relation
We define relation to be the value ordering of the four numbers involved in any two matching edges. As shown in Figs. 1.6(a) and (b), the numbers involved are $r_{i_{u}}, r_{i_{x}}, l_{i_{v}}, l_{i_{y}}$, and we have that $r_{i_{u}}>r_{i_{x}}$ and $l_{i_{v}}>l_{i_{y}}$. Without loss of generality, we can assume that $r_{i_{u}}>l_{i_{v}}$, because other cases with $r_{i_{u}}<l_{i_{v}}$ are just symmetrical, and we do not need to discuss them again. Then there are three cases of their relation:
Type 1: $r_{i_{u}}>l_{i_{v}}>r_{i_{x}}>l_{i_{y}}$.
Type 2: $r_{i_{u}}>l_{i_{v}}>l_{i_{y}}>r_{i_{x}}$.
Type 3: $r_{i_{u}}>r_{i_{x}}>l_{i_{v}}>l_{i_{y}}$.

Edge-switch and $\Delta$ Cost
Edge-switch basically means the exchange between two ending points of any two matching edges. It helps us merge cycles. For example in Fig. 1.7(a), there are three cycles in the ideal case. If we do two edge-switches at the boundary between $\left(r_{2}, l_{1}\right),\left(r_{3}, l_{4}\right)$ and the boundary $\left(r_{4}, l_{3}\right),\left(r_{5}, l_{6}\right)$, then three cycles get merged as shown in Fig. 1.7(b). Obviously, any two matching edges can be switched, and if they are from two different regions then two cycles get merged. $\Delta$ Cost is the increment of Cost* of the matching during an edge-switch. To make it clear, we can put edge-switch into two categories to discuss following.

Type 1: Edge-switch from non-crossing to crossing. As shown in Fig. 1.6,
from (a) to (b), it is a type 1 edge-switch, since two matching edges are not crossing each other in (a) but they are crossing in (b). We discuss its $\Delta C o s t$ in three different types of relation between the numbers involved in this edge-switch.

1. Type 1 Relation:

In Fig. 1.6(a), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ before the edge-switch. In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+l_{i_{v}}$ after the edge-switch. Thus, $\Delta$ Cost $=l_{i_{v}}-r_{i_{x}}>0$.
2. Type 2 Relation:

In Fig. 1.6(a), Cost $^{*}=r_{i_{u}}+l_{i_{y}}$ before the edge-switch. In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+l_{i_{v}}$ after the edge-switch. Thus, $\Delta$ Cost $=l_{i_{v}}-l_{i_{y}}>0$.

## 3. Type 3 Relation:

In Fig. 1.6(a), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ before the edge-switch. In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ after the edge-switch. Thus, $\Delta$ Cost $=0$.

Consequently, for type 1 edge-switch, $\Delta$ Cost $_{\text {Type }} \geq 0$.
Type 2: Edge-switch from crossing to non-crossing. As shown in Fig. 1.6, from (b) to (a), it is type 2 , and we also discuss its $\Delta$ Cost in three cases.

1. Type 1 Relation:

In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+l_{i_{v}}$ before the edge-switch. In Fig. 1.6(a), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ after the edge-switch. Thus, $\Delta$ Cost $=r_{i_{x}}-l_{i_{v}}<0$.
2. Type 2 Relation:

In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+l_{i_{v}}$ before the edge-switch. In Fig. 1.6(a), $\operatorname{Cost}^{*}=r_{i_{u}}+l_{i_{y}}$ after the edge-switch. Thus, $\Delta$ Cost $=l_{i_{y}}-l_{i_{v}}<0$.
3. Type 3 Relation:

In Fig. 1.6(b), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ before the edge-switch. In Fig. 1.6(a), Cost $^{*}=r_{i_{u}}+r_{i_{x}}$ after the edge-switch. Thus, $\Delta$ Cost $=0$.

Consequently, for type 2 edge-switch, $\Delta \operatorname{Cost}_{T y p e 2} \leq 0$.
From the case study above, we can find that the value of $\Delta$ Cost can be determined in Table 1.1. So, if the two numbers on one side are both larger than the two on the other side, namely type 3 relation, the $\Delta$ Cost of the edge-switch is always zero. Otherwise, the absolute value of $\Delta$ Cost is the


Figure 1.7: Edge-switch-on-boundary and edge-switch-not-on-boundary.
difference between the smaller one of the top two numbers and the larger one of the bottom two numbers. The sign of $\Delta$ Cost is determined by the type of the edge-switch.

Edge-switch-on-boundary
We define edge-switch-on-boundary literally as an edge-switch which takes place right on the boundary such that all four numbers involved are located right on the boundary. As shown in Fig. 1.7(b), there are two edge-switch-on-boundaries. For other cases of edge-switch, they are edge-switch-not-onboundary, i.e. the only edge-switch as shown in Fig. 1.7(c).

### 1.3.4 Cycle Merging

In this section, we solve the remaining part of the problem, which is how to address the case of multiple cycles after the naive matching. The problem is defined as a Cycle Merging Problem (CMP): Merge all cycles (regions in the sorted array $R$ and L) after naive matching into one cycle by finite steps of edge-switches such that total $\Delta$ Cost is minimized.

We adopt a minimum spanning tree (MST) based algorithm to merge all cycles and further generate a valid order of pairs. Since the naive matching gives the lower bound of Cost*, we need to minimize the Cost* increments of the edge-switches during the merging. Thus, we start from the ideal case, and eventually get a valid solution by adopting appropriate steps of edgeswitches.

Benefitting from the sorted array, we can merge cycles by merging regions in $R$ and $L$. For any two regions, we can pick any edge from each region and switch them in order to merge the regions. But in our algorithm, we just consider the edge-switch-on-boundaries, such as the case shown in Fig. 1.7(b), because of Lemma 2.

Lemma 2. For any not edge-switch-on-boundary, the $\Delta$ Cost is equal to or larger than the summation of all $\Delta$ Cost belonging to all edge-switch-onboundaries in between.

The proof is given in the Section 1.4. Additionally, all edge-switch-onboundaries in between can merge all regions in that area instead of just two regions. For instance as shown in Figs. 1.7(b) and (c), if you choose ( $r_{1}, l_{2}$ ) and $\left(r_{6}, l_{5}\right)$ to switch like (c), it would be better to switch $\left(r_{2}, l_{1}\right)$ and $\left(r_{3}, l_{4}\right)$ as well as $\left(r_{4}, l_{3}\right)$ and $\left(r_{5}, l_{6}\right)$, because they have the smaller $\Delta$ Cost by Lemma 2 and not only merge two regions but all the three regions from (a). So we only need to consider the edge-switch-on-boundary as our potential selection for edge-switch.

With all possible edge-switch-on-boundaries and their $\Delta C o s t$, we can construct a graph $H$ by assigning a vertex for each region and connect two vertices if the regions that they represents have a common boundary. Additionally, the distance of each edge in $H$ is the $\Delta$ Cost of the edge-switch on the corresponding boundary. For example in Fig. 1.5(b), we can construct a graph as shown in Fig. 1.8(a), where $\Delta$ Cost $_{i}^{\text {I,II }}$ means the $\Delta$ Cost of the edge-switch on the $B_{i}^{\mathrm{I}, \mathrm{II}}$.

Consequently, merging all regions into one region with the smallest total $\Delta$ Cost becomes finding the MST in this graph, because the MST connects all vertices and thus all regions are merged into one if we actually switch the edge picked by the MST.


Figure 1.8: Find the optimal solution by performing the MST algorithm.

## Minimum spanning tree

We use the same example in Fig. 1.5(b) to explain our algorithm. If we have all pairs as $\left(l_{1}, r_{1}\right)=(15,3),\left(l_{2}, r_{2}\right)=(10,1),\left(l_{3}, r_{3}\right)=(8,9),\left(l_{4}, r_{4}\right)=(5,6)$, $\left(l_{5}, r_{5}\right)=(7,16),\left(l_{6}, r_{6}\right)=(12,11),\left(l_{7}, r_{7}\right)=(14,4),\left(l_{8}, r_{8}\right)=(2,17)$, then from top to bottom:

$$
\begin{aligned}
& \Delta \operatorname{Cost}_{1}^{\mathrm{II}, \mathrm{III}}=\left(r_{8}+r_{5}\right)-\left(r_{8}+r_{5}\right)=0, \\
& \Delta \operatorname{Cost}_{1}^{\mathrm{III}}=\left(r_{5}+l_{7}\right)-\left(r_{5}+l_{6}\right)=2, \\
& \Delta \operatorname{Cost}_{1}^{\mathrm{I}, \mathrm{III}}=\left(r_{6}+l_{6}\right)-\left(l_{6}+l_{2}\right)=1, \\
& \Delta \operatorname{Cost}_{2}^{\mathrm{II}, \mathrm{III}}=\left(l_{3}+l_{5}\right)-\left(l_{3}+l_{5}\right)=0, \\
& \Delta \operatorname{Cost}_{3}^{\mathrm{II}, \mathrm{III}}=\left(l_{5}+l_{4}\right)-\left(l_{5}+l_{4}\right)=0 .
\end{aligned}
$$

Then we use Kruskal's algorithm [14] to find the MST shown in Fig. 1.8(b). In this example, the MST contains $B_{2}^{\mathrm{II}, \text { III }}$ and $B_{1}^{\mathrm{I}, \mathrm{III}}$.

Edge-switch after MST
Next, we need to perform the edge-switches picked by the MST algorithm.
We switch the edge on $B_{2}^{\text {II,III }}$ and $B_{1}^{\mathrm{I}, \mathrm{III}}$ and thus obtain a valid solution of only one cycle, as shown in Fig. 1.9(b). Finally, we have the final $\operatorname{Cost}_{A L G}^{*}$ as

$$
\begin{equation*}
\operatorname{Cost}_{A L G}^{*}=\operatorname{Cost}_{I D E A L}^{*}+\Sigma \Delta \operatorname{Cost}(e), \forall e \in \mathrm{MST} \tag{1.3}
\end{equation*}
$$

There is one circumstance that we need to discuss a little more. As shown in Fig. 1.10(a), after finding the MST, if we want to do edge-switches on both $B_{1}^{\text {I,II }}$ and $B_{1}^{\text {II,IIII }}$, then as shown in Fig. 1.10(b), after we switch edges on $B_{1}^{\mathrm{I}, \mathrm{II}}$, the problem is that we no longer have the edge-switch-on-boundaries on


Figure 1.9: Merge cycles based on MST.


Figure 1.10: Edge-switch examples.
$B_{1}^{\mathrm{II}, \mathrm{III}}$ available. However, we can do the edge-switch between edge $\left(r_{i_{3}}, l_{i_{3}}\right)$ and either edge $\left(r_{i_{1}}, l_{i_{2}}\right)$ or edge $\left(r_{i_{2}}, l_{i_{1}}\right)$.

The solution is that we always easily select the edge containing the smaller value of $r_{i_{2}}$ and $l_{i_{2}}$ to switch with $\left(r_{i_{3}}, l_{i_{3}}\right)$. Without loss of generality, if $r_{i_{2}}>$ $l_{i_{2}}$, we select the edge $\left(r_{i_{1}}, l_{i_{2}}\right)$ and do the switch as shown in Fig. 1.10(d). After this step, we claim that we have the same total $\Delta C o s t=\Delta C o s t_{1}^{\mathrm{I}, \mathrm{II}}+$
$\Delta C o s t_{1}^{\mathrm{II}, \mathrm{III}}$ as what we desire. The reason is as follows.
In Fig. 1.10, we already have $\Delta \operatorname{Cost}_{1}^{\mathrm{I}, \mathrm{II}}$ toward the total $\Delta$ Cost from (a) to (b). So, in order to merge cycle II and cycle III, we just need to show that the $\Delta$ Cost of (b) to (d) is still $\Delta \operatorname{Cost}_{1}^{\text {II,III }}$ which is the $\delta \operatorname{Cost}$ of (a) to (c). The four numbers involved in those two edge-switch operations, $\left(r_{i_{2}}, l_{i_{2}}, r_{i_{3}}, l_{i_{3}}\right)$ in (a) and ( $r_{i_{1}}, l_{i_{2}}, r_{i_{3}}, l_{i_{3}}$ ) in (b) have the same relation type, because $r_{i_{1}}>$ $r_{i_{2}}>l_{i_{3}}$ implies that the largest number among the four becomes $r_{r_{1}}$ from $r_{i_{2}}$ and all other numbers stay the same. According to Table 1.1, the largest value in the relation does not affect the $\delta$ Cost of the edge-switch. Thus, from (b) to (d), the $\Delta$ Cost is still $\Delta \operatorname{Cost} t_{1}^{\mathrm{II}, I I I}=l_{i_{2}}-r_{i_{3}}$. Even if we have more consecutive edges that need to be switched, we just need to adopt this technique iteratively. As a result, we solve this problem without losing any optimality.

To sum up our algorithm to solve ROP, we first address the problem of WAMP and then obtain the optimal order of pairs with minimum Cost based on the matching. The overall flow of solving ROP is presented as the following algorithms.

```
Algorithm 1: ROP's algorithm
    Data: A set of margin pairs of \(\left(l_{i}, r_{i}\right)\)
    Result: An order of pairs with minimal Cost
    Construct a complete bipartite graph \(G\);
    Obtain an almost-perfect matching by solving WAMP;
    return the order determined by the almost-perfect matching;
```

```
Algorithm 2: WAMP's algorithm
    Data: Graph \(G\) built by pairs of \(\left(l_{i}, r_{i}\right)\)
    Result: A minimal Cost almost-perfect matching creating an order
                of pairs
    Sort \(r_{i}\) and \(l_{i}\) respectively and do Naive Matching;
    switch Number of cycles after the naive matching do
        case One cycle do
            Delete the last edge in the sorted \(G\);
            return the almost-perfect matching;
        case Multiple cycles do
            Merge all cycles by solving CMP;
            Go to case One cycle;
```

```
Algorithm 3: CMP's algorithm
    Data: Cycles (regions) determined by naive matching
    Result: One cycle (region) with minimal sum of all \(\delta\) Cost
    Construct a graph \(H\) by regions and Boundaries;
    Find the MST in \(H\) by Kruskal's algorithm;
    3 Switch the edges picked by the MST;
    return the cycle after edge-switches;
```


### 1.4 Proof

In this section, we will prove the optimality of the algorithm we presented in Section 1.3.

Note that there might be more than one optimal ordering, but our algorithm can only output one of them. If we have the optimal solution OPT which has smaller cost Cost $_{O P T}$ than our algorithm's $\operatorname{Cost}_{A L G}$, we will show that this is not possible. We also use $\operatorname{Cost}_{\text {IDEAL }}$ to represent the case just after the naive matching with the lower bound of Cost. Because of the reason stated in Section 1.3.1, we think of ordering as matching instead, in other words, proving the optimality of WAMP instead of proving ROP directly. Note that if we just have one cycle after the naive matching, then we use cut strategy to have the optimal solution. So we just need to consider the multiple cycles case and its $\operatorname{Cost}_{I D E A L}^{*}, \operatorname{Cost}_{A L G}^{*}$ and $\operatorname{Cost}_{O P T}^{*}$. In order to determine the relationship between the ideal case and all possible ordering, we have the following lemma.

Lemma 3. Any perfect matching in $G$ can be achieved by finite steps of type 1 edge-switch with $\Delta$ Cost $\geq 0$ from the ideal case.

Actually, because all type 1 edge-switches have non-negative $\Delta$ Cost, we can see that

$$
\begin{equation*}
\text { Lemma } 3 \Rightarrow \text { Lemma } 1 \tag{1.4}
\end{equation*}
$$

Thus, proving Lemma 3 can be applied to prove Lemma 1.

## Proof of Lemma 3:

Base step: (1) $N=1$. As shown in the Fig. 1.11(a), there is only one possible ordering. (2) $N=2$. There are two possible matching cases as shown respectively in Fig. 1.11(b) and (c). One type 1 edge-switch can be done from the ideal case (b) to (c).


Figure 1.11: Base cases.

Inductive hypothesis:
Assume that when $N=k$, any perfect matching can be achieved by finite steps of type 1 edge-switch from the ideal case.

When $N=k+1$, we have $k+1$ pairs. Say we have an arbitrary perfect matching between $R$ and $L$ as shown in Fig. 1.12(a). As shown Fig. 1.12(b), we have another perfect matching in Fig. 1.12(b) which is the same as in Fig. 1.12(a) except for the edges between $r_{i_{1}}, l_{i_{1}}, r_{i_{x}}, l_{i_{y}}$. It is obvious that from Fig. 1.12(b) to Fig. 1.12(a), we just need one type 1 edge-switch step. For (b), the bottom $k$ pair matchings, by the hypothesis, can be transformed from the ideal case by finite steps of type 1 edge-switch. Thus, with one more edge-switch of type 1 , we can always achieve an arbitrary perfect matching of $k+1$ pairs. So the lemma is true. Next, we prove Lemma 2.


Figure 1.12: Inductive steps.

## Proof of Lemma 2:

Refer to Fig. 1.7. If we do an edge-switch-not-on-boundary of $\left(r_{x}, l_{x}\right)$ with $\left(r_{y}, l_{y}\right)$ for $0 \leq x<y \leq N$, and there are $M$ edge-switch-on-boundaries of $\left(r_{m_{i}}, l_{m_{i}}\right)$ with $\left(r_{m_{i}+1}, l_{m_{i}+1}\right)$ such that $m_{i} \geq x$ and $m_{i}+1 \leq y$, then we want to show that edge-switch-not-on-boundary $\Delta C o s t^{n b}$ is larger or equal to $\Sigma_{i=1}^{i=M} \Delta$ Cost $_{m_{i}}^{o b}$ where $\Delta$ Cost $_{m_{i}}^{o b}$ represents the $\Delta$ Cost for the edge-switch of $\left(r_{m_{i}}, l_{m_{i}}\right)$ with $\left(r_{m_{i}+1}, l_{m_{i}+1}\right)$.

1. Type 3 relation of $r_{x}, r_{y}, l_{x}, l_{y}$
$\Delta$ Cost $^{n b}=0$ by Table 1.1. Additionally, since $r_{x}>r_{y}>l_{x}>l_{y}$, $m_{i} \geq x$ and $m_{i}+1 \leq y$, we have $\forall m_{i}, r_{m_{i}}>r_{m_{i}+1}>l_{m_{i}}>l_{m_{i}+1}$ and $\Delta$ Cost $_{m_{i}}^{o b}=0$. Thus, $\Delta$ Cost $^{n b}=\sum_{i=1}^{i=M} \Delta$ Cost $_{m_{i}}^{o b}=0$, and the lemma is true in this case.
2. Type 1 and 2 relations of $r_{x}, r_{y}, l_{x}, l_{y}$

Assume that we have a counterexample such that $\Delta \operatorname{Cost}^{n b}<\sum_{i=1}^{i=M} \Delta \operatorname{Cost}_{m_{i}}^{o b}$. Then by Table 1.1, we have

$$
\begin{aligned}
\Delta \operatorname{Cost}^{n b}< & \Sigma_{i=1}^{i=M} \Delta \operatorname{Cost}_{m_{i}}^{o b} \\
\min \left(r_{x}, l_{x}\right)- & \max \left(r_{y}, l_{y}\right) \\
< & \underbrace{\min \left(r_{m_{1}}, l_{m_{1}}\right)}_{<0} \\
& \underbrace{-\max \left(r_{m_{1}+1}, l_{m_{1}+1}\right)+\min \left(r_{m_{2}}, l_{m_{2}}\right)}_{<0} \\
& -\max \left(r_{m_{2}+1}, l_{m_{2}+1}\right)+\ldots+\min \left(r_{m_{M}}, l_{m_{M}}\right)
\end{aligned}, \quad \begin{aligned}
& \min \left(r_{m_{1}}, l_{m_{1}}\right)-\max \left(r_{m_{M}+1}, l_{m_{M}+1}\right)
\end{aligned}
$$

Note that $-\max \left(r_{m_{i}+1}, l_{m_{i}+1}\right)+\min \left(r_{m_{i+1}}, l_{m_{i+1}}\right) \leq 0, \forall i \leq M$, since $r_{m_{i}+1}$ and $l_{m_{i}+1}$ are above the $r_{m_{i+1}}$ and $l_{m_{i+1}}$ in the sorted $R$ and $L$ arrays. But it is not possible because $\min \left(r_{x}, l_{x}\right) \geq \min \left(r_{m_{1}}, l_{m_{1}}\right)$ and $\max \left(r_{y}, l_{y}\right) \leq \max \left(r_{m_{M}+1}, l_{m_{M}+1}\right)$. Thus, there is no counterexample and the lemma is true in this case.

So, Lemma 2 is true.

## Proof of optimality:

If we have the optimal solution $O P T$, by Lemma 3, it can be achieved by finite steps of type 1 edge-switch from the ideal case. At the starting point of the ideal case, we have multiple cycles, but for any one of those switches, if its four numbers involved are all inside the region, it cannot make any contribution to merge the cycles. Thus, some of those switches must be cross two regions. Besides, all regions must be merged, so those switches must touch all cycles. Thus, those switches can construct an spanning tree in a graph $H^{\prime}$ where there is one vertex for each cycle (region), and there is an edge $(u, v)$ with $\Delta \operatorname{Cost}_{(u, v)}$ for one of all possible switches that crosses any two different regions $u$ and $v$. Next, we just need to prove that this spanning tree in $H^{\prime}$ has the total $\Delta$ Cost larger than or equal to the total $\Delta$ Cost of the MST in $H$ defined in Section 1.3.4.

By Lemma 2, $H^{\prime}$ can be transformed into $H$ by a way that for every edge of edge-switch-not-on-boundary, replace it by one or more edges of edge-switch-on-boundary, and then merge the edges with the identical $\Delta$ Cost. Additionally by the Lemma 2 , after the transformation, the smallest $\Delta$ Cost between any two vertices stays the same. Thus, the MST in $H$ is also the MST in $H^{\prime}$. As a result, we prove that the spanning tree of $O P T$ in $H^{\prime}$ has the total $\Delta$ Cost larger than or equal to the total $\Delta$ Cost of the MST in $H$ by our algorithm. By Eq. 1.3, $\operatorname{Cost}_{O P T}^{*} \geq \operatorname{Cost}_{A L G}^{*}$. Then, by Eq. 1.2 Cost $_{\text {OPT }} \geq$ Cost $_{A L G}$. OPT could not be more optimal than $A L G$, so our algorithm can output an optimal solution.

The overall running time of our algorithm is definitely polynomial. Sorting and doing the naive matching to obtain the ideal case takes $O(N \log N)$ time, where $N$ is the size of the set of characters. For finding the MST in the graph $G$, the number of edges in $G$ is at most $N$, since the number of boundaries in the sorted arrays is at most $N$. Consequently, it can be done within $O(N \log N)$ by Kruskal's algorithm.

Generally, we have found that all possible solutions can be transformed from the ideal case and our algorithm can give the optimal solution which complete the transformation with the smallest Cost increment. Thus, our algorithm can generate the optimal solution for stencil row planning problem.
Table 1.2: Comparisons with previous algorithm

| $\# C P$ | $[10]$ |  | $[11]$ |  |  | ALG |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Cost | CPU(s) | Cost | CPU(s) | Cost | CPU(s) | Improve $[10]$ | SpdUp $[10]$ | Improve $[11]$ | SpdUp $[11]$ |
| 96 | 104000 | 6.54 | 132400 | 0.099 | 95924 | 0.00178 | $7.7 \%$ | x 3666 | $27.5 \%$ | x 55.5 |
| 294 | 306000 | 306.3 | 389980 | 0.104 | 288664 | 0.005627 | $5.6 \%$ | x 54427 | $26.0 \%$ | x 18.5 |
| 495 | 505000 | 2376.3 | 657980 | 0.122 | 478280 | 0.009335 | $5.2 \%$ | x 254555 | $27.3 \%$ | x 13.1 |
| 581 | 619000 | 3949.3 | 81900 | 0.136 | 582730 | 0.010847 | $5.8 \%$ | x 364094 | $28.8 \%$ | x 12.5 |
| 767 | NA | Hours | 1033000 | 0.157 | 766197 | 0.015108 | NA | NA | $25.8 \%$ | x 10.4 |
| 929 | NA | Hours | 1271000 | 0.172 | 937129 | 0.017747 | NA | NA | $26.3 \%$ | x 9.7 |

### 1.5 Experimental Results

We implement our algorithm in $\mathrm{C}++$ and test it on a Linux workstation 2.5 G Hz CPU and 126 GB memory. Since the previous works [11, 10, 12] use some assumptions to generate characters having similar left and right blank margins, which might not be realistic, we create our own benchmarks with a set of characters with blank margins generated randomly. They are controlled to be less than the actual character size (the area impossible to be overlapped). Input is a certain set of characters, and output is the minimum total length of those characters in a row. We run our algorithm and only algorithms of $[11,10]$ on our benchmarks, since [12] is shots saving driven and cannot insert more characters comparing to the other two. The comparison result is shown in Table 1.2. The number of characters is reported in column 1. The total length of blank margins, namely Cost, and running time are reported for all three algorithms. Speed-up and length improvement are also calculated. Because of [10]'s running time issue, the last two test cases are not reported for it.

As shown in Table 1.2, comparing to [10], we can improve the result by more than $5 \%$ and it is orders of magnitude faster, because [10] uses the Hamiltonian path based method, which approximates the result but is still not efficient, especially when the number of characters is growing. By the comparison to [11], we can improve the result a lot and also have good speedup. And the runtime confirms that the time complexity of our algorithm is $O(n \log n)$ where $n$ denotes the number of characters. According to [10], solving row ordering problem consumes most part of the running time of the entire stencil planning flow. So our algorithm can be adopted and have a great impact of performance on the solution of the overall problem. As for the quality of the result, because of the limited data set that we have right now, those heuristics with assumptions might perform poorly on future industrial data. On the other hand, since we have already proved the optimality of our algorithm, we can always achieve the best solution theoretically and the experiments also agree with that. Adopting our algorithm, we save space in a row, thus more characters can be inserted into the stencil and further reduce the number of shots needed to print the layout. Additionally, our algorithm becomes essential if the number of characters is large.

### 1.6 Conclusion

In this chapter, we propose a polynomial time algorithm to solve the 1D row ordering problem optimally for EBL stencil planning. Optimality is proved theoretically, and the high quality as well as the high efficiency of our algorithm are also verified by the experiments compared with previous works. In the CP technology, our algorithm serves as a key subroutine for the high-level character selection and distribution problem. Those problems are proved NP-hard, but any solution of them can still benefit significantly from our algorithm.

## CHAPTER 2

## MODEL-BASED MULTIPLE PATTERNING LAYOUT DECOMPOSITION

### 2.1 Introduction

Conventional lithography (193i) has reached its limit, as the minimum feature size is consistently shrinking below 10 nm technology node. Other emerging alternative lithography technologies such as E-beam [15], Extreme ultraviolet (EUV) and Derected Self-Assembly (DSA) [16, 4] have been proposed and researched for decades. However, E-beam suffers from its low throughput issue. EUV and DSA have many fabrication process challenges to solve before it is used for high volume manufacture. Thus, Multiple Patterning Lithography (MPL) is widely adopted along with 193i in industry as the favorite advanced resolution enhancement technique. Moreover, MPL could also be used as a hybrid with EUV and DSA, if the minimum feature size keeps shrinking.

Because of the optical diffraction effects, small features or ones that are too close to each other cannot be printed by a single exposure. More specifically, features within the minimum distance $d_{\text {min }}$ are defined as conflicted and have to be printed separately by different exposures in MPL. Thus, different masks are needed, and the most challenging issue of MPL becomes how to decompose the layout into different masks, e.g. two masks if Double Patterning Lithography (DPL) and three masks if Triple Patterning Lithograph (TPL). Traditionally, the decomposition is done by assigning the features within $d_{\text {min }}$ into different masks. It is called Rule-Based Decomposition (RBD), and most research efforts have been devoted to it. Yu et al. [17] propose an ILP-based algorithm, but it suffers exponential runtime. A semi-definite programming technique is used to improve the runtime. However, it may run into sub-optimal solution. Fang and Pan [18] use a graph based method, which cannot always find a solution even if it exists and also relatively gen-
erates more stitches. Tian et al. [19] propose a polynomial algorithm for row structure layout, which can solve RBD optimally.

However, using the minimum distance as the only criteria to separate features is obviously inadequate, for example, it does not consider the interaction of near field waves and the fact that close proximity effect may be beneficial. As shown in Fig. 2.1(b), the middle polygon has better corner rounding if two polygons are closer than $d_{\text {min }}$ comparing to Fig. 2.1(a) where two polygons are far apart. However, RBD may separate the two polygons in Fig. 2.1(b) into two different masks and the corner rounding would be like in Fig. 2.1(a). In general, RBD is not accurate. Model based decomposition (MBD) is consequently needed to improve the actual printability. MBD decomposes the layout into multiple masks based on optical simulations and aims for achieving best printability on all the masks. The quality of the decomposition is determined by the Edge Placement Error (EPE) or Intensity Log Slope (ILS) of the simulation result for all masks. To our best knowledge, there are several works on MBD, Rodrigues and Kundu [20] introduce a model based double patterning decomposition method based on simulated annealing. But both of the optical simulations and the convergence of simulated annealing are very time consuming. According to its experimental results, thousands of the polygons need to be processed for more than 10 hours. On the other hand, there may be millions or even billions in the layout of big designs, so this work not only cannot guarantee optimality by using simulated annealing, it is also considered impracticable. Recently, ASML [21] proposes a patent to solve the MBD for multiple patterning. It creates simulation points along the features' boundaries and keeps track of their ILS from the simulations and then iteratively change the decomposition trying to improve the result. (Though the experimental result or implementation detail is not revealed to public, the convergence of this method could have very large running time by nature.) Also another major drawback is that many stitches can be potentially produced in this method, and it is well known that the stitches increase the difficulty of manufacture. The detailed comparison to previous works will be illustrated in Section 2.2. In general, model based decomposition potentially consumes more computational resources comparing to RBD, since it needs to simulate the patterns by optical model for exponential times which could be very expensive in terms of running time. Additionally we cannot construct a conflict graph of features and run graph
algorithms on it like RBD, since there is limited information of one decomposition solution's quality unless it is actually simulated. So the problem is far more difficult than RBD. For a standard cell design, we propose a novel framework to solve MBD for a whole layout in a reasonable runtime. We first preprocess the standard library by simulations, secondly build our library for possible local decomposition solutions, and finally construct a graph, where a shortest path algorithm runs to select the optimal decomposition solution. The details will be explained in following sections.


Figure 2.1: (a)-(b), MBD has better corner rounding. (c)-(d), other frameworks versus ours.

The chapter is organized as following. The motivations and the literature reviews are given in Section 2.2. The preprocessing step is discussed in Section 2.3. The algorithm is illustrated in Section 2.4. Finally, the experimental
result is shown in Section 2.5.

### 2.2 Motivation

In this section, we introduce some background terminologies and explain the motivations of our work by analyzing the drawbacks of the previous work. The flow of our framework is also illustrated.

The runtime of MBD comes from two part: simulation and optimization. It is well known that optical simulations are time consuming. However, both previous works [20, 21] use a similar framework that iteratively assigns some features into different masks and then do the simulation to evaluates the improvement and decide whether the new assignment is accepted or not, as shown in Fig. 2.1(c). This strategy may have runtime issues, since its convergence may be slow in some cases and also may get stuck in the local optimal solution. It is even worse that for every iterative step, we need to do optical simulations. So we have to iterate through the loops of optimization phase and simulation phase, which is extremely inefficient. They use some techniques trying to reduce the runtime. We use the term ambit(AM) for the distance that optics have notable influence. The first technique is that instead of simulating the full layout, we only re-simulate the ambit area of the changing features in every iteration. We adopt the similar idea but use windows, which will be explained in Section 2.3. The second type of technique is to speed up the convergence. Rodrigues et al. [20] use simulated annealing, while Socha [21] relies on the gradient and the Hessian of ILS regarding to mask assignment to indicate whether a mask movement is the most beneficial.

Nevertheless, the iterative method itself by nature is inefficient and possibly gets sub-optimal results in some cases, since it lacks the global view of the optimization and wastes computational resources on local optimization moves. Simulated annealing [21] is very consuming according to the experimental results, and only thousands of features can be processed. Though Socha [21] does not reveal experiment data nor its implementation detail, but it is easy to see that it potentially consumes lots of runtime, since it fragments the layout and creates many evaluation points, which enlarges the input set of the program. Additionally, it may creates many unnecessary
stitches. The difficulty of the problem becomes how to reduce the runtime but be able to achieve the quality of MBD for the entire layout. Thus, as shown in Fig. 2.1(d), we propose a framework that instead of doing optimization and simulation in series. We first preprocess the simulations of some patterns to get enough information of possible layout decompositions and build a library based on it. Secondly we construct a graph by scanning through the layout and looking up the library. Finally we run an efficient algorithm on the graph to solve for the best mask assignment. In general, our framework does the simulation and optimization in parallel such that the optimization does not need to wait for the simulation, and the simulations can be reused for a different layout since they just need to be done once for the standard cell library.

### 2.3 Preprocessing

Since we assume that the layout design is based on a standard cell library, there are many frequent patterns in the layout. Thus, we can preprocess the simulations to save runtime. The preprocess is done by three steps: (1) defining the window, (2) getting a decomposition solution set for individual windows, and (3) getting the decomposition solution set for consecutive windows. The goal is to build a library such that when we solve the decomposition of the whole layout, we do not need to re-simulate but rather look up the available solutions.

### 2.3.1 Windows Creation

First, we define a window to be the unit area that is actually simulated. In our framework, the height of the windows is selected as the height $H$ of all standard cells, and the width is selected as the minimum width $W$ of all standard cells. The size of the windows is a design decision and can be designed differently, such as the largest common factor for all standard cells. However, we will show later that our framework can work with any window width less than $W$. As an example standard cell library shown in Fig. 2.2(a), the window size is determined as the size of an inverter. Next, all other cells in the library are divided into unit windows and we create a window library


Figure 2.2: Standard cells are split into windows.
$\pi=\left\{w_{i}\right\}$. Note that we have partial window sometime, but it could be handled. The total of different windows in the library is $N$. In the example, we have a set of 11 different windows $w_{1}$ to $w_{11}$. If the layout is produced by the standard cells, it is also covered by window library $\pi$. For the layout shown in Fig. 2.2(b), it is covered by patterns from $w_{1}$ to $w_{11}$. Note that the windows are overlapped vertically at the power tracks, but we assume that all the power tracks are preassigned to mask 1.


Figure 2.3: Each window has a set of possible solutions.

### 2.3.2 Solution Set of a Window

Secondly, we build a solution set $\alpha_{i}=\left\{S_{k}^{i}\right\}$ for each window $w_{i}$, where $S_{k}^{i}$ represents each solution. Since the number of features in one window is limited (usually less than 10), thus we can easily simulate all enumerations of features' mask assignments (decompositions) and get their EPE values. In $\alpha_{i}$, we record a cost $S_{k}^{i}$ to indicate the quality of the $k^{t h}$ solution of $w_{i}$, where the cost is defined as the EPE value. Note that we preassign all the power track into one of the mask, say mask 1 . Considering the memory usage, one option is that we can only record the $K$ best solutions. So $K$ could be altered by designers. In this chapter, we use $K=3$ for illustration convenience and the experiments are also done under the same assumption. As shown by an example in Fig. 2.3, for one row in the layout, we have the solution sets from $\alpha_{1}$ to $\alpha_{11}$.

### 2.3.3 Solution Set of Consecutive Windows

However, the optical influence can go across the windows, thus it is not sufficient to just record the cost of the solutions of individual windows. Because of the standard cell design assumption, the layout is row-based as shown in Fig. 2.2(b). Additionally the power tracks are preassigned and they are much thicker than features, thus the optical influence between vertical windows can be neglected [19]. Horizontally, we have ambit $A M$ defined as the optical influence distance. So for each window $w_{i}$, we group all windows within the ambit, say they have indices from $r_{1}$ to $r_{C}$, where $C$ is the number of windows in the group. Then we simulate all possible combinations of the available solution from each of their solution set $\alpha_{r_{i}}$ if $1 \leq i \leq C$, and we call those windows the relatives of $w_{i}$. For convenience, we assume that the ambit is the same with the unit width of a window. As an example shown in Fig. 2.4(a), $w_{2}$ 's simulation quality can be affected by $w_{1}$ and $w_{3}$, thus all combinations of $S_{x}^{1}, S_{y}^{2}$ and $S_{z}^{3}$ are simulated and each one gets a cost $S_{x y z}^{123}$, where $x, y, z$ are the indices of the solutions perspectively. In other words, we pick one solution for each window in the group, which is illustrated by one line with a different color in the figure, and do the simulation for the whole group based on the selected solutions. Besides, $w_{5}$ shows an example of more than three consecutive windows' solutions that need to be simulated
together, in which the partial window is handled. As a result, we can build a solution set $\beta_{r_{1}, \ldots, r_{C}}$ for window sequence $w_{r_{1}}, \ldots, w_{r_{C}}$ and keep a record of their costs. If all $\alpha_{i}$ have size $K$, then the size of $\beta_{r_{1}, \ldots, r_{C}}$ is $K^{C}$, where $C$ is the total number of consecutive windows in ambit. In the example, $C\left(w_{2}\right)=3$.

The maximum number of simulations depends on the maximum number of possible window sequences in the cell library and the number of available solutions for each sequence, namely the maximum size of all $\beta_{r_{1}, \ldots, r_{C}}$ which is $K^{C_{\text {max }}}$. Here, we give a loose upper bound of the number of possible window sequences. Suppose that the standard cell library has size of $M$, and the length of the longest window sequence $C_{\max }=\max \left(C\left(w_{1}\right), \ldots, C\left(w_{i}\right)\right)$. If the sequence crosses multiple cells, it generates at most $M^{C_{\max }} * L_{\max }$ possible sequences, where $L_{\max }$ is the largest number of windows in one cell. The reason is that the window sequence crosses at most $C_{\text {max }}$ cells, and in the first cell we have at most $L_{\max }$ choices of the starting window. If the sequence is in one cell, then the number of possible sequences is at most $M\left(L_{\max }-2\right)$, since there are $M$ cells, and in each of them there are $\left(L_{\max }-2\right)$ possible sequences, since the sequence has the minimum length of three. Thus the bound of the total number of simulations is $K^{C_{\max }}\left(M^{C_{\max }} * L_{\max }+M\left(L_{\max }-2\right)\right)$. Since the term $M\left(L_{\text {max }}-2\right)$ is small compared to $M^{C_{\max }} * L_{\text {max }}$, we can neglect it. Thus, we have $L_{\max } K^{C_{\text {max }}} M^{C_{\text {max }}}$. It is obvious that the bound is very loose. First, in the extreme case that every cell has only one window, we could achieve $M^{C_{\text {max }}}$, while in practice, we may just have one cell that contains one window. This will dramatically reduce the number of possible window sequences. Also, we could not always achieve $C_{\max }$ and $L_{\max }$, and not all cells have chance to be adjacent. So to reduce the number of simulations, one technique is that we can simply go through the layout and remember all possible sequences of windows for each $w_{i}$.

To sum up, we first define the window, and at the second step do simulation on individual windows, the at the third step do simulations on consecutive windows. Compared to simulate the whole layout, we simulate a much smaller area of a window or window sequence each time, so the runtime of each simulation is relatively small. Further, instead of having an extremely large number of possible decomposition solutions of the layout if we simply enumerate it, which has runtime growing exponentially, the second step narrows down the solution space by looking at the local window area and the third step is to capture the influence between the windows. In terms of the
running time, the number of the simulation is bounded and not exponentially increasing with the number of polygons or cells. As a result, we have a library that contains pre-calculated solutions of window sequences for look-up. The decision of choosing a solution for a window will be done in a graph.


Figure 2.4: In (a), by the different combinations of solutions in each window, we have a solution set for each window sequence. Different color shows different combinations. In (b), a graph is constructed and the shortest path is found.

### 2.4 Algorithm

In this section, we firstly show how to construct a graph based on the library that we build in Section 2.3. Secondly, the algorithm to optimally solve the solution selection problem is illustrated.

After having a library of solutions of all window sequences, we construct a graph to abstract the layout by window sequences and their solutions sets. Based on the assumptions in the previous section, we just need to consider one row of the layout at a time. As shown in the Fig. 2.4(b), we create
the graph $G(V, E)$ by going through the row of windows. For each window, we get the window sequence $w_{r_{1}}, \ldots, w_{r_{C}}$ that is covered by ambit, and then look it up for its solution set $\beta_{r_{1}, \ldots, r_{C}}$. Next for each solution in the set, we create a vertex with a weight equaling to the $\operatorname{cost} S_{x_{1}, \ldots, x_{C}}^{r_{1}, \ldots, r_{C}}$. As shown in the Fig. 2.4(b), if $K=2, w_{2}$ has $K^{3}=8$ vertices. At window $w_{i}$, for one vertex, create a directed edge from itself to all vertices at the next window if those vertices have the same solutions for all overlapped windows. In the example shown in Fig. 2.4(b), vertex $S_{111}^{123}$ has two out-going edges, one to $S_{111}^{234}$ and the other one to $S_{112}^{234}$, since they use the same solution for window $w_{2}$ and $w_{3}$. As a result, we have a graph $G$ such that vertices present the possible solutions of a window, and edges represent its compatibility with the surrounding windows' solutions that affect simulation of this window. Note that one path in $G$ from the rightmost window to the leftmost window means a valid solution selection for each window. Since we have each vertex with a weight of cost to record the simulation quality, a shortest path in $G$ gives us the optimal solution selection for all windows, which also solves the MBD problem. After creating a source node $S$ and a sink node $T$ as shown in Fig. 2.4(b), the shortest path from $S$ to $T$ can be done in polynomial time by Dijkstra's algorithm.

| Number of <br> polygons | Rodrigues et al. <br> [ISQED'11] <br> Run-time | Rodrigues et al. <br> [GLSVLS'10] <br> Run-time |
| :--- | :--- | :--- |
| 558 | $3 m 22 s$ | $36 m 54 s$ |
| 932 | $4 m 32 s$ | $2 h 14 m 13 s$ |
| 1911 | $1 h 8 m 5 s$ | $8 h 52 m 12 s$ |
| 2234 | $1 h 54 m 11 s$ | $10 h 32 m 45 s$ |
| 2347 | $1 h 10 m 12 s$ | $11 h 12 m 14 s$ |

(a)

(b)

Figure 2.5: (a) shows the previous works. (b) shows the optimality of our shortest path algorithm.

### 2.5 Experimental Result

We implement our algorithm in $\mathrm{C}++$ and test it on a Linux workstation with four 3.00 GHz CPUs and 190 GB memory. The main issue of MBD is runtime, so we mainly compare and discuss the running time. We create a set of 100 random windows, and based on the windows we build two benchmarks of 30 standard cells and 50 standard cells respectively. Each cell has the largest length of 10 windows. Then we generate a random layout different sizes shown in Table 2.1. We keep three best decomposition solutions of each window with a random cost value 1 to 10 . We assign the cost of a window sequence by summing up the costs of individual windows and random connectivity costs from 1 to 10 for every adjacent two window pair. For quality, we compare the total cost of the shortest path with the cost of a random path. The result is shown in Fig. 2.5(b), x-axis is the number of windows in the layout and $y$-axis is the total cost. We reduce nearly half of the cost. For the runtime, as shown in Fig. 2.5(a), previous works [22, 20] can process only thousands of polygons within hours, which has a scalability issue. For our runtime, there are two parts, preprocessing and the shortest path algorithm. For runtime report, we use Calibre Workbench V2013 by Mentor Graphic to do the simulations. For the area of a window sequence, the average time of pure optical imaging time is 0.24 s . Then we multiply this time with the number of all possible sequences of windows with solutions to estimate the runtime of preprocessing. As shown in the Table 2.1, if we have 50 cells with 100 windows in our standard cell library and about 1 million windows with 0.15 million cells in the layout, we need less than 10 hours to solve MBD. The shortest path algorithm is very fast and relatively neglectable. So for our framework, we can see that it is possible to process a full layout with a million windows within hours. The preprocess only needs to be done once, so as long as we have the standard cell library preprocessed, finding the optimal MBD is very efficient. Additionally, since all window sequences are independent to each other, so the preprocessing actually can be done in parallel, which would further reduce the runtime.

Table 2.1: Runtime analysis

| Cell library size |  | Layout size |  | Our algorithm runtime |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| \# windows | \# cells | \# windows | \# cells | Preprocessing | Shortest path |
| 100 | 30 | 99675 | 15181 | 3 h 15 m | 22.0 s |
| 100 | 30 | 996621 | 147465 | 3 h 15 m | 217.9 s |
| 100 | 50 | 99635 | 14549 | 7 h 25 m | 22.2 s |
| 100 | 50 | 996491 | 154347 | 7 h 45 m | 220.2 s |

### 2.6 Conclusion

In this chapter, we propose a novel framework of model-based multiple patterning layout decomposition. We first preprocess the standard cell library and then optimize the decomposition by constructing a graph and running a polynomial time optimal algorithm on it. In terms of the major runtime issue of this problem, our method is far more practical comparing it to previous works. By this work, it is possible to process a large layout with millions of features.

## CHAPTER 3

# COLORING RECTANGULAR AND DIAGONAL GRID GRAPHS FOR MULTI-PATTERNING LITHOGRAPHY 

### 3.1 Introduction

Rectangular grid graphs (RGGs) and diagonal grid graphs (DGGs), formed as induced subgraphs of regular square grids and diagonal grids shown in Fig. 3.1(a) and (b) respectively, are well defined and widely used in graphical representation of geo-location, placement/route, urban planning, etc. [23, 24]. The application can be further extended to cover any problem where instances are regularly distributed and sized. Their intrinsic regularity gives us leverages to solve hard graph problems by efficient algorithms. However, the coloring problem on those graphs, especially for DGG, does not draw much attention. In this chapter, we discuss the k-coloring problem along with its variants on RGGs and DGGs, as they can be directly applied to several scenarios in the multiple patterning problem of design for manufacturing (DFM).

In the territory of DFM, multiple patterning lithography (MPL) technique that k-partitions the layout, is still a challenging design problem, yet to be solved with a more complex process in the sub-10 nm technology node [25]. Given that metal layers lean into a regular 1D design, where the coloring is studied heavily and can be helped by using cuts [26], the outstanding challenge concerns via/contact layer decomposition. As via/contact layer layouts can be extremely complex for random logics and no stitches can facilitate the coloring, unlike metals, their decomposition problem must be addressed with great care in order to adopt MPL.

However, if we match the challenges in multi-patterning on via/contact layers with the opportunities provided by RGG/DGG, we can find natural correlations, as illustrated by Figs. 3.1(a), (b) and (c). First, due to many advantages of simple design rules over complex ones [27, 28], especially in fa-


Figure 3.1: (a) The corresponding rectangular grid graph induced from a full rectangular grid for the contact/via layout in (c). (b) The corresponding diagonal grid graph induced from a full diagonal grid for the contact/via layout in (c). (c) A sample contact/via layout.
vor of design specialists, regular/semi-regular 1D layouts are most commonly adopted for metal layers. As a result, the vias and contacts are usually on grid [29] and the grid size is most likely equally distributed by virtue of the regular design rule and the concern over OPC/SRAF-insertion. Second, the color conflict during the multiple patterning is defined by design rules to avoid optical diffractions. Design rules are usually distance based, thus nodes within a distance are considered conflicted and connected by edges [30]. Such distance can be a multiple of the grid cell length: when it is between 1 and $\sqrt{2}$ cell lengths, the conflict graph naturally forms a RGG as shown in Fig. 3.1(a); when it is between $\sqrt{2}$ and 2 cell length, it falls into a DGG as shown in Fig. 3.1(b). Those two cases are the most fundamental and commonly appear in today's designs. Additionally, even if some outlier contact/vias might be slightly off-grid, the conflict graph is still very likely a
subgraph of RGG or DGG.
Those natural connections provide a perfect solution to address the multipatterning lithography on via/contact layer through RGG and DGG. Unfortunately, we are shorthanded from existing results. Several previous works have worked on the problem of partitioning layouts, but they aimed at solving a relatively general graph. Tian et al. proposed a polynomial algorithm of 3 -coloring a row-structure layout [19]. Kuang et al. [31] utilized graph simplification techniques to reduce the problem size and assumed that the resulting subproblem left is of trivial size. This method can hardly be used for via/contacts, as the conflict graph may remain large after adopting those techniques. In [32], the major contribution focuses on inserting stitches that are not possible for contact/vias. Yu et al. modeled the coloring into integer linear programming (ILP) and also solved it by a semidefinite programming approximation [17]. Zhang et al. provided solutions based on a randomized iterative method using pairwise coloring [33]. Nevertheless, none of them targets at contact/via layer and is able to take advantage of the regularity of the conflict graphs. On the other hand, coloring RGG and DGG is also open in graph theory area, though many theorems exist on coloring graphs and planar graphs [34]. Therefore, for both of theoretical and practical purposes, it is significantly valuable to investigate the coloring property of RGG and DGG.

In this chapter, we completely analyze the k-coloring problem on RGG and DGG, and claim that all except for 3-coloring DGG are tractable. Moreover, we prove that the 3 -coloring DGG graph is actually NP-complete, which means that we are not able to have an efficient algorithm to color a via/contact layer by three colors unless NP $=\mathrm{P}$. This result also implies that 3 -coloring 1D metal layers without using cuts is NP-complete, because the conflict graph of a via layer can be treated as a special case of a 1D metal layer if every metal polygon is as small as a via. Besides, coloring properties and some 3 -colorable subclasses of DGG are explored. Based on that, an exact algorithm is proposed to handle sufficiently large DGGs, and the experiments show that our algorithm has good performance with much better results over heuristics.

The rest of the chapter is organized as follows. In Section 3.2, we define the notations of graphs and their coloring problems. In Section 3.3, we analyze the problem of coloring RGG. In Section 3.4, we discuss how to color DGG
and provide several sufficient conditions. We prove the NP-completeness of 3 -coloring DGG in Section 3.5. In Section 3.6, we propose an exact algorithm of 3-coloring DGG and discuss its effectiveness and performance by experiments. Finally, we conclude the chapter in Section 3.7.

### 3.2 Problem Definition

We provide a formal definition of our problems in this section. We first define the rectangular and diagonal grid graphs, namely RGG and DGG.

## Definition 1. Rectangle grid graph (RGG) / Diagonal grid graph

 ( $\boldsymbol{D} \boldsymbol{G} \boldsymbol{G}$ ): Given a grid $Z_{r}^{2}\left(Z_{d}^{2}\right)$ whose vertices correspond to the points with integer coordinates in the plane, and in which two vertices are connected by an edge whenever the corresponding points are within distance $1(\sqrt{2})$, a rectangle (diagonal) grid graph is an induced subgraph of $Z_{r}^{2}\left(Z_{d}^{2}\right)$.Then we define our coloring problems on the two graph classes:
Definition 2. $\boldsymbol{K}$-coloring an $\boldsymbol{R} \boldsymbol{G} \boldsymbol{G} / \boldsymbol{D} \boldsymbol{G} \boldsymbol{G}$ : Given an $R G G\left(V_{R}, E_{R}\right)$ or $D G G\left(V_{D}, E_{D}\right)$, assign each $v \in V_{R}\left(V_{D}\right)$ a color such that $\forall(u, v) \in E_{R}\left(E_{D}\right)$ $u$ and $v$ do not share the same color and the total number of colors used $k$ is minimized.

We will study the k -coloring problems in the following sections.

### 3.3 Coloring a Rectangular Grid Graph

We first investigate the problem on RGG. By intuition, we first ask: When is RGG 1-colorable? Obviously any edge will make the coloring impossible. So RGG is 1-colorable if and only if it is isolated.

Then the next question is: When can RGG be colored by two colors? As shown in Fig. 3.2(a), an RGG cannot contain an odd cycle. The reason is as follows: If we have a clockwise orientation for any cycle in RGG, the cycle must have equal numbers of $\uparrow$ edges and $\downarrow$ edges, so the total number is even. Similarly it is also true for $\leftarrow$ edges and $\rightarrow$ edges. Thus the total number of edges, which is the same as the number of vertices in a cycle, is even. As a


Figure 3.2: (a) No odd cycle in a rectangular grid graph. (b) Coloring a full rectangular grid. (c) Coloring a full diagonal grid.
result, RGG is 2-colorable. Alternately, we observe that RGG is a subgraph of a full rectangular grid, which can be colored by two colors as illustrated in Fig. 3.2(b), so we have the following theorem.

Theorem 4. $R G G$ is 2-colorable.
Consequently there is no need to discuss its k-coloring for $k>2$.


Figure 3.3: (a) An example of DGG is not 3-colorable due to $K_{4}$. (b) An example of DGG is not 3-colorable because all red vertices must have the same color. (c) A DGG is not 3-colorable, but it is 3-coloring critical, since if we remove the red vertex, it will be 3-colorable. (d) A 3-colorable DGG with maximum degree less than 4.

### 3.4 Coloring a Diagonal Grid Graph

Similarly to RGG, DGG is 1-colorable if and only if it is isolated. Its 2colorability can be checked by a linear algorithm of searching for odd cycles. Then the following question is whether DGG can be 3-colored. If not, what is the minimum number of colors to guarantee a valid coloring? The answer to the latter question is 4. As shown in Fig. 3.2(c), a full diagonal grid can be colored by 4 . Therefore, as its induced subgraph, DGG is always 4 -colorable. We have the following.

Theorem 5. The $k$-coloring problem of $D G G$ is solvable in polynomial time except for $k=3$, and $D G G$ is always 4-colorable.

However, the remaining problem of 3-coloring becomes difficult. Before the analysis of 3-coloring, we can always conduct graph simplification techniques to remove vertices that have degree 1 or 2 , or are cut vertex, and delete edges that are cut or in a two-edge cut pair, as the 3 -colorability would not be affected [31]. Note that DGG could potentially be divided into smaller components, but the results may not be trivial to conduct enumeration (exponential) algorithms as stated in [31]. W.L.O.G, we assume that the DGG discussed in the rest of the chapter is a connected component free of those vertices and edges.

In fact, many structures that are not 3-colorable could appear in DGG as shown in Figs. 3.3(a), (b) and (c). For example in Fig. 3.3(a), DGG can potentially contain a $K_{4}$ (complete graph of four vertices), which cannot be 3 -colored and thus spoils the 3-colorability of the whole graph. $K_{4}$ can be easily found by checking whether a tile of the grid has all of its 4 vertices in the DGG. Then it is natural to ask about the 3-colorability of DGG without $K_{4}$. To simplify the illustration, we use $G$ refer to a DGG not containing $K_{4}$.

As shown in Fig. 3.3(c), $G$ is a planar graph consisting of triangles and polygon faces. Its maximum degree $\Delta(G)$ is 6 , as shown by the black vertex. Thus, $G$ is considered sparse and seems to be a restricted subclass of planar graphs. Thus, we first apply our knowledge in graph theory, in this case some sufficient conditions of 3-coloring, to discuss the property of DGG.

### 3.4.1 The Maximum Degree

First of all, the maximum degree of a graph is highly related to the coloring problem. Intuitively, the smaller the degree of a vertex, the greater its flexibility for coloring. Since the maximum degree of our graph $\Delta(G)=6$, it is natural to ask what happens if $\Delta(G) \leq 5$; however, $G \leq 5$ is not strong enough to guarantee the 3 -colorability as shown in Fig. 3.3(b). On the other hand, $\Delta(G) \leq 3$ is able to imply the 3-colorability of $G$, because if a connected graph $G^{\prime}$ is not a complete graph or an odd cycle, then its chromatic number equals its maximum degree $\Delta\left(G^{\prime}\right)$ by Brook's theorem. The only case in which $G$ is a complete graph is when it is a triangle that is 3 -colorable, and the cases in which $G$ is an odd cycle can be 3-colored too. As an example shown in Fig. 3.3(d), a $G$ with $\Delta(G) \leq 3$ is colored by three colors, and we have the following lemma.

Lemma 6. $G$ is 3-colorable if its maximum degree is less than 4.


Figure 3.4: The path of the outer face makes (a) a $45^{\circ}$ turn, (b) a $90^{\circ}$ turn, (c) a $90^{\circ}$ turn, (d) a $135^{\circ}$ turn, (e) a $135^{\circ}$ turn. (f)-(h) show the cases of possible neighbors of $u$. (i) An orientation forms a path of the outer face.

### 3.4.2 Triangles and G without Diamond

Triangles always make the graph hard to be 3-colored. Inspired by that, researchers have worked to prove the 3-colorability for graphs with constraints on triangles. The first famous result is Grotzsch's theorem, which states that a triangle-free planar graph is 3 -colorable. Besides, [34] has shown that graphs with a small number of triangles (fewer than four) or with triangles far


Figure 3.5: (a) Two diamonds $\left(K_{4}-e\right)$. (b) A chain of diamonds. (c) An invalid connection of three diamond chains that introduces a $K_{4}$ in red dashes. (d) A valid connection of 4 diamond chains.
apart from each other (distance is at least three) are likely to be 3-colorable. In this section, we improve those conditions for our graph $G$ and show that $G$ without diamond (two triangles sharing an edge) is 3-colorable. We first prove the following lemma.

Lemma 7. Suppose that $G^{*}$ is a $G$ without diamond. There exists a vertex $v$ in $G^{*}$ such that $d(v) \leq 2$.

Proof. Take one component of $G^{*}$ and delete all leaves (degree 1), which does not affect the colorability. Then create a clockwise orientation of its boundary (the outer face) as shown in Fig. 3.4(i). There always exists a vertex $v$ on the boundary such that the angle $\theta$ from its incoming edge to its outgoing edge is less than $180^{\circ}$; namely the path of the boundary has to make a convex turn at some point, since the boundary encloses an area. As shown in Figs. 3.4(a) to (c), for the cases of $\theta=45^{\circ}$ to $\theta=90^{\circ}, v$ has the maximum degree 2; otherwise, a $K_{4}$ or a diamond is created. When $\theta=135^{\circ}$, we have two possible cases as shown in Figs. 3.4(d) and (e). In Fig. 3.4(d), v cannot have degree 3, otherwise a diamond is created. For the case in Fig. 3.4(e), $v$ is able to have degree 3, but for vertex $u$, if $u$ has no other neighbor as shown in Fig. 3.4(e) then $u$ is the vertex with degree less than 3. If $u$ has a neighbor as shown in Figs. 3.4(f) and (g), then a diamond appears, which is not allowed. If $u$ has a neighbor shown in Fig. 3.4(h), then the path of the directed edges $e_{1}, e_{2}, e_{3}$ retains the direction of $e_{1}$ and thus contradicts the fact that a turn has to be made. Therefore, there is a vertex with degree at most 2 in all cases, and the lemma is true.

Theorem 8. $G^{*}$ is 3-colorable.

Proof. Based on the lemma, it implies that $G^{*}$ is a 2-degenerate graph, which implies the 3 -colorability. There is always a vertex $v_{i}$ with degree at most 2 and deleting it will not affect the 3-colorability of the graph. Thus, if we keep deleting the smallest degree vertex (degree 1 or 2 ) from $G^{*}$ until the last vertex, the remaining vertex can be assigned any color, and $G^{*}$ can be 3 -colored in the way of adding those deleted vertices back iteratively because in each iteration at most two edges are created. So the proof is complete.

To sum up, we have several results on the 3-colorability of DGG. However, none of them is a necessary and sufficient condition, which leads us to develop optimal algorithms. Indeed, the problem is NP-complete as explained in the next section.

### 3.5 3-Coloring a Diagonal Grid Graph

In this section, we demonstrate the hardness of 3-coloring DGG and provide an NP-complete proof. We utilize a common structure in DGG, diamond ( $K_{4}-e$ ), to establish our proof. As shown in Fig. 3.5(a), two red vertices of a diamond must have the same color.

If we connect the diamonds as a chain, shown in Fig. 3.5(b), all red vertices must share the same color and all green vertices must be assigned another color, and it is the same for the remaining vertices. Thus, a part of $G$ could be a chain of diamonds as shown in Fig. 3.5(b) and is highly constrained for coloring. Then interactions between diamond chains become the key to solve for 3 -coloring. For example, in Fig. 3.3(b), a loop-back of a diamond chain makes it impossible to 3 -color, since two red vertices have to share the same color but they are connected by an edge. The key observation is that vertices in a diamond chain, such as the red vertices in Fig. 3.5(b), must share the same color and thus can be treated as a single vertex. Additionally all green red vertices and red vertices must have different colors. Thus, if we use one node to present all red vertices and one node for green vertices, we can have an edge connect the two nodes to enforce the distinct colors. Therefore, it might be possible to construct a random graph by "vertice" and "edge" formed by diamond chains. Based on the idea, the remaining section will discuss the hardness of this problem and prove the following theorem.

Theorem 9. Suppose $D G G$ is an induced subgraph of a diagonal grid. The problem of determining its 3-colorability is NP-complete.

Nevertheless, the construction is not straightforward, as the regularity and sparsity of DGG may forbid many structures. The problems whether we can connect a point to anywhere with only two directions of chains and whether a point is able to connect to enough other points need to be addressed. For instance, chains cannot be connected arbitrarily because of the limitation on the degree of a vertex. As shown in the example of Fig. 3.5(c), three diamond chains tend to be incident to one vertex but it introduces a forbidden $K_{4}$, while it is possible to have four diamond chains incident to one vertex as shown in Fig. 3.5(d). So the construction is critical.

The general idea of the proof for Theorem 9 is a polynomial reduction from the planar graph 3-coloring problem, which is known as NP-complete. The outline of the proof is as follows: First, we transform an instance $H\left(V_{h}, E_{h}\right)$ of planar graph into a straight line embedding and expand its vertices to blocks. Second, we construct a rectilinear embedding and prove that it occupies a polynomial area on the grid in order to ensure the reduction is polynomial. Third, the rectilinear embedding can be subdivided by replacing its vertices and edges with a chain of triangles so that an instance of $G$ is obtained. Then solving 3 -colorability for the resulting $G$ can be used to solve the 3colorability of arbitrary $H$, and then 3 -coloring DGG can be proved as an NP-complete problem.

### 3.5.1 Straight Line Embedding and Vertex Expansion

A straight line embedding $E^{S}$ exists for any simple planar graph $H\left(V_{h}, E_{h}\right)$ by Fary's theorem [34], as shown in Fig. 3.6. However, a stronger rectilinear embedding is needed to construct $G$. On the other hand, a rectilinear embedding is only possible when the degree of all vertices is at most 4 . Therefore, we introduce a technique to expand vertices of $H$ by replacing them with blocks.

Definition 3. Vertex expansion: There are two steps of vertex expansion. First, given a straight line embedding $E_{H}^{S}$, replace each $v_{i} \in V_{h}$ by a square block $p_{i}$ that is chosen to have the edge length as the degree of $v_{i}$ such that there are enough access points on both left and right sides of the block.


Figure 3.6: Straight line embedding of a planar graph.

Second, for each edge $\left(v_{i}, v_{j}\right) \in E_{h}$, connect pin $v_{i, k}$ in $p_{i}$ and pin $v_{j, l}$ in $p_{j}$ such that the geometric relation of pins preserves the ordering of incident edges.

We name the resulting graph after expansion as $\bar{H}$. Because the vertex expansion preserves the circular ordering of edges, the new edges remain straight and non-crossing. As a result, there is always a valid vertex expansion; for example, the graph in Fig. 3.7(a) is obtained by applying vertex expansions on the graph in Fig. 3.6(b).

We horizontally stretch and rotate $\bar{H}$ such that we can divide the plane into n slabs $S_{1}, S_{2}, S_{3}, \ldots, S_{n}$, and each $S_{i}$ solely contains $p_{i}$ as shown in Fig. 3.7(b). The intersection points of the slab boundary and the edges of $\bar{H}$ are indexed as $u_{i, k}$ if $u_{i, k}$ is on the right boundary of $S_{i}$ and it is the $k^{t h}$ intersection point from top to bottom along that boundary.

### 3.5.2 Rectilinear Embedding

We follow the definition of rectilinear embedding in [35]. In order to construct the target $G$, we want to obtain a rectilinear embedding $E_{H}^{R}$ of $H$ first. Additionally, we need to restrict the area occupied by the rectilinear embedding to a polynomial of the size of $H$ in order to ensure the polynomial reduction. At the initial step, $p_{1}$ is added onto the grid as well as its edges $\left(v_{1, l}, u_{1, m}\right)$. Since it is the leftmost block, and all its edges go to blocks on the


Figure 3.7: The graph after vertex expansion and dividing the plane into slabs.
right, $\left(v_{1, l}, u_{1, m}\right)$ is assigned to a horizontal line from $v_{1, l}$ to the boundary of $S_{2}$. At an inductive step shown in Fig. 3.8(b), everything up to the boundary of $S_{k+1}$ has been constructed as a rectilinear embedding, and $p_{k+1}$ needs to be placed onto the grid. We want to make an arrangement of $p_{k+1}$, edges $\left(u_{k, m}, v_{k+1, l}\right)$ and edges $\left(u_{k, m}, u_{k+1, n}\right)$ such that rectilinearity is satisfied and all edges can proceed right to the boundary of $S_{k+2}$. At first, we consider the placement of $p_{k+1}$. For edges entering $S_{k+1}$ through $u_{k, m}$ as shown in Fig. 3.8(b), enforced by the embedding $\bar{H}$ in Fig. 3.7(b), they follow a strict order from the top to bottom:

1. edges (black) not going to connect $p_{k+1}$ (through $u_{k, 1}, u_{k, 2}$ and $\left.u_{k, 3}\right)$.


Figure 3.8: The inductive steps of constructing a rectilinear embedding.
2. edges (purple) going to incident to pins of $p_{k+1}$ (through $u_{k, 4}, u_{k, 5}$ and $u_{k, 6}$ ).
3. edges (black) not going to connect $p_{k+1}$ (through $u_{k, 7}$ and $u_{k, 8}$ ).

We define the navy region as the half-plane above the $u_{k, m_{-}}$if $u_{k, m_{-}}$is the highest $u_{k, m}$ connecting to $p_{k+1}$ according to $\bar{H}$ in Fig. 3.7(b). Similarly, the purple region is defined as opposite to the navy region. We first place $p_{k+1}$ such that its highest pin on the left side matches up with the edge from $u_{k, m_{-}}$ vertically and the edge can naturally extend to it as shown in Fig. 3.8(b). Note that the block size is controlled to guarantee sufficient pins on both sides and every intersection point with the grid is an available pin. The graph in the navy region can remain unchanged and the edges can extend to the next slab, but operations are needed for the components in the purple region, since the pins are not aligned with edges. As shown in Fig. 3.8(c), when $p_{k+1}$ is small, we need to stretch the edges such that we can route to the pins. This results in an enlarged gap between blocks, and the gap is bounded by the degree of $p_{k+1}$. In Fig. 3.8(d), when $p_{k+1}$ is large and some edges need to detour to the next slab, we move all blocks in the purple region down and some edges between navy and purple regions are stretched. The operations of moving and stretching are shown in Fig. 3.8(a). Then, we are able to achieve a rectilinear embedding from $S_{1}$ to $S_{k+1}$. By induction, we can obtain the rectilinear embedding $E \frac{R}{H}$ by such procedures.

### 3.5.3 Construct G from the Rectilinear Embedding

At the final step, we construct our target $G$ based on the rectilinear embedding $E \frac{R}{H}$ by replacing its blocks and edges by pre-designed structures.

## Edge design

A rectilinear edge can be designed as a combination of structures in Fig. 3.9(a). Based on the coloring property of diamonds, $(a, b)$ are enforced to have the same color by a diamond chain, while $(c, d)$ are enforced to have different colors by the structure named as bridge. Thus, by connecting them, we can obtain any edge $\left(v_{i, j}, v_{k, l}\right)$ in $E \frac{R}{\bar{H}}$ such that ( $v_{i, j}$ and $v_{k, l}$ ) have distinct colors. Nevertheless, we need to avoid a rhombus that will produce a $K_{4}$ in
G. Thus, we have to adopt some operations for those structures. As shown in Figs. 3.10(b) and (d), each one has a rhombus and we flip the chain to resolve it as shown in Figs. 3.10(c) and (e) respectively. In this way, two pins connected by an edge in $E \frac{R}{H}$ are ensured to have distinct colors.


Figure 3.9: (a) Transformations of rectilinear edge into a chain of diamonds or a bridge. (b) A block with four pins on each side is designed as chains of diamonds. (c) A DGG obtained by rotating the block in (b) by $45^{\circ}$.

Block design
We want to design the block in such a way that all its pins $v_{i, j}$ have to be colored the same. We utilize chains of diamonds again by connecting them in a way shown in Fig. 3.9(b). The polygon at the center formed by a chain has all its cornered vertices the same color and they reach out to pins on block boundaries. In this way, the design of a block with any number of pins can be established. As a result, the rectilinear embedding can be transformed into a graph with those structures.


Figure 3.10: (a) A forbidden rhombus. (b) A forbidden pattern when a diamond chain turns. (d) A forbidden pattern when diamond chains connect. (c) and (e) show flipping the chain at the half point to resolve the forbidden patterns.

Grid rotation
At the last step, as shown in Fig. 3.9(c), we rotate the resulting graph by $45^{\circ}$ to get $G$.


(e)
(c)


(f)

Figure 3.11: The process to construct an instance of G for H in Fig. 3.6(b) is sketched from (a) to (b).

### 3.5.4 NP-Completeness

Following the construction procedures, which are sketched in Fig. 3.11, we can draw an induced subgraph of a diagonal grid $G$ without containing any $K_{4}$ for arbitrary planar graph $H$ such that determining the 3-colorability of $H$ is the same as determining the 3 -colorability of $G$. We also prove that the reduction is polynomial:

Lemma 10. The size of the grid containing $G$ built by the construction is polynomial to the size of $H$.

Proof. First of all, the width of the rectilinear embedding $E \frac{R}{\bar{H}}$ is bounded by the total width of blocks and gaps between them. Second, since rectilinear edges are replaced by the structure in Fig. 3.9(a), each gap has width as a polynomial function of vertex degree in $H$. A block is replaced by the structure in Fig. 3.9(b), and its width is proportional to the degree of the corresponding vertex as well. Third, following the same logic, the height of the $E \frac{R}{H}$ is bounded by the total height of blocks, which is the same as its width. Note that there is no gap between blocks vertically by our construction, and the degree of vertex is bounded by the number of vertices. Therefore, the total area is bounded by the product of width and height which is a polynomial function of $V_{h}$.

Because 3-coloring $H$ is NP-complete and DGG without $K_{4}$ is a subclass of DGG, Theorem 9 is proved.

(a)

(b)

Figure 3.12: (a) Two diamond vertices are glued. (b) A diamond chain is contracted to a single triangle.

### 3.6 An Exact 3-Coloring Algorithm and Its Experiments

The previous sections conclude the k-coloring properties of DGG, but do not give out a practical solution to the problems like triple patterning contact/vias due to the intractability of 3 -coloring DGG. On the other hand, in this section, we learn from the proof and propose an exact algorithm to 3-color DGG. Though it has exponential complexity, but we will show that it is indeed efficient for considerably large graphs.

At first, we introduce a technique to contract diamond chains. For a diamond shown in Fig. 3.12(a), $v_{1}$ and $v_{2}$ must have the same color, then we can glue them such that all their neighbors are connected to the new vertex $v_{1 / 2}$. The resulting graph retains its 3 -coloring property. By applying a sequence of the gluing operations, a chain of diamonds can be contracted to one triangle, as shown in Fig. 3.12(b). The pseudo code is shown in Algorithm 4.

Though the diamond contraction technique was commonly observed by researchers, it is very effective in the case of 3-coloring DGG and the reasons are as follows. We can categorize DGG into two types: (1) Sparse and easy to color. (2) Dense and hard to color. For type 1, the algorithm should be able to return the result in a relatively timely manner. For type 2 , it requires extra computational resources, since an exact algorithm needs to explore a much larger solution space to make sure that it is not colorable. However, by the proof, we know that the hardness of coloring is mainly due to diamond chains and their interactions. Therefore, if we can contract all diamond chains, the problem size for type 2 DGG can be reduced dramatically. Based on the observations, we propose an exact algorithm based on the following two steps: (1) We apply diamond contraction. (2) We adopt maximum-degree-of-saturation based (DSATUR) backtracking, which picks the most saturated vertex (has the least number of available colors) to color first and backtracks when a vertex is saturated. The pseudo code is illustrated in Algorithm 5.

To evaluate our algorithm, we randomly generate a set of $N \times N$ grids with density $d$ and obtain their DGG. We exclude the cases that contain $K_{4}$, since its uncolorability will be returned immediately. We also run UTDecomposer (UTD) in [17] on those graphs to compare with our algorithm, as it is the

```
Algorithm 4: Contract diamond chains
    Data: A graph \(G(V, E)\)
    Result: A graph \(G^{\prime}(V, E)\) free of diamond chain
    while diamond exists do
        Find any diamond \((a, b, c, d)\) with \((b, c)\) as the shared edge of two
        triangles;
        For any edges \(\left(v_{i}, a\right) \in E\) such that \(v_{i} \in V /\{b, c\}\), replace it by
        \(\left(v_{i}, d\right)\);
        Remove vertex \(a\);
    return \(G^{\prime}\);
```

```
Algorithm 5: Our exact algorithm
    Data: A graph \(G(V, E)\)
    Result: Whether \(G(V, E)\) can be 3-colored
    Function Backtracking ( \(G(V, E)\) )
        if all vertices are colored then
            return True;
        Update the saturation \(\operatorname{sat}\left(v_{i}\right)\) for all uncolored vertices, where
        \(\operatorname{sat}\left(v_{i}\right)\) equals the number of distinct colors of neighbors;
        Pick the most saturated vertex \(v_{x}\);
        if \(\operatorname{sat}\left(v_{x}\right)\) is 3 then
            return False;
        forall possible color \(c_{j}\) for \(v_{x}\) do
            Color \(v_{x}\) by \(c_{j}\);
            if Backtracking \((G)\) returns True then
                return True;
        return False;
    Function Main ( \(G(V, E)\) )
        Contract diamonds in \(G\) by Algorithm 1;
        Find the maximum clique and color it by distinct colors;
        return Backtracking \((G)\);
```

newest layout decomposer and has the best performance and quality of results among [32, 31, 33] (see Section VIII of [17] for details). Note that UTD is an application-orientated software and is designed to color metal layers which have more general conflict graphs, while our algorithm just targets at coloring DGG. Our exact algorithm is implemented in C++ and we obtain the binary of UTD from its website. The experiments are conducted on a four-core Linux machine with 3.2 GHz and 24 GB memory. At each $N$ and $d, 1000$ random grid graphs are generated and used as benchmarks. In order to examine the robustness of the algorithms regarding to the grid size, we make the mean of $d$ to be $50 \%$ and run experiments on various grid sizes from $N=60$ to $N=400$. As shown in Table 3.1, the number of 3 colored DGGs, the number of uncolorable ones and the average runtime over the 1000 testcases are reported in columns 2-4 and 5-6, 8 for our algorithm and UTD respectively. Based on the data, our algorithm can efficiently (average runtime less than 7 minutes) handle the large grids with size up to $400 \times 400$, which potentially could have about $16 \times 10^{4}$ vertices. What is more important is that our algorithm is optimal while UTD mistakenly determines many colorable cases to be uncolorable and colors them with conflicts. The colorability accuracy of UTD is calculated as the percentage of the colorable cases found in all colorable DGGs and it is reported in column 7. As shown in the column, the accuracy of UTD reduces rapidly with increasing grid size and it can barely produce an optimal coloring when $N$ is over 300. Additionally, our algorithm has average runtime on a single core while the SDP solver in UTD takes advantage of the four cores in our machine. In the future, our algorithm has potential to be parallelized. If we consider a more dense layout and increase the grid density to $60 \%$, as shown in Table 3.2, our algorithm can still handle $100 \times 100$ grids efficiently. On the contrary, except that the accuracy remains low, UTD's runtime increases significantly so that the experiment on $100 \times 100$ grids is too time-consuming to collect data.

Indeed, for any algorithm that checks the 3-colorability optimally, its runtime should increase exponentially as the graph size according to Theorem 9. However, according to the experiments, our algorithm shows superior effectiveness and performance. Compared to the best existing heuristic UTD, our algorithm can determine the colorability of the graph much more accurately, and it is extremely important to provide such optimal coloring results to

Table 3.1: Comparisons between UTD and our algorithm for less dense DGGs

| $\mathrm{d}: 50 \%$ | Ours |  |  | UTD [17] |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $N$ | \# C | \# UnC | CPU | \# C | \# UnC | Accuracy | CPU |
| 60 | 995 | 5 | 0.99 s | 715 | 285 | $71.86 \%$ | 1.16 s |
| 80 | 991 | 9 | 0.99 s | 559 | 441 | $56.41 \%$ | 1.53 s |
| 100 | 983 | 17 | 1.65 s | 362 | 621 | $36.83 \%$ | 1.90 s |
| 200 | 910 | 90 | 23.28 s | 8 | 992 | $0.88 \%$ | 6.81 s |
| 300 | 780 | 220 | 118.46 s | 0 | 1000 | $0 \%$ | 16.66 s |
| 400 | 735 | 265 | 383.03 s | 0 | 1000 | $0 \%$ | 30.86 s |

Table 3.2: Comparisons between UTD and our algorithm for dense DGGs

| $\mathrm{d}: 60 \%$ | Ours |  |  | UTD [17] |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $N$ | \# C | \# UnC | CPU | \# C | \# UnC | Accuracy | CPU |
| 40 | 877 | 123 | 1.00 s | 102 | 898 | $11.63 \%$ | 14.25 s |
| 60 | 728 | 272 | 2.146 s | 3 | 997 | $0.41 \%$ | 165.27 s |
| 80 | 602 | 398 | 7.27 s | 0 | 1000 | $0 \%$ | 928.36 s |
| 100 | 339 | 661 | 19.52 s | - | - | - | - |

layout designers in order to avoid additional cost by using EBL or DSA to remedy the conflicts. Moreover, considering that the via/contact layers are very dense, especially in advanced technology nodes, our algorithm shows greater performance over the heuristic in such cases. Besides, even though it is not scalable theoretically, it can be used as a subroutine to facilitate other algorithms or heuristics. For example, in [31], a graph is partitioned into smaller components. Instead of assuming that it has size less than 7 and otherwise using heuristics, our exact algorithm can be adopted. Moreover, some smart heuristics can be developed by a strategy of dividing the graph into partitions and solving them independently by our method.

Table 3.3: A complete classification of k -coloring problems

|  | 2-colorability | 3-colorability | 4-colorability |
| :---: | :---: | :---: | :---: |
| RGG | YES | YES | YES |
| DGG w/ small degree | Polynomial | YES | YES |
| DGG w/o diamond | Polynomial | YES | YES |
| DGG | Polynomial | NP-complete | YES |

### 3.7 Conclusion

Our major results are presented in Table 3.3 as a complete classification of the complexity for k-coloring problems on grid graphs. All necessary proofs are provided in previous sections. Based on those insights, they can facilitate the development of algorithms for litho-related problems, such as layout decomposition, design strategies in terms of place/route and design rule development. With our study in place, designers working on MPL can estimate the decomposition difficulty to decide the feasibility of the design pitch. When the design falls into the case of high complexity such as 3 -coloring DGG, designers can first try to avoid complex layout and obtain a simpler conflict graph, such as DGG with small degree or DGG without diamond by enforcing some design rule. Moreover, a router that incrementally assigns nets may also attempt to keep the connected component of conflicted contact/vias within a relatively small size, such as $400 \times 400$, which is highly possible in practice, such that our exact algorithm can report the 3-colorability to the router and facilitate its decision of placing next contact/via. The optimal result can be returned with high confidence in a timely manner. For designs with larger connected components, our study can be applied in a divide-andconquer fashion to solve the decomposition problem. We plan to work on it as our future works.

## CHAPTER 4

## GROUPING AND COLORING DIAGONAL GRID GRAPHS FOR DIRECTED SELF-ASSEMBLY LITHOGRAPHY

### 4.1 Introduction

As the dimensions of features keep shrinking toward sub-10 nm or further, it is impossible for conventional lithography (193i) to match the resolution requirement. Next-generation lithography techniques, such as extreme ultraviolet lithography (EUV) and electron beam lithography (E-beam) [15], have been heavily researched in the past decades. However, there are still several challenges of EUV. For instance, it has issues of mirror defects and low source power. For E-beam, it suffers from the major drawback of low throughput. Alternatively, multiple patterning lithography (MPL) [17] and block copolymer directed self-assembly (DSA) [4, 16] becomes the most promising candidates of solutions to enhance the resolution. In order to print a dense layout, MPL utilizes $k$ masks ( $k=2$ for double patterning (DPL) and $k=3$ for triple patterning (TPL)) to print features within the minimum feature distance separately. According to [36, 37, 38, 39], DSA is capable of printing regular shape patterns such as holes by grouping them into a guiding template, which relaxes the resolution requirement. They both are compatible to work with either conventional lithography or EUV.

Based on the research progress for the last few years, DSA has been shown to be a perfect fit for manufacturing the layer of contacts/vias because of its capability to generate holes that are uniform shaped and regular positioned. In addition, it is confirmed recently that DSA can be used with MPL as DSA-MPL hybrid Lithography (litho-DSA-litho-DSA...)[40] such that the number of masks is reduced, which leads to a lower cost of the fabrication process. Given the layout of contacts/vias, the first step is grouping them into templates, and the second step is to decompose the resulting templates into different masks as shown in Figs. 4.1 (d) and (e). As the contact/via layout
can be represented by grid graphs introduced in Chapter 3, in this chapter, we study the problem of grouping and coloring grid graphs, especially the diagonal grid graph (DGG).

(a)

(c)

(b)

(d)

(e)

Figure 4.1: (a) The corresponding rectangular grid graph induced from a full rectangular grid for the contact/via layout in (c). (b) The corresponding diagonal grid graph induced from a full diagonal grid for the contact/via layout in (c). (c) A sample contact/via layout. (d) Robust DSA guiding templates. From top to bottom: $1 \times 1,1 \times 2,1 \times 3,2 \times 2$. (e) A DGG is grouped by templates from (d). The edges between templates will inherit from the edges between the vertices of DGG.

As shown in Fig. 4.1(d), predefined DSA templates can be used to print neighboring contact/vias simultaneously, but in the most cases it is still not sufficient to resolve all conflicts in a layout. On the other hand, when working together with MPL, DSA [40] is found useful in reducing the chromatic number k , because it can group vertices into one template, which essentially deletes (contracts) edges in the conflict graph like RGG/DGG, as shown in Fig. 4.1(e). Alternatively, the k-coloring problem can be seen as group-
ing by $1 \times 1$ templates and then coloring with minimum number of colors. So grouping-k-coloring is a broader problem than k-coloring and recently generated attention from [40, 41, 42]. In [40], an ILP formulation and a simple greedy heuristic are proposed, while the former method cannot scale to handle large RGG/DGG and the latter one has no guarantee of solution quality. Kuang et al. [42] worked on the problem of $k=2,3$, but their empirical assumptions are ineligible for RGG/DGG. In fact, RGG/DGG is not always easy to simplify and the edge constraint graphs may not be sufficiently small to find all maximal independent sets, which requires exponential runtime. Besides, the row structure layout studied by Xiao et al. [41] does not apply to RGG/DGG as well. In this chapter, we investigate the grouping-k-coloring problem based on the knowledge built in the previous study. Consequently, given a commonly used template library [4, 42], we prove the NP-completeness when $k=2$ and demonstrate a solution when $k=3$.

The rest of the chapter is organized as follows. In Section 4.2, we define the problem to solve. In Section 4.3 we demonstrate a solution for grouping3 -coloring DGG, and in Section 4.4, we show that the problem of grouping-2-coloring is NP-complete. Finally, we conclude the chapter in Section 4.5.

### 4.2 Problem Definition

We provide a formal definition of our problem in this section. We adopt the definition of diagonal grid graph (DGG) from Chapter 3.

## Definition 4. Grouping-k-coloring an $R G G / D G G$ :

Given an $\operatorname{RGG}\left(V_{R}, E_{R}\right)$ or $\operatorname{DGG}\left(V_{D}, E_{D}\right)$ and a template library $T=\{1 \times$ $1,1 \times 2,1 \times 3,2 \times 2\}$, a valid grouping is a partitioning of $v \in V_{R}\left(V_{D}\right)$ into disjoint subsets $\left\{g_{i}\right\}$ such that any $g_{i}$ belongs to $T$. Subset $g_{i}$ and $g_{j}$ have a conflict edge if there exists any $v_{x} \in g_{i}, v_{y} \in g_{j}$ such that $\left(v_{x}, v_{y}\right) \in E_{R}\left(E_{D}\right)$. The problem is to find a valid grouping such that subsets can be $k$-colored.

### 4.3 Grouping-3-Coloring a Diagonal Graph

After the discussion of direct k-coloring RGG/DGG, in this section, we study the problem of grouping-3-coloring enabled by DSA+MPL technology. Since RGG is already 2-colorable and trivial to solve for the grouping-1-coloring, DGG is sufficient for the discussion. In this chapter, we consider the library of templates: $1 \times 1,1 \times 2,1 \times 3,2 \times 2$, as shown in Fig. 4.1(d), since they are commonly recognized robuster comparing to other larger and irregularshaped templates, such as $1 \times 4$ and "L" shape [4]. Besides, the templates can only be used horizontally and vertically, since the "peanut" shape template along the 45 degree diagonal is not desirable [4]. As a result, the problem becomes how to cover the horizontal and vertical edges in DGG by those templates disjointly such that the resulting conflict graph is 3 -colorable. In fact, we have the following theorem.

Theorem 11. $D G G$ is grouping-3-colorable.
This implies that we can always adopt a grouping strategy and 3-color it in order to decompose DGG. In this section, we show the correctness of this theorem by proposing a solution.

(a)

(b)

(c)

Figure 4.2: (a) Forbidden grouping that produces a $K_{4}$. (b) Forbidden grouping in which the bottom template has three neighbors on its upside. (c) An example of DGG is grouped based on the observations.

Two kinds of grouping patterns are undesirable: (1) four templates produce a conflict graph of $K_{4}$, as shown in Fig. 4.2(a); and (2) one template shares edges with more than two templates on one side (up or down) as shown in Fig. 4.2(b), since it forms a diamond that could potentially make a
graph not 3 -colorable based on previous sections. Following the hard and the soft constraint, it is not difficult to do the grouping row by row as shown in Fig. 4.2(c), but we still need to figure out the grouping that can be 3-colored. We find that a hexagonal matrix of $1 \times 2$ template can be 3 -colored as shown in Fig. 4.3(a), as templates can be periodically assigned red, blue and green in each row. We use such matrix to cover the vertices in DGG as shown in Fig. 4.3(b), and remove the extra templates, and then shrink the wasted $1 \times 2$ templates to $1 \times 1$ ones. By making the remaining templates hold the same colors from Fig. 4.3(a), they can be 3-colored as shown in Fig. 4.3(b). Consequently, we are able to conclude that all DGG can be group-3-colored by this approach and Theorem 11 is proved.


Figure 4.3: (a) A hexagonal matrix of $1 \times 2$ DSA templates. (b) A valid grouping-3-coloring derived from the matrix in (a).

### 4.4 Grouping-2-Coloring a Diagonal Graph

In contrast to grouping-3-coloring, we find that it is intractable to do grouping-2-coloring on DGG. The problem is essentially how to cover the horizontal and vertical edges by those templates disjointly such that all odd cycles are eliminated. Thus, covering an edge can be seen as contracting it and subtracting one from the affected cycle length when considering the 2-colorability of the resulting graph. The problem is solvable if the edge covering does not have to be disjoint and the graph is planar, since it can be formulated as an
edge contraction bipartite problem [42] and a polynomial algorithm based on perfect matching is available. However, edges have to be grouped disjointly as shown in Fig. 4.4(b). On the other hand, in the scenario of DGG, its regularity and sparsity may ease the difficulty. Only the horizontal and vertical edges are the candidates for grouping as contracting an diagonal edge would produce a forbidden peanut shape template. Besides, two neighboring edges cannot be grouped simultaneously except that they are aligned and able to be grouped by a $1 \times 3$ template.

We notice that the relation of vertical and horizontal edges in an odd cycle can be formulated as XOR-SAT expressions. For instance in Fig. 4.4(b), one of $e_{1}$ and $e_{2}$ must be grouped (contracted), since they are the only two options in an triangle. If we use 1 to signify that the edge is grouped and 0 otherwise, $e_{1}$ and $e_{2}$ will have to obey the equation $e_{1} \oplus e_{2}=1$. XOR-SAT is solvable in polynomial time by Gaussian elimination. However, there exists other more complicated constraints in the graph. For instance in Fig. 4.4(b), the diamonds on the right can be grouped in two ways (top and bottom). Generally, Fig. 4.4(b) shows some possible ways to group adjacent edges. In fact, due to the complex grouping relation between edges, we are able to prove the following theorem.

Theorem 12. Grouping-2-coloring $D G G$ is NP-complete.
The proof will be demonstrated in the following sections. In general, we reduce the planar 3-SAT problem to the grouping-3-coloring. The reduction can be done polynomially by using the techniques from Section 3.5.

### 4.4.1 Planar 3-SAT

3-SAT is one of the most famous NP-complete problems, while planar 3-SAT is a special case in which the bipartite graph $B$ of variables and clauses is planar as shown in Fig. 4.4(c). The problem remains NP-complete. Moreover, any 3-SAT clause can be written as a form of 2 SAT-XOR as follows.

$$
\begin{equation*}
(a+b+c) \tag{4.1}
\end{equation*}
$$

is equivalent to

$$
\begin{equation*}
(a+\bar{y})(y \oplus b \oplus z)(\bar{z}+c) \tag{4.2}
\end{equation*}
$$



Figure 4.4: (a) Library of templates. (b) Grouping examples. Red edges are grouped while yellow ones are not. (c) Planar 3-SAT problem. Blue blocks stand for variables, and they are connected to a red block if they appear in the corresponding clause.
where $y, z$ are auxiliary variables. To prove our Theorem 12 , we use edges in DGG as binary variables in planar 3-SAT. An edge is assigned 1 if it is grouped and 0 otherwise. In the rest of this section, we explain how to build an instance of DGG such that it can be mapped to a planar 3-SAT expression. The graph $B$, including edges and Var/Clause blocks as shown in Fig. 4.4(c), will be implemented by DGG components.

### 4.4.2 Edge Implementation

Diamond chains are unsurprisingly utilized as the DGG component to implement edges in $B$. As an example shown in Fig. 4.5(a), all red (vertical) edges have to be grouped simultaneously, which is also true to all yellow edges. Otherwise, a non-2-colorable triangle will be formed by three templates as shown in Fig. 4.5(b). Consequently, two ends of the chain, the leftmost and the rightmost edges, can be used to connect variable and clause blocks such that their corresponding edge values are enforced to be the same as shown in Fig. 4.5(c). Note that this structure can extend and make turns anytime if necessary.


Figure 4.5: (a) Diamond chain is used as an edge. All yellow/red edges have to be the same value. (b) The grouping is not 2-colorable because the three red templates form a triangle. (c) The diamond chain in (a) can connect blocks.


Figure 4.6: (a) The structure for variable block. We have $e_{1}=e_{5}=e_{6}=e_{3}=$ $e_{7}=e_{8}$. (b) The structure of negating a variable. We have $e_{1}=e_{2}=\overline{e_{3}}=\overline{e_{4}}$.

### 4.4.3 Variable Block Implementation

Because diamond chains can only connect two variables, we need a specific structure for variable blocks such that multiple edges can be connected to have the same value. As shown in Fig. 4.6(a), since the odd cycle 1 in green has length 11 and its diagonal edges cannot be used to group, the only horizontal edge $e_{1}$ and the only vertical edge $e_{2}$ have to be different in order to make the cycle length even. The same rule applies to $e_{3}$ and $e_{4}$ due to
the odd cycle 2. Since $e_{1}$ and $e_{4}$ cannot be grouped together, otherwise a triangle will be produced, we have $e_{1}=e_{3}$. If we choose $e_{1}=1$ shown as red and $e_{2}=0$ shown as yellow, then we must pick all the red edges to group. Thus, it implies that $e_{1}=e_{5}=e_{6}=e_{3}=e_{7}=e_{8}$. Based on this structure, we are able to connect four edges $\left(e_{5}, e_{6}, e_{7}\right.$ and $\left.e_{8}\right)$, and enforce them with the same value. Figure 4.6 (b) utilizes the similar structure to have negated variables, where $e_{1}=e_{2}=\overline{e_{3}}=\overline{e_{4}}$. To achieve variable blocks with any degree, we can build larger blocks by connecting multiple structures. An example of variable block with degree 8 is shown in Fig. 4.7. To sum up, we can implement variable blocks for any planar 3-SAT expression with those DGG structures.


Figure 4.7: A variable block that has 8 pins consists of 5 structures in Fig. 4.6(a) rotated by $45^{\circ}$.

### 4.4.4 Clause Block Implementation

In order to implement the clause block of the planar 3-SAT, we utilize its equivalent 2SAT-XOR expression. The DGG structure is shown in Fig. 4.8(a). The green cycle in the middle has even length, and we can force $e_{4}=1$ by the structure cycled by gray dashes. The number of grouped edges in the cycle

(a)




(b)

Figure 4.8: (a) The structure for clause block, where we have $e_{y} \oplus e_{2} \oplus e_{z}$. (b) For $e_{3}$ and $e_{y}$, all 4 possible value assignments are shown. Only when $e_{3}=0$ and $e_{y}=1$, there is no valid grouping-2-coloring due to the lack of 4 -node template. So we have $e_{3}+\overline{e_{y}}$.
must be even to avoid any odd cycle and diagonal edges cannot be grouped, so the number of ones in $e_{y}, e_{2}$ and $e_{z}$ must be odd, which is equivalent to $e_{y} \oplus e_{2} \oplus e_{z}$. Besides, based on Fig. 4.8(b), when $e_{3}=0$ and $e_{y}=1$, there is no valid grouping-2-coloring, but other cases can produce a valid solution. Therefore, we have $e_{3}+\overline{e_{y}}$. Similarly, we can have $e_{1}+\overline{e_{z}}$. All the constraints must be satisfied. Thus, by this DGG component, we could have 2SAT-XOR expression: $\left(e_{1}+\overline{e_{y}}\right)\left(e_{y} \oplus e_{2} \oplus e_{z}\right)\left(\overline{e_{z}}+e_{3}\right)$. Here $e_{1}, e_{2}$ and $e_{3}$ can further connect to edges implemented by diamond chains.

### 4.4.5 NP-Completeness

As a result, once we are given an instance of a planar 3-SAT problem, we can construct a DGG such that it is grouping-2-colorable if and only if all planar 3-SAT clauses can be satisfied. Additionally, the size of the resulting DGG has the polynomial size as explained in Section 3.5. Since planar 3-SAT is NP-complete, grouping-2-coloring DGG is also NP-complete. This completes
the proof of Theorem 12.

Table 4.1: A complete classification of group-k-coloring problems

|  | g-2-colorability | g-3-colorability |
| :---: | :---: | :---: |
| RGG | YES | YES |
| DGG | NP-complete | YES |

### 4.5 Conclusion

Our results are presented in Table 4.1 as a complete classification of the complexity for group-k-coloring problems on grid graphs. All necessary proofs are provided in previous sections. By our study, designers are able to understand the properties of the group-coloring problem and devise the algorithms to place contact/vias such that they can be manufactured by DSA lithography.

## CHAPTER 5

# DENSITY DRIVEN PLACEMENT OF SUB-DSA RESOLUTION ASSISTANT FEATURES (SDRAFS) FOR DIRECTED SELF-ASSEMBLY LITHOGRAPHY 

### 5.1 Introduction

In sub-22 nm technology node, conventional lithography (193i) has reached its limit due to continuously shrinking feature size. Alternatively, electron beam lithography (E-beam) [15, 43], extreme ultraviolet lithography (EUVL) [44] and directed self-assembly (DSA) [16, 4, 45] have been proposed as next generation lithography techniques and been intensively researched for years. However, E-beam suffers from its low throughput problem and EUVL keeps delayed because of mirror defects and low source power. DSA has been proven as a promising candidate to generate periodic patterns in a large area. Therefore, it is a perfect fit to print contact/via layers, which are usually the densest and hardest to print. Recent studies also show that DSA can also work compatibly with multiple patterning, which can achieve even smaller feature size. In DSA process for random logics, guiding templates are used to confine the block copolymers such that small clusters of cylinders can be formed inside the template. A matrix of templates with two cylinders (holes) are shown as an example in Fig. 5.1.

However, DSA also suffers from possible defects. Previous works [45, 46] have demonstrated that uneven block copolymer fill level of the templates may cause missing hole defects. Since block copolymers are spin-coated uniformly over the substrate, a template within the relatively lower density region will be overfilled and consequently produces no hole inside the template. Even though neutral substrate is more robust to this defect, it suffers poor (LCDU) due to the uneven fill level [47]. Thus, it becomes critical to uniform the local density of the templates in the layout.

To mitigate this problem, Yi et al. [45] proposed sub-DSA resolution assistant features (SDRAFs). SDRAF is a template with smaller dimensions such
that no transferable pattern will be printed on the wafer, but it can preform as a reservoir to divert redundant co-polymers away from the overfilled templates [45]. Therefore, SDRAFs can be placed to the area with lower density in order to even out the density. As an example shown in Fig. 5.2 (a), holes are missing in the templates marked by red squares because they are in low density region, but in Fig. 5.2 (b) holes are formed by adding SDRAFs in this area (blue) to consume the copolymers.

SDRAFs could be difficult to print [45]. Consequently, in order to minimize the process variations, it is undesirable to insert unnecessarily many SDRAFs. Besides, enough space needs to be reserved for other features such as sub-resolution assistant features (SRAFs). In this chapter, we propose an algorithm to place SDRAFs into the layout such that the density can be as even as possible and the number of SDRAFs are minimized at the same time.

The rest of chapter will be organized as following. Section 5.2 will introduce the background of SDRAFs and give the problem definition. Section 5.3 will demonstrate the proposed placement algorithm in detail. Section 5.4 will show the experimental results and we will conclude our chapter in Section 5.5.


Figure 5.1: Two-hole templates.

### 5.2 Preliminary

In this section, we will demonstrate the background knowledge of SDRAFs and introduce the definition of the density evening problem that will be addressed in this chapter.


Figure 5.2: (a) Two templates in red squares have missing hole defect. (b) With SDRAFs in blue area, holes are formed back inside the two templates in red square.

For a random logic circuit, the contact/vias may not be uniformly distributed as an example in Fig. 5.3 (a). Thus, after grouping them into guiding template, the local density of templates may vary dramatically. As results, the fill levels of the templates are also significantly uneven. Recent work [45] has shown that holes may not be formed inside the templates that are overfilled or underfilled. However, the block copolymer film thickness must be adjusted to ensure that sufficient copolymers are deposited to the densest region. Since the thickness is uniform, block copolymers will inevitably increase the fill level of templates in the less dense region and cause DSA holes disappeared [45]. Although templates with neutral substrates have been shown to be more robust to this defect due to template overfill, Doise et al. [48] have reported that varying fill levels in templates cause poor LCDU, because the DSA holes in templates with more copolymers have higher aspect ratio than those with less copolymers. The scenario of overfilling is shown in Fig. 5.2 (a). Because of the lower density around, the templates in red squares are overfilled by BCPs and have no hole formed inside.

To remedy the uneven density issue, Yi et al. [45] proposed sub-DSA resolution assistant features (SDRAFs) to balance the template density. SDRAFs are small openings in the template layer such that they can act as sinks to share extra polymers, which can effectively prevent the overfilling problem in the lower density region [47]. It is essential that no transferable pattern
will be actually printed by SDRAF and thus the dimensions of SDRAFs need to be controlled precisely. Thus, it is not desirable to add unnecessarily SDRAFs and increase the process variations, which might result in unexpected patterns. Consequently we aim at using the minimum number of SDRAFs to even out the density. For instance as shown in Fig. 5.3, a layout of vias in (a) could be fully filled by SDRAFs in order to uniform the density as presented in (b), but many redundant SDRAFs are inserted. Indeed, Fig. 5.3 (c) shows that our algorithm can place SDRAFs to even out the density in a more efficient way and the number of SDRAFs can be reduced dramatically.


Figure 5.3: Blue rectangles are templates, and green ones are SDRAFs. (a) A sample layout of vias. (b) SDRAFs fully fill out the layout. (c) The optimal placement uses much less SDRAFs to even out the density.

The density of templates is defined on a local circular region based on the diffusion nature of polymers. Formally, we first define the interactive region (IR) of a template $t_{i}$ as the following.

Interactive Region $I R\left(t_{i}\right)$ : The interactive region $I R$ of a template $t_{i}$ is the circular area centered at $t_{i}$ with radius $R\left(t_{i}\right)$ so that block copolymers inside could possibly diffuse into $t_{i}$.

An example IR is shown in Fig. 5.3 (a). Then, we define the local density of a template $t_{i}$ as the following.

Density $D\left(t_{i}\right)$ : The density $D$ of a template $t_{i}$ is the total number of guiding templates and SDRAFs in its IR.

Note that $R\left(t_{i}\right)$ could be different but we assume that $R\left(t_{i}\right)$ is the same for all $t_{i}$ for simplicity to illustrate our algorithm. Thus, the problem left is how to place SDRAFs in the layout such that the density of each template
can be as even as possible and the number of SDRAFs used is minimized. If we use variance to measure the evenness of the density, we can define this problem as the following.

SDRAF Placement Problem: Given a layout, use the minimum number of SDRAFs to minimize the variance of densities $\operatorname{Var}\left(D\left(t_{i}\right)\right)$ for all $i$.

### 5.3 Algorithm

In this section, we demonstrate the algorithm to solve the SDRAF placement problem. We adopt an iterative approach, which is illustrated by a flowchart shown in Fig. 5.4 (a). First, we preprocess the input layout to find the densest region and obtain the maximum density. Second, we determine all available locations to insert an SDRAF. Third, we calculate priority and choose the area with the highest one to place an SDRAF. Then, we update the priority and lock the densest region. Finally, we continue to conduct those steps iteratively until no location available to add an SDRAF that can improve the evenness of the density. The details will be explained in the following sections.


Figure 5.4: (a) A flowchart of proposed algorithm. (b) All possible SDRAF locations (yellow) inside one interactive region. The red arrow indicates that this location does not violate the spacing rule with the template (blue).

### 5.3.1 Preprocess

In order to reduce the variance of density through out the layout, we need to add SDRAFs to the lower density region to match up with the highest density region, since there is no way to remove any template from the layout. Thus, we first need to find the hight density as $D_{M A X}=\max _{i}\left(D\left(t_{i}\right)\right)$ for all $i$.


Figure 5.5: $D_{M A X}=7$ is assumed. (a) The value of demand of each IR is shown in the black circle. (b) The demands are sum up in intersected area to be a part of the priority.

Next, we place a grid of candidate locations for SDRAFs as shown in Fig. 5.4 (b). The length of each line segment in the grid is $d_{\text {min }}^{S S}$ which is the minimum distance between two SDRAFs. Note that the grid points that violate the minimum spacing rule with DSA templates are removed from the candidate set.

### 5.3.2 Priority

In each iteration of our algorithm, we find one candidate location to place a SDRAF, which is determined by priority $P_{x, y}$, where $x$ and $y$ are the indices of grid points. The priority is calculated by two factors: demand and flexibility.

Demand of an IR, denoted by $M\left(t_{i}\right)$, represents how large the gap is between the current density to $D_{M A X}$ and it is also the number of SDRAFs
needed to add in this IR. Formally, $M\left(t_{i}\right)=D_{M A X}-D\left(t_{i}\right)$. SDRAFs essentially are added to reduce the demand. By intuition, the area with the higher demand should have higher priority to insert an SDRAF. Areas with zero demand including the area outside any IR or inside the IR with the maximum density should not have any SDRAF added, which can avoid lots of unnecessary SDRAF used. To further reduce the number of SDRAFs, the area that are intersected by multiple IRs should have higher priority, since adding SDRAFs in those areas can effectively decrease the demand of multiple IRs instead of just one. As an example shown in Fig. 5.5 (a), the values of demands are shown for all IRs. While, in Fig. 5.5 (b), if an area is intersected by more than one IRs, we sum up all demands from those IRs and this sum would be a part of the priority. In this example, we have the highest sum as 14 in this layout if $D_{M A X}=7$.


Figure 5.6: (a) SDRAFs are selected only depending on demand. (b) Assuming $D_{M A X}=7$, IRs of blue dashes are locked up. There is no location available to any new SDRAF in the IR of red dashes though two SDRAFs are needed.

However, demand cannot cover all factors of the priority by itself. So, we introduce flexibility $F\left(t_{i}\right)$ to keep track of the number of available locations left to place SDRAFs. As an example in Fig. 5.6 (a), if only the demands are considered, SDRAFs (green) will be added because of the high demands. Consequently, in Fig. 5.6 (b), the IRs (blue dashes) are locked up because they reach the maximum density, and all possible locations are removed
in those IRs. However, the IR with red dashes loses all its places to add any SDRAF, so it will not reach the maximum density any more. Thus, it preferable to assign high priority to IRs with a small number of available locations and we use flexibility as a trade-off factor to the demand when calculating the priority.

As a result, we compute the priority by $P_{x, y}=\omega_{M} \times \Sigma M\left(t_{i}\right)-\omega_{F} \times$ $\max \left(F\left(t_{i}\right)\right)$ for all $t_{i}$ that their IR contain the location $(x, y)$, where $\omega_{M}$ and $\omega_{F}$ are two positive weights. We keep updating the priority, and pick the location with highest priority to place an SDRAF, and lock up any IR with the highest density at each iteration. The algorithm will be terminated when no location is available to place an SDRAF and decrease the demand.


Figure 5.7: A placement result of a sample layout with 1000 DSA templates by algorithm proposed.

### 5.4 Experimental Result

In this section, we will show the experimental result of our proposed algorithm. The algorithm described in the previous sections is implemented by C++ and it is optimized by using KD-tree structure to store the DSA template locations to speed-up the geometric query. We run the program on a Linux workstation with four cores of 3.2 GHz and 24 GB memory. The results are shown in Table 5.1. The first column shows the number of templates in the layout. The second to the forth column present the initial variance of the density, the result variance of the density and the percentage of this variance reduced respectively. The fifth to the seventh column present the
number of SDRAFs inserted if the whole layout are filled by SDRAFs, the number of SDRAFs inserted by our algorithm and the percentage of this number reduced respectively. The last column reports the running time. Our benchmarks consist of five layouts with 100, 400, 700, 1000 and 10000 templates respectively. We run our placement algorithm on each layout for 10 times and pick the best result in each case. We use 96 nm for the minimum distance between two SDRAFs and between an SDRAF and a DSA template. As shown in the table, we use variance to measure the evenness of the density, and we can significantly reduce the variance by more then $85 \%$ by placing SDRAFs by our algorithm. Our algorithm can also reduce the number of SDRAFs by around $50 \%$ comparing to filling the whole layout with SDRAFs. An example layout after the placement is shown in Fig. 5.7.

### 5.5 Conclusion

The DSA process is one of the most promising lithography techniques to print contact/via layers, but it has defect issues caused by uneven density of the templates. Sub-DSA resolution assistant features (SDRAFs) can be utilized to mitigate the problem by evening out the density. We propose an SDRAF placement algorithm that can make the density as even as possible and minimize the number of SDRAFs. The experimental results indicate that our method is very effective and efficient. This SDRAF placement scheme gives a path to integrating SDRAFs into random logic contact/via layouts and to mitigating the effects of template overfill due to density non-uniformity. This investigation of SDRAF placement sheds light on the DSA density variation problem and suggests future paths to mass deployment of DSA.
Table 5.1: Experimental results

| \# DSA | Init. <br> Var. | Res. <br> Var. | \% Drop <br> of Var. | \# SDRAF <br> full layout | \# SDRAF <br> algorithm | \% Drop of <br> \# SDRAF | Run <br> time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 100 | 7.25 | 0.52 | $92.85 \%$ | 2546 | 149 | $94.15 \%$ | 0.26 s |
| 400 | 24.82 | 3.08 | $87.56 \%$ | 2793 | 508 | $81.81 \%$ | 2.66 s |
| 700 | 25.49 | 3.00 | $88.23 \%$ | 4317 | 866 | $79.94 \%$ | 7.74 s |
| 1000 | 39.98 | 3.19 | $92.02 \%$ | 6291 | 1462 | $76.76 \%$ | 18.68 s |
| 10000 | 156.53 | 12.45 | $92.05 \%$ | 34497 | 19811 | $42.58 \%$ | 36 m 21 s |

## CHAPTER 6

## DENSITY BALANCING AWARE MASK ASSIGNMENT IN DSA-DPL HYBRID LITHOGRAPHY FOR CONTACT LAYERS

### 6.1 Introduction

In sub-10 nm technology node, conventional lithography (193i) has reached its limit because of optical diffraction. Other options are explored including extreme ultraviolet lithography (EUV) and electron beam lithography (E-beam) [15]. However, they suffers problems of defects and low throughput respectively. Meanwhile, to print contact/vias that usually has the most dense layers in a circuit, directed self-assembly (DSA) technology [4, 16] shows its great potential, since it can generate uniformly shaped and distributed cylinders [36, 37, 38, 39]. DSA first groups the contacts into guiding templates and contact holes are formed inside a template by copolymers to achieve better resolution. However, is it hardly sufficient for a single lithoDSA process to resolve all conflicts (features violate the minimum spacing rule) in the layout. Therefore, DSA usually works with multiple patterning, for instance double patterning (litho-DSA-litho-DSA) in order to further improve the resolution.

In order to form desired patterns inside the DSA template, the number of the block co-polymers (BCPs) must be controlled precisely such that no template is overfilled or underfilled [45]. On the other hand, the BCPs are spin-coated uniformly over the substrate. As a result, some isolated template is likely to be overfilled since all nearby BCPs diffuse toward it, and thus undesirable patterns are generated inside the template. Consequently, the density of the template distribution becomes critical.

For DSA without MPL, one of the options to mitigate the condition is to insert sub-DSA resolution assistant features (SDRAFs) specially designed such that no etch-transferable pattern is formed inside but they can perform like reservoirs to share the BCPs [45] as shown in Fig. 6.1(b). Though this
could potentially make the density as even as possible, if too many SDRAFs are added, it most likely generates a much denser layout, and has to run the risk of undesirable hole on the wafer created by faulty SDRAFs. Additionally, it leaves little space for sub-resolution assist features (SRAFs) used to help for printing and increases the complexity for optical proximity correction (OPC) optimization [49]. Because of those reasons, when considering DSA-MPL, it is important to assign the templates into different masks in a balanced way such that the template density could be relatively even distributed and needs a minimum number of SDRAFs added.

Given the layout of contacts/vias, the first step is grouping them into templates such that the minimum number of masks are required, and the second step is to decompose the resulting templates into different masks. The former problem of grouping contacts is shown hard to solved optimally [40]. In this chapter, we focus on optimizing the latter step of mask assignment for DSA with DPL (DSA-DPL) such that the optimal density of the templates is achieved and the minimum number of SDRAFs is required. To our knowledge, it is the first work to handle the optimization problem of mask assignment aware of the DSA template density issue. Our contribution is summarized as the following.

1. A novel optimization problem of mask assignment is formulated for DSA template density balancing purpose.
2. An integer linear programming (ILP) formulation is presented to solve for the mask assignment optimally.

The rest of the chapter is organized as follows. Section 6.2 introduces the background of the technology, and then discusses the optimization objective, and finally provide the definition of the problem. Section 6.3 presents an ILP approach to solve the problem. Finally, the experimental results are discussed in Section 6.4. We conclude this chapter in Section 6.5.

### 6.2 Background and Problem Formulation

In this section, in order to illustrate the formulation of the problem, we first introduce the background knowledge of DSA-MPL hybrid lithography


Figure 6.1: (a) Relatively low density causes no hole formed in the templates (red). (b) After adding SDRAFs (blue), the templates are not overfilled.
as well as its density issue. Then the problem objective is analyzed. Finally we define our optimization problem.

### 6.2.1 DSA-MPL Hybrid Lithography

Left unconfined, block copolymers are known to self-assemble into arrays of hexagonally packed cylinders. However, this long-range periodicity is incongruous with the needs of random logic, where contacts and vias are scattered aperiodically in layouts. For this reason, topographical guiding templates are used to confine the block copolymer, causing it to instead form small clusters of cylinders according to the template shape and at a pitch dictated by the natural pitch of the block copolymer [50]. Because the block copolymer can self-assemble into multiple holes per guiding template, block copolymers can effectively enable pitch multiplication. Vias that would conventionally have to be printed on separate masks can instead be printed together on one mask in a multi-hole template and later resolved by the block copolymer into separate holes. In this way, the use of block copolymer directed self-assembly to print circuits can allow contacts and vias to be grouped in such a way that fewer masks can be used relative to conventional optical lithography.

When decomposing a via layout for DSA, it is important to consider the
range of templates that can be used. Previous work has posted that for DUV, template fidelity is insufficiently high to allow for good DSA hole placement accuracy in templates larger than doublets or triplets. Indeed, in Karageorgos et al. [51], the range of templates available for layout decomposition for 193i MPL is specifically linked to the layout grid dimensions for the technology node. The template alphabet, such as singlet, doublet and triplet can be used to group vias into templates. The grouping of the vias may not be sufficient to resolve all minimum distance rule violations, then MPL is adopted and the templates are assigned to different masks.

### 6.2.2 Template Density and SDRAFs

A notable feature of these decomposed layouts is their inherent density variation that comes as a result of the uneven distribution of vias. Recent work [45] has demonstrated that this density variation can prove problematic for the DSA process, as uneven density can cause the fill level in templates to vary dramatically. Because the BCP film thickness across a given collection of templates is uniform and the amount of BCPs deposited must be optimized for the most dense region, spare BCPs around a template in less dense area would diffuse toward the template and increase its fill level. The uneven fill level has been shown in PMMA-affinitive templates to cause missing DSA holes in areas where the templates have overfilled due to relatively low template density [52]. Although templates with neutral substrates have been shown to be more robust to missing holes due to template overfill, Doise et al. [48] have reported poor LCDU control post-etching in templates with varying fill level, as the DSA holes in templates with more polymer have higher aspect ratio than those with less polymer. The scenario of overfilling is shown in Fig. 6.1 (a). Because of the lower density around, the templates in red squares are overfilled by BCPs and have no hole formed inside. For convenience, we define impact region, radius and its density as following.

Impact Region $I R\left(t_{i}\right)$ : The impact region $I R$ of a template $t_{i}$ is the circular area that the BCPs inside could possibly diffuse to $t_{i}$.

Radius $R\left(t_{i}\right)$ : The radius $R$ of a template $t_{i}$ is the radius of its $I R$.
Density $D_{m}\left(t_{i}\right)$ : The density $D_{m}$ of a template $t_{i}$ is the total number of templates on mask $m$ in its $I R$. For $D P L, m=0$ or 1 .

The density is measured for each template to evaluate the chance of overfilling, and it is defined on an circular region because of the diffusion nature of BCPs.

In MPL, the templates are partitioned into different masks, which possibly generates uneven layouts and has some template overfilled. As shown in Fig. 6.2(a), a set of templates in layout needs to be printed by double patterning (two masks). Though the partitioning shown in Fig. 6.2(b) resolves all minimum distance violations, this unbalanced partitioning enlarges the template density variation, for instance, the densities in the circle area of the top and bottom figures are much different. Thus, the template at the center of the circle in the lower figure will be possibly overfilled. On the other hand, Fig. 6.2(c) shows another partitioning result that both masks have relatively even density. It lowers the possibilities of overfilling dramatically and thus is desirable.

As a complementary to balanced partitioning, another way to solve the uneven template fill levels proposed by Yi et al. [45] calls for the use of sub-DSA resolution assist features (SDRAFs) to balance template density illustrated in Fig. 6.1(b). These SDRAFs surrounded by blue lines are placed in regions of low template density, and act as polymer sinks to divert copolymer away from overfilled templates. They do not create actual holes on the wafer during the DSA process. In DSA-MPL, after the mask assignment is done, the uniform BCP film thickness is optimized to be compatible with the region with the largest density of all masks. Then the rest of areas have lower density and templates in those areas are risky to be overfilled. To remedy the issue, SDRAFs are inserted to increase the density of the rest of the layout to match the most dense region. However, the size and shape of SDRAFs need to be controlled precisely such that they do not themselves produce any etch-transferrable features (holes).

### 6.2.3 Problem Formulation

In this chapter, we mainly study the problem of DSA with double patterning (DSA-DPL). Given a layout of vias, after grouping the vias into templates, they are 2 -colored and assigned to mask 0 and mask 1 . The template distribution on each mask may be unbalanced. Thus, SDRAFs are inserted to


Figure 6.2: Layout (a) is colored into 2 masks. The partitions in (b) have larger density variation than the partitions in (c).
even the template density, but they are potentially to be faulty and produce etch-transferrable features due to fabrication variations. Consequently, in order to reduce the uncertainty caused by SDRAFs, it is desirable to minimize the number of the SDRAFs added. The problem becomes how to assign templates to the masks such that the density is optimized in a way that the number of the SDRAFs needed is minimized. If we use $N_{S D R A F}$ to present the number of SDRAFs, the problem is to minimize

$$
N_{S D R A F}=\frac{D_{M A X}(0)}{A(I R)} \times A(0)-N_{t}(0)+\frac{D_{M A X}(1)}{A(I R)} \times A(1)-N_{t}(1)
$$

$D_{M A X}(i)$ is the maximum density of mask $i . A(I R)$ is the area of an IR, which has neglectable variance between different templates, so we treat it as a constant in this work. $A(i)$ is the total area occupied by IRs on mask $i$. $N_{t}(i)$ is the number of original templates assigned to mask $i$. Note that the summation of $N_{t}(0)$ and $N_{t}(1)$ is the total number of templates, so they can be treated as constant as well. As a result, the problem becomes to minimize

$$
D_{M A X}(0) \times A(0)+D_{M A X}(1) \times A(1)
$$

Table 6.1: Terminologies

| $N_{t}(i)$ | The number of templates in mask $i$. |
| :---: | :--- |
| $N_{M}$ | The number of masks. |
| $N_{S D R A F}$ | The number of SDRAFs needed. |
| $T$ | The set of all templates in the layout. |
| $t_{i}$ | The $i^{\text {th }}$ template for $1 \leq i \leq N$. |
| $I R\left(t_{i}\right)$ | The Impact Region of $t_{i}$. |
| $R\left(t_{i}\right)$ | The Radius of $I R\left(t_{i}\right)$. |
| $D\left(t_{i}, m\right)$ | The number of templates in $I R\left(t_{i}\right)$ on <br> Mask $m$. |
| $D_{M A X}(i)$ | The maximum density of templates on <br> mask $i$. |
| $M\left(t_{i}\right)$ | The binary indicator for the mask as- <br> signment of $t_{i}$. It is 0 if $t_{i}$ is assigned to <br> Mask 0. It is 1 if it is assigned to Mask <br> 1. |
| $N e\left(t_{i}\right)$ | The set of templates $t_{j}$ such that $t_{i}$ and <br> $t_{j}$ are neighbors (in each other's IR). |

Additionally, $A(i)$ is very insensitive to different mask assignments because (1) two conflict templates will produce almost the same IR on each mask because they are very close to each other, and (2) the radius of IR is quite large comparing to the distance between templates. Therefore, as shown in Fig. 6.3, the total union area of IRs in mask 0 is almost the same to the total area of the mask 1 in practice. So, we can neglect $A(i)$. Thus, the problem is to choose the color assignment such that the following is minimized.

$$
D_{M A X}(0)+D_{M A X}(1)
$$

### 6.3 ILP Approach

In this section, we formulate our problem into an integer linear programming (ILP) and solve it optimally by an ILP solver. Some notations are shown in Table 6.1.

Given a layout of templates, a conflict graph $G\left(V_{g}, E_{g}\right)$ can be constructed such that each vertex refers to a template $t_{i} \in V_{g}$ and an edge $\left(t_{i}, t_{j}\right) \in E_{g}$ if $t_{i}$ and $t_{j}$ violate the minimum spacing rule. In this chapter, we assume


Figure 6.3: (a) IRs of all templates are shown in red. (b) IRs of templates assigned to mask 0 is shown in green. (c) IRs of templates assigned to mask 1 is shown in blue.
that the given layout is 2-decomposable, namely $G$ is 2-colorable, since we focus on the stage of mask assignment. As shown in Table 6.1, we define the neighborhood of $t_{i}$ as follows.

Neighbor: $t_{i}$ and $t_{j}$ are neighbors if they are in each other's $I R$.

Note that we assume the radius is a constant, so $t_{i}$ and $t_{j}$ are neighbors of each other simultaneously. In other words, $t_{j} \in N e\left(t_{i}\right)$ always implies $t_{i} \in N e\left(t_{j}\right)$.

First, in order to enforce the conflicts, $M\left(t_{i}\right) \oplus M\left(t_{j}\right)=1$ if $t_{i}$ and $t_{j}$ are conflicted. The exclusive or operation enforces that $M\left(t_{i}\right)$ and $M\left(t_{j}\right)$ are assigned to different masks. Because $M\left(t_{i}\right)$ is a binary, this can be interpreted into the following.

## Conflict constraints:

$$
M\left(t_{i}\right)+M\left(t_{j}\right)=1 \forall\left(t_{i}, t_{j}\right) \in E_{g}
$$

Second, we use positive integer variables $D\left(t_{i}, m\right)$ to represent the number of templates in the IR of $t_{i}$ on Mask $m$. Note that if $t_{i}$ is assigned to mask 1 , then $D\left(t_{i}, 0\right)$ is set to 0 . They are calculated as the following.

Density constraints:

$$
\left\{\begin{array}{l}
D\left(t_{i}, 1\right)=M\left(t_{i}\right)+\sum_{t_{j} \in N e\left(t_{i}\right)} M\left(t_{j}\right) \\
D\left(t_{i}, 0\right)=1-M\left(t_{i}\right)+\sum_{t_{j} \in N e\left(t_{i}\right)} 1-M\left(t_{j}\right)
\end{array}\right.
$$

Third, we need to get the maximum density on each mask. We present it by $D_{M A X}(i)$. This is enforced by the following.

Maximum constraints:

$$
\begin{aligned}
& D\left(t_{i}, 1\right) \leq D_{M A X}(1) \forall t_{i} \in T \\
& D\left(t_{i}, 0\right) \leq D_{M A X}(0) \forall t_{i} \in T
\end{aligned}
$$

Combining all those constraints, the final ILP is as follows.
$\operatorname{minimize} D_{M A X}(0)+D_{M A X}(1)$
subject to:

$$
\begin{array}{ll}
M\left(t_{i}\right)+M\left(t_{j}\right)=1 & \forall\left(t_{i}, t_{j}\right) \in E_{g} \\
D\left(t_{i}, 1\right)=M\left(t_{i}\right)+\sum_{t_{j} \in \operatorname{Ne}\left(t_{i}\right)} M\left(t_{j}\right) & \forall t_{i} \in T \\
D\left(t_{i}, 0\right)=1-M\left(t_{i}\right)+\sum_{t_{j} \in N e\left(t_{i}\right)} 1-M\left(t_{j}\right) & \forall t_{i} \in T \\
D\left(t_{i}, 1\right) \leq D_{M A X}(1) & \forall t_{i} \in T \\
D\left(t_{i}, 0\right) \leq D_{M A X}(0) & \forall t_{i} \in T \\
M\left(t_{i}\right) \in\{0,1\} & \forall t_{i} \in T
\end{array}
$$

### 6.4 Experimental Results

We solve our ILP by GUROBI [53] solver on a Linux workstation with 3.2 GHz CPU and 7.5 GB memory. We build our benchmarks by randomly picking a portion from an industrial metal 0 via layout by applying a various size window. We use $1 \mu m$ as the radius of the impact regions. We compare the results from ILP to random color assignment and show them in Table 6.2. Based on the results, the ILP can be solved for the large testcase with more than 31000 templates in hundreds of seconds and it can affectively reduce the number of SDRAFs needed more than $10 \%$ compared to randomly assignment.

### 6.5 Concluding Remarks

For the 10 nm technology node and beyond, DSA-MPL technology has presented a unique opportunity of mask design optimization for improving the manufacturability. This chapter is the first work studying the density aware mask assignment. For the first time, we define the density of a template by using a circular impact region based on the property of block co-polymers diffusion. Then we formulate the objective to minimizing the maximum densities. The problem can be optimally and efficiently solved by integer linear programming (ILP).
Table 6.2: Comparisons between random assignment and ILP

| Test | \# templates | Random Assignment |  |  | ILP |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $D_{M A X}(0)+D_{M A X}(1)$ | $N_{S D R A F}$ | $D_{M A X}(0)+D_{M A X}(1)$ | $N_{S D R A F}$ | CPU | \% Reduction |
| 1 | 988 | 114 | 580 | 103 | 416 | 0.67 | 28 |
| 2 | 4067 | 119 | 3146 | 108 | 2479 | 3.36 | 21 |
| 3 | 11374 | 224 | 13567 | 204 | 11340 | 19.69 | 16 |
| 4 | 22198 | 308 | 12139 | 295 | 10689 | 128.72 | 12 |
| 5 | 31944 | 426 | 15548 | 388 | 11756 | 444.39 | 24 |

## REFERENCES

[1] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. IEEE Press, 2008, pp. 465-472.
[2] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on. IEEE, 2011, pp. 1-8.
[3] H. Zhang, Y. Du, M. D. Wong, Y. Deng, and P. Mangat, "Layout small-angle rotation and shift for euv defect mitigation," in ComputerAided Design (ICCAD), 2012 IEEE/ACM International Conference on. IEEE, 2012, pp. 43-49.
[4] Y. Du, D. Guo, M. D. Wong, H. Yi, H.-S. P. Wong, H. Zhang, and Q. Ma, "Block copolymer directed self-assembly (DSA) aware contact layer optimization for 10 nm 1d standard cell library," in ComputerAided Design (ICCAD), 2013 IEEE/ACM International Conference on. IEEE, 2013, pp. 186-193.
[5] A. Fujimur, "Beyond light: The growing importance of e-beam," in Proc. Int. Conf. on Computer Aided Design, 2009.
[6] K. Yoshida, T. Mitsuhashi, S. Matsushita, L. L. Chau, T. D. T. Nguyen, D. MacMillen, and A. Fujimura, "Stencil design and method for improving character density for cell projection charged particle beam lithography," Sep. 2 2009, US Patent App. 12/552,373.
[7] T. Fujino, Y. Kajiya, and M. Yoshikawa, "Character-build standard-cell layout technique for high-throughput character-projection EB lithography," in Photomask and Next Generation Lithography Mask Technology XII. International Society for Optics and Photonics, 2005, pp. 160-167.
[8] M. Sugihara, K. Nakamura, Y. Matsunaga, and K. Murakami, "CP mask optimization for enhancing the throughput of MCC systems," in 26th Annual BACUS Symposium on Photomask Technology. International Society for Optics and Photonics, 2006, p. 63494B.
[9] M. Sugihara, T. Takata, K. Nakamura, R. Inanami, H. Hayashi, K. Kishimoto, T. Hasebe, Y. Kawano, Y. Matsunaga, K. Murakami et al., "Technology mapping technique for throughput enhancement of character projection equipment," in SPIE 31st International Symposium on Advanced Lithography. International Society for Optics and Photonics, 2006, p. 61510Z.
[10] K. Yuan, B. Yu, and D. Z. Pan, "E-beam lithography stencil planning and optimization with overlapped characters," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 31, no. 2, pp. 167-179, 2012.
[11] B. Yu, K. Yuan, J.-R. Gao, and D. Z. Pan, "E-blow: e-beam lithography overlapping aware stencil planning for MCC system," in Proceedings of the 50th Annual Design Automation Conference. ACM, 2013, p. 70.
[12] J. Kuang and E. F. Young, "A highly-efficient row-structure stencil planning approach for e-beam lithography with overlapped characters," in Proceedings of the 2014 on International Symposium on Physical Design. ACM, 2014, pp. 109-116.
[13] C. Chu and W.-K. Mak, "Flexible packed stencil design with multiple shaping apertures for e-beam lithography." in $A S P-D A C, 2014$, pp. 137142.
[14] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proceedings of the American Mathematical society, vol. 7, no. 1, pp. 48-50, 1956.
[15] D. Guo, Y. Du, and M. D. Wong, "Polynomial time optimal algorithm for stencil row planning in e-beam lithography," in Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific. IEEE, 2015, pp. 658-664.
[16] Z. Xiao, D. Guo, M. D. Wong, H. Yi, M. C. Tung, and H.-S. P. Wong, "Layout optimization and template pattern verification for directed selfassembly (DSA)," in Proceedings of the 52nd Annual Design Automation Conference. ACM, 2015, p. 199.
[17] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 3, pp. 433-446, 2015.
[18] J.-S. Yang and D. Z. Pan, "Overlay aware interconnect and timing variation modeling for double patterning technology," in Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on. IEEE, 2008, pp. 488-493.
[19] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. D. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in 2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2012, pp. 57-64.
[20] R. Rodrigues and S. Kundu, "Model based double patterning lithography (DPL) and simulated annealing (SA)," in Quality Electronic Design (ISQED), 2011 12th International Symposium on. IEEE, 2011, pp. 1-8.
[21] R. J. Socha, "Method, program product and apparatus for performing decomposition of a pattern for use in a DPT process," July 23 2013, US Patent 8,495,526.
[22] R. Rodrigues and S. Kundu, "A mask double patterning technique using litho simulation by wavelet transform," in Proceedings of the 20th Symposium on Great lakes Symposium on VLSI. ACM, 2010, pp. 103-106.
[23] M. Schäffter, "Drawing graphs on rectangular grids," Discrete Applied Mathematics, vol. 63, no. 1, pp. 75-89, 1995.
[24] M. Holzer and S. Jakobi, "Grid graphs with diagonal edges and the complexity of xmas mazes," in International Conference on Fun with Algorithms. Springer, 2012, pp. 223-234.
[25] L. Dignan, "IBM research builds functional 7 nm processor," 2015.
[26] Y. Du, H. Zhang, M. D. Wong, and K.-Y. Chao, "Hybrid lithography optimization with e-beam and immersion processes for 16 nm 1d gridded design," in 17th Asia and South Pacific Design Automation Conference. IEEE, 2012, pp. 707-712.
[27] M. Smayling, "Gridded design rules: 1-d design enables scaling of CMOS logic," Nanochip Technology Journal, vol. 6, no. 2, pp. 33-37, 2008.
[28] R. T. Greenway, R. Hendel, K. Jeong, A. B. Kahng, J. S. Petersen, Z. Rao, and M. C. Smayling, "Interference assisted lithography for patterning of 1d gridded design," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2009, p. 72712U.
[29] J. Ryckaert, "Scaling beyond 7nm: Design-technology co-optimization at the rescue," in Proceedings of the 2016 on International Symposium on Physical Design. ACM, 2016, p. 89.
[30] C. Cork, J.-C. Madre, and L. Barnes, "Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns," in Photomask and NGL Mask Technology XV. International Society for Optics and Photonics, 2008, p. 702839.
[31] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in Proceedings of the 50th Annual Design Automation Conference. ACM, 2013, p. 69.
[32] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 3, pp. 397-408, 2014.
[33] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in Proceedings of the International Conference on Computer-Aided Design. IEEE Press, 2013, pp. 170-177.
[34] B. Randerath and I. Schiermeyer, "Vertex colouring and forbidden subgraphs-a survey," Graphs and Combinatorics, vol. 20, no. 1, pp. $1-40,2004$.
[35] H. J. Bandelt, V. Chepoi, and M. Laurent, "Embedding into rectilinear spaces," Discrete \& Computational Geometry, vol. 19, no. 4, pp. 595604, 1998.
[36] C. T. Black, R. Ruiz, G. Breyta, J. Y. Cheng, M. E. Colburn, K. W. Guarini, H.-C. Kim, and Y. Zhang, "Polymer self assembly in semiconductor microelectronics," IBM Journal of Research and Development, vol. 51, no. 5, pp. 605-633, 2007.
[37] M. P. Stoykovich, H. Kang, K. C. Daoulas, G. Liu, C.-C. Liu, J. J. de Pablo, M. Müller, and P. F. Nealey, "Directed self-assembly of block copolymers for nanolithography: Fabrication of isolated features and essential integrated circuit geometries," Acs Nano, vol. 1, no. 3, pp. 168-175, 2007.
[38] H.-S. P. Wong, C. Bencher, H. Yi, X.-Y. Bao, and L.-W. Chang, "Block copolymer directed self-assembly enables sublithographic patterning for device fabrication," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2012, p. 832303.
[39] H. Yi, X.-Y. Bao, J. Zhang, R. Tiberio, J. Conway, L.-W. Chang, S. Mitra, and H.-S. P. Wong, "Contact-hole patterning for random logic circuits using block copolymer directed self-assembly," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2012, p. 83230W.
[40] Y. Badr, A. Torres, and P. Gupta, "Mask assignment and synthesis of DSA-MP hybrid lithography for sub-7nm contacts/vias," in Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE. IEEE, 2015, pp. 1-6.
[41] Z. Xiao, C.-X. Lin, M. D. Wong, and H. Zhang, "Contact layer decomposition to enable DSA with multi-patterning technique for standard cell based layout," in Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific. IEEE, 2016, pp. 95-102.
[42] J. Kuang, J. Ye, and E. F. Young, "Simultaneous template optimization and mask assignment for DSA with multiple patterning," in Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific. IEEE, 2016, pp. 75-82.
[43] R. Pease, "Electron beam lithography," Contemporary Physics, vol. 22, no. 3, pp. 265-290, 1981.
[44] C. W. Gwyn, R. Stulen, D. Sweeney, and D. Attwood, "Extreme ultraviolet lithography," Journal of Vacuum Science \& Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena, vol. 16, no. 6, pp. 3142-3149, 1998.
[45] H. Yi, J. Bekaert, R. Gronheid, G. Vandenberghe, K. Nafus, and H.S. Wong, "Experimental study of sub-DSA resolution assist features (SDRAF)," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2015, p. 94231F.
[46] H. Yi, J. Bekaert, R. Gronheid, G. Fenger, K. Nafus, and H.-S. Wong, "Study of DSA interaction range using Gaussian convolution," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2015, p. 94232A.
[47] M. Tung, J. Doise, I. Karageorgos, J. Ryckaert, P. Wong et al., "Design strategy for layout of sub-resolution directed self-assembly assist features (SDRAFs)," in Proc. EIPBN 2016, 2016.
[48] J. Doise, J. Bekaert, B. T. Chan, S. Hong, G. Lin, and R. Gronheid, "Influence of template fill in grapho-epitaxy DSA," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2016, p. 97791G.
[49] C. H. Wallace, P. A. Nyhus, and S. S. Sivakumar, "Sub-resolution assist features," Dec. 15 2009, US Patent 7,632,610.
[50] H. Yi, X.-Y. Bao, J. Zhang, C. Bencher, L.-W. Chang, X. Chen, R. Tiberio, J. Conway, H. Dai, Y. Chen et al., "Flexible control of block copolymer directed self-assembly using small, topographical templates: Potential lithography solution for integrated circuit contact hole patterning," Advanced Materials, vol. 24, no. 23, pp. 3107-3114, 2012.
[51] I. Karageorgos, J. Ryckaert, M. C. Tung, H. Wong, R. Gronheid, J. Bekaert, E. Karageorgos, K. Croes, G. Vandenberghe, M. Stucchi et al., "Design strategy for integrating DSA via patterning in sub-7 nm interconnects," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2016, p. 97810N.
[52] P. P. Barros, A. Gharbi, A. Sarrazin, R. Tiron, N. Posseme, S. Barnola, S. Bos, C. Tallaron, G. Claveau, X. Chevalier et al., "DSA planarization approach to solve pattern density issue," in SPIE Advanced Lithography. International Society for Optics and Photonics, 2015, p. 94280D.
[53] "Gurobi." [Online]. Available: http://www.gurobi.com/

