The Dissertation Committee for Kun Yuan
certifies that this is the approved version of the following dissertation:

VLSI Physical Design Automation for Double Patterning and Emerging Lithography

Committee:

Zhigang Pan, Supervisor

Jacob Abraham

Gi-Joon Nam

Michael Orshansky

Nur Touba

VLSI Physical Design Automation for Double Patterning and Emerging Lithography

by

Kun Yuan, B.E., M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2010

To my parents and friends.

Acknowledgments

First of all, I would like to thank my supervisor Professor David Z. Pan for bringing me into the University of Texas at Austin. During my Ph.D. years, his kindness guidance and encouragement helped me enjoy the investigation of advanced topics in VLSI CAD area. He also gave invaluable advice in my future career. Without Professor Pan, I could not have proceeded so far.

I also would like to express my deep thanks to my Ph.D. committee members Prof. Jacob A. Abraham, Dr. Gi-joon Nam, Prof. Michael Orshansky and Prof. Nur Touba, for insightful advice and comments, despite of busy schedules.

I feel blessed being in the group of UTDA group. I have been being continuously assisted by smart and helpful colleagues: Tao Luo, Anand Ramalingam, Anand Rajaram, Peng Yu, Sean Xiaokang Shi, Anurag Kumar, Katrina Lu, Suhail Ahmed, James Yong-Chan Ban, Ashutosh Chakraborty, Duo Ding, Jhih-Rong Gao, Wooyoung Jang, Joydeep Mitra, Jiwoo Pak, Jae-Seok Yang, Yilin Zhang, Kiwoon Kim, Donnie Chen, Shanhu Shen, Wen Zhang, and Bei Yu. Specially, I would like to thank Minsik Cho for being my mentor during my junior years.

I also owe thanks to the members of Friday Dinner: Jun Zhang, Ningyu Shi, Xiaohui Gao, Qiu Zhao, Danhua Shao, Fangkai Yang, Hailong Xiao. I

appreciate the help from many other friends as well. To name a few, Yang Li, Shaoping Lu, Hui Chen, Wenjing Zhan, Wenfeng Qiu, Pinyu Wu, Wande Zhang and Fan Zhang.

Finally, I would like to thank my parents for their love, sacrifice and support through my pursuit of PhD degree. No words can express my appreciation.

VLSI Physical Design Automation for Double Patterning and Emerging Lithography

Publication No. _____

Kun Yuan, Ph.D.
The University of Texas at Austin, 2010

Supervisor: Zhigang Pan

Due to aggressive scaling in semiconductor industry, the traditional optical lithography system is facing great challenges printing 32nm and below circuit layouts. Various promising nanolithography techniques have been developed as alternative solutions for patterning sub-32nm feature size. This dissertation studies physical design related optimization problem for these emerging methodologies, mainly focusing on double patterning and electronic beam lithography.

Double Patterning Lithography (DPL) decomposes a single layout into two masks, and patterns the chip in two exposure steps. As a benefit, the pitch size is doubled, which enhances the resolution. However, the decomposition process is not a trivial task. Conflict and stitch are its two main manufacturing challenges.

First of all, a post-routing layout decomposer has been developed to perform simultaneous conflict and stitch minimization, making use of the integer linear programming and efficient graph reduction techniques. Compared to the previous work which optimizes conflict and stitch separately, the proposed method produces significantly better result.

Redundant via insertion, another key yield improvement technique, may increase the complexity in DPL-compliance. It could easily introduce unmanufacturable conflict, while not carefully planned and inserted. Two algorithms have been developed to take care of this redundant via DPL-compliance problem in the design side.

If design itself is not DPL-friendly, post-routing decomposition may not achieve satisfactory solution quality. An efficient framework has been further proposed to perform wire spreading for better conflict and stitch elimination. The solution quality has been improved significantly, with small layout perturbations.

As another promising solution for sub-22nm, Electronic Beam Lithography (EBL) is a maskless technology which shoots desired patterns directly into a silicon wafer, with charged particle beams. EBL overcomes the diffraction limit of light in current optical lithography system, however, the low throughput becomes its key technical hurdle. The last work of this dissertation formulates and investigates a bin-packing problem for reducing the processing time of EBL.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 Optical Lithography

Optical lithography technology [7–10] has been widely adopted in semiconductor industry for printing circuit patterns. Fig. 1.1 illustrates a typical optical lithography system for VLSI manufacturing. The mask contains the desired chip layouts, and is projected into the wafers through a couple of high-precision lens. In single exposure infrastructure, as Rayleigh criterion [11–15] describes, $hp = k_1 * \lambda/NA$, the minimum printable half pitch hp depends on three parameters. k_1 is the process difficulty factor, NA is numerical aperture and λ is the light wavelength.

As technology scales, the lithography technology [16] has been facing its limit in printing smaller feature sizes in 32nm/22nm. Currently, keeping pitch scaled becomes extremely difficult, even with all of advanced techniques: immersion lithography [17, 18], resolution enhancement techniques [19–22], restrictive design rules [23], and so on. Conventionally, the feature size is scaled down with help of smaller wavelength λ. However, the immaturity of EUV [24–27] makes the 193nm wavelength remaining the main stream lithography in

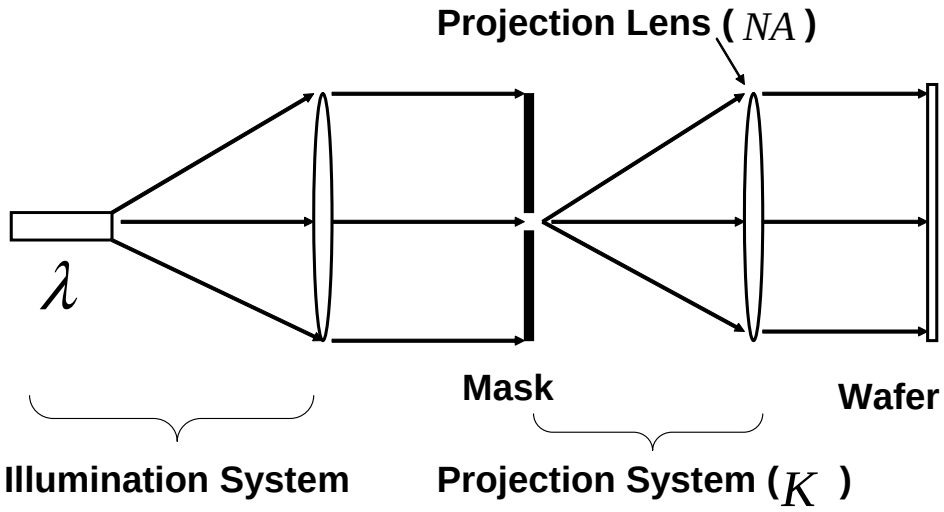


Figure 1.1: A typical optical system for circuit manufacturing.

the product line. On the other side, $k1$ has been pushed into its lower bound, and the high NA (1.35) is also challenged by the practical processing.

1.1.2 Double Patterning Lithography

Double patterning lithography (DPL)[28–34] emerges as one of the most promising alternative for 32nm/22nm nodes and it is already used for NAND-flash production. In DPL, a single layout is decomposed into two masks and manufactured through two exposure/etching steps. As a benefit, the pitch size is doubled, which enhances the resolution as illustrated in Fig. 1. Although DPL requires two masks and increases the design cost, it is widely considered as a most likely solution for 32nm, 22nm and even 16nm.

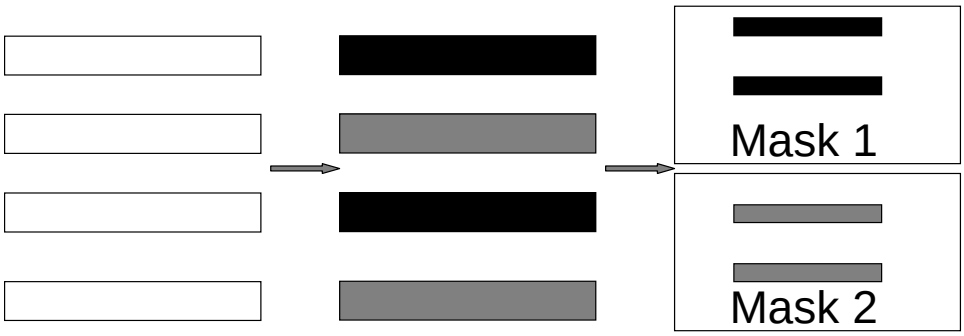


Figure 1.2: One single design is decomposed into two masks and the pitch size is increased effectively in DPL.

There are two critical issues with DPL: coloring conflict and splitting stitch.

Coloring Conflict: If the distance between two separate features is less than minimum coloring spacing min_{cs}, they should be assigned to different masks (colors). Otherwise, there will be a coloring conflict.

Fig. 1.3 (a) shows a layout with three features, and any two of them are required to have different colors because of the insufficient spacing. A coloring conflict will be unavoidable as in Fig. 1.3 (b). Sometimes, such a violation can be eliminated by appropriately splitting the features like Fig. 1.3 (c). There are also unresolvable conflicts, as Fig. 1.3 (d) indicates, which requires modifying the design.

Splitting Stitch: The stitch exists when two *touched* features are assigned to different masks. The stitch can be inserted to split some features

to resolve the conflict as shown in Fig 1.3 (c). However, stitch insertion can have negative effects on yield due to overlay error between the two masks as Fig. 1.4 (a) illustrates. In addition, the line-end will cause pattern degradation.

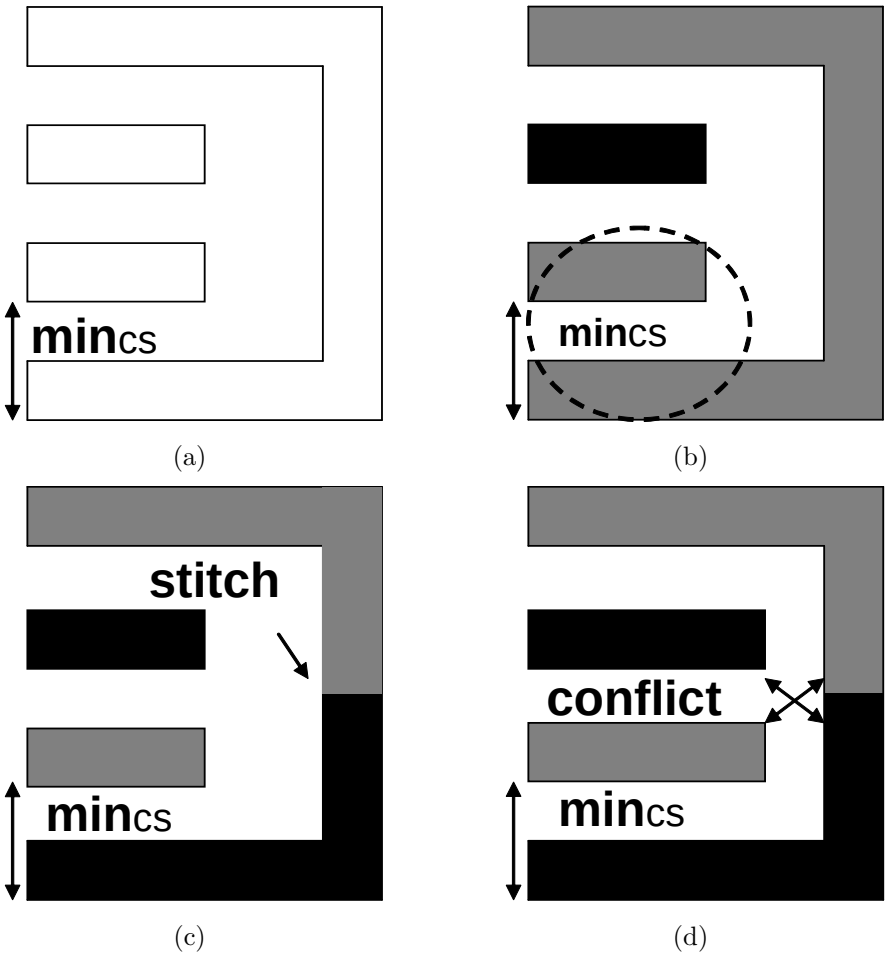
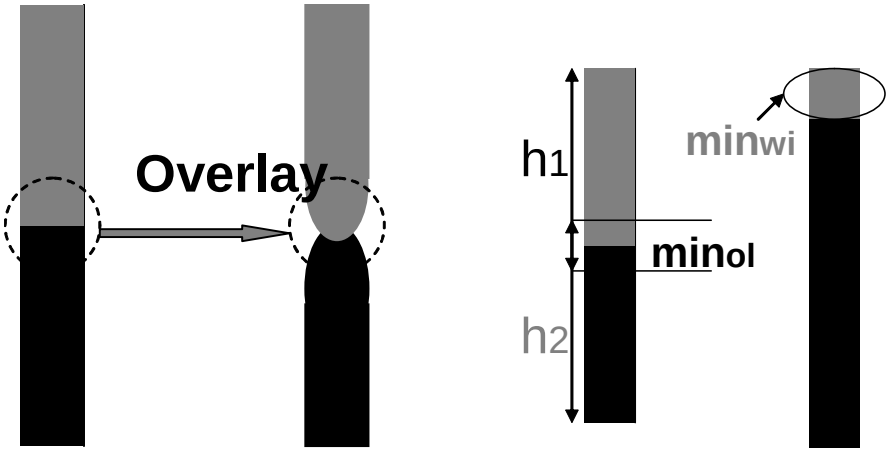


Figure 1.3: The concept of conflict.

There are several practical guidelines for splitting. As Fig. 1.4 (b) shows, in order to control the overlay, there is a *minimum overlap length* min_{ol}

requirement for stitch insertion. The segments h_1 and h_2 on different masks should be overlapped to certain amount ensuring better manufacturability. Moreover, we do not want to have any *minimum width* min_{wi} rule violation during splitting, as marked by the circle Fig. 1.4 (b).



(a) (b)

Figure 1.4: The concept of stitch

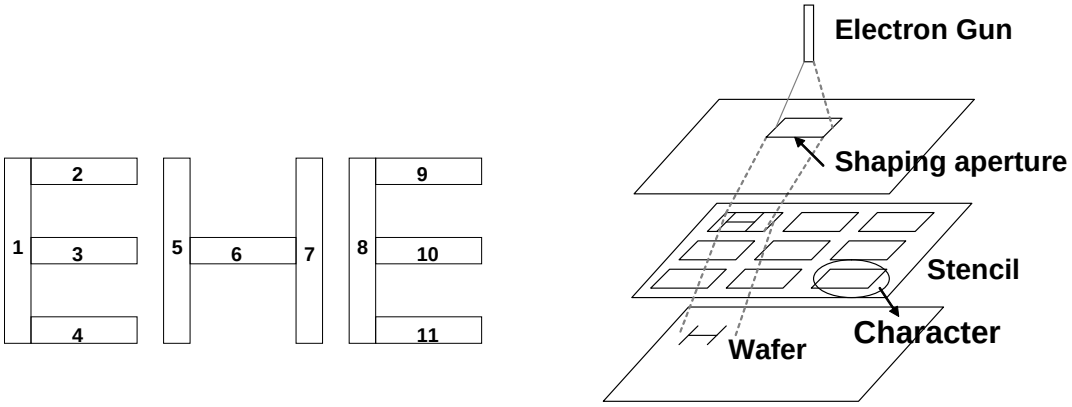
In this dissertation, this dissertation addresses on different phases of VLSI physical design for achieving good decomposition resolution, with as less as possible conflicts and stitches. First of all, a high-quality post-routing layout decomposition algorithm [35] has been developed for minimizing conflict and stitch simultaneously. Then, from the design side, a DPL-friendly routing is proposed framework [36] to take into account redundant via DPL-compliance problem. The third work is about wire spreading algorithm for conflict and stitch elimination [37].

1.1.3 Electronic Beam Lithography

Electronic Beam Lithography [38–43] is a maskless technology which shoots desired patterns directly into a silicon wafer, with charged particle beam. The primary advantage is that it is one of the ways to beat the diffraction limit of light of current well-adopted optical lithography [44]. However, the key limitation of electron beam lithography is low throughput.

The conventional type of EBL system is Variable Shaped Beam (VSB). In VSB, the layout is usually decomposed into a set of rectangles, and each one would be shot into resist by dose of electron sequentially. As Fig. 1.5 (a) shows, the pattern of "EHE" is divided into eleven rectangles and needs total eleven shots. The whole processing time of this technique increases with number of beam shots. This makes its throughput very low for modern complicated design, which is commonly composed of significant number of small rectangles.

The Character Projection (CP) technology [41–43] has been invented for improving the throughput of VSB methods. The key idea is to print some complex shapes in one electronic beam shot, rather than writing multiple small rectangles. This reduces manufacturing time significantly. In detail, as the projection system of CP in Fig. 1.5 (b) illustrates, a library of layout configurations, called **Character**s, or **Template**s, are prepared on a **stencil** first. During manufacturing, if any character exists in the targeted design, it will be chosen in the system and projected into the wafer. To print the example of Fig. 1.5 (a), suppose two characters "E" and "H" are pre designed for the stencil. By adjusting the shaping aperture, we can print the patterns



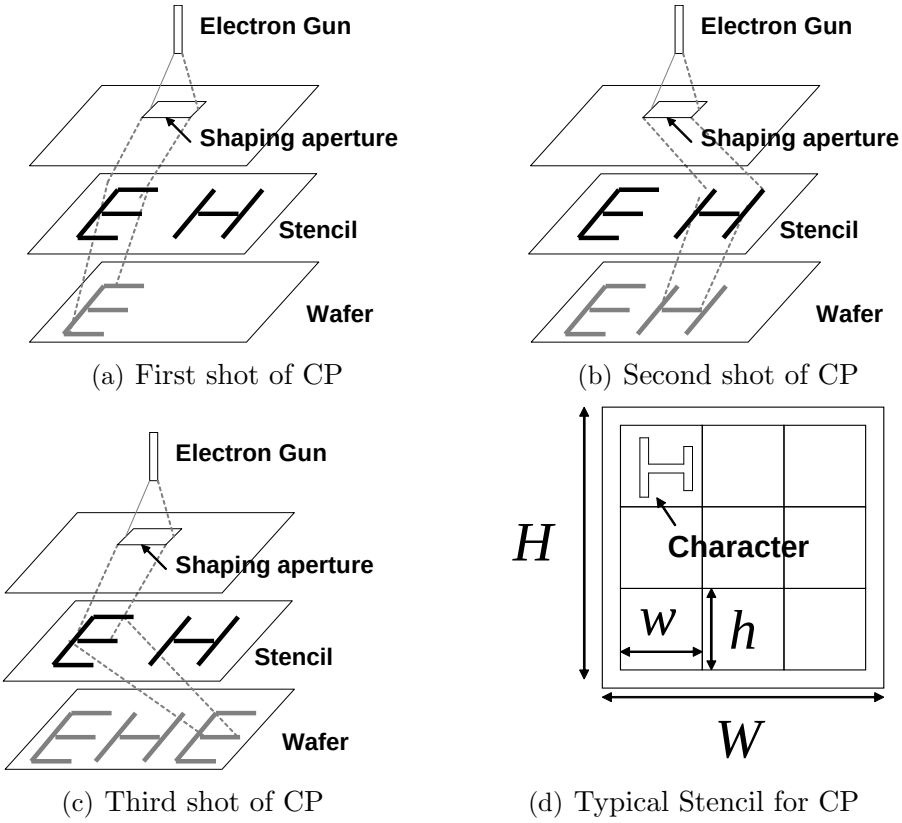
(a) VSB (b) CP

Figure 1.5: Electron Beam Lithography.

of "E", "H", "E" in sequential manner, as Fig. 1.6 (a)-(c) shows. Totally, it only takes three shots.

Due to less beam shots for the same layout, CP system is much faster than VSB. However, the number of characters is limited due to the area constraint of the stencil. As in the example of Fig. 1.5 (d), there are only maximum $\lfloor W/w \rfloor \lfloor H/h \rfloor$ characters. For modern design, it is not practical to fully make use of CP, due to numerous distinct circuit patterns. Those patterns, which do not match any character, are still required to be written by VSB.

This dissertation works on planning and optimization of electronic beam lithography stencil for throughput improvement, which aims to minimizes total projection time of both CP and VSB.



(a) First shot of CP

(b) Second shot of CP

(c) Third shot of CP

(d) Typical Stencil for CP

Figure 1.6: The details of Character Projection electronic beam lithography

1.2 Overview and Contributions of This Dissertation

This dissertation investigates four related topics in physical design automation for emerging lithographes. The first three research topics, on double patterning lithography, come from the point view of post-routing decomposition, routing, and wire spreading, respectively. The last topic studies a bin packing problem to improve overall throughput for electronic beam lithography. The main contributions of this dissertation are stated as follows:

Double Patterning Layout Decomposition for Simultaneous Conflict and Stitch Minimization: In Chapter 2, we propose an algorithm to minimize the number of conflicts and stitches simultaneously. In DPL, coloring conflict and splitting stitch are two major challenges and highly correlated with each other. While previous layout decomposition approaches perform coloring and splitting separately, their results could easily get stuck in the local optima. Our simultaneous optimization enables larger search space and produces higher quality solutions, based on grid layout model and integer linear programming. Three techniques, grid mergence, independent component computation and layout partition, are proposed to reduce runtime of proposed algorithm.

Double Patterning Friendly Detailed Routing with Redundant Via Consideration: In Chapter 3, we present the first work of considering double patterning lithography and redundant via insertion together. The metal, which is used to cover the via and its redundant via in both layers, could complicate DPL-compliant and introduce many additional conflicts and stitches. Two algorithms are proposed to perform DPL and redundant via co-optimization in post-routing and during-routing stages respectively.

Wire Spreading Enhanced Double Patterning Mask of Decomposition: Post-routing mask decomposition algorithms may not be enough to achieve high quality solution for DPL-unfriendly designs, due to complex metal patterns. In Chapter 4, an efficient framework of WISDOM has been proposed to perform wire spreading and mask assignment simultaneously for enhanced

decomposability. A set of Wire Spreading Candidates (WSC) are identified to eliminate coloring constraints or create additional splitting locations. Based on these candidates, an Integer Linear Programming (ILP) formulation is proposed to simultaneously minimize the number of conflicts and stitches, while introducing as less layout perturbation as possible. To improve scalability, three acceleration techniques are also proposed without loss of solution quality: odd-cycle union optimization, coloring-independent group computing, and suboptimal solution pruning.

Electronic Beam Lithography Stencil Planning and Optimization: Chapter 5 formulates and discusses a bin packing problem for throughput improvement of electronic beam lithography. To reduce the processing time, conventionally, some complex shapes, **characters**, are usually pre designed in a stencil. However, only limited number of characters can be employed, due to the area constraint of the stencil. In Chapter 5, a new problem of electronic beam lithography stencil design with overlapped characters is investigated. The blanking space of adjacent characters are allowed to be shared, which enables more characters to be put on the stencil, and helps increasing overall throughput. Different from previous works, besides selecting appropriate characters, their placements on the stencil are also optimized in our framework. Two algorithms are proposed to handle one and two dimensional stencil design problem, respectively.

Chapter 2

Double Patterning Layout Decomposition

2.1 Introduction

Double patterning layout decomposition[1, 2, 45–49] is a process that assigns two features within the given minimum coloring space to different masks. As mentioned in Section 1.1.2, conflict and stitch are its two major challenges.

If the spacing between two features (polygons) is less than certain minimum coloring distance, they have to be assigned opposite colors. However, a layout may contain a pattern which is unable to assign a color. In this case, a feature may be split into two parts and colored differently to resolve the conflict, which generates stitches. Stitches will cause yield loss and increase manufacturing cost due to overlay errors, which is 5 or 6nm under current 32nm double patterning lithography. Some mask misalignment direction [50] could be actually beneficial for printability. However, on the presence of various process uncertainties, such as dose, focus and mask errors, the printed stitch width could be easily smaller than 25nm and result in design failure. Pushing overlay below 3nm [51] is very challenging. Moreover, the additional line-ends may cause more pattern degradation and reduce yield in case of defo-

cus and dose variation. After splitting, a few unresolved or even unresolvable conflicts may remain and will be corrected by time consuming layout redesign. Therefore, it is important to produce high quality decomposition solution with less conflicts and stitches.

Without altering layout in the scope, the general objective of layout decomposition can be stated as minimizing the unresolved conflicts by introducing as few as possible stitches. There are a few works focusing on stand-alone layout decomposition. A heuristic approach is proposed in [2] to cut troublesome patterns after finding the coloring conflicts. The patterns are pre-fragmented into smaller pieces in [1] to perform coloring. All these works do not have a systematical way to minimize the number of conflicts and stitches. Coloring and splitting are considered in separate steps while they are highly correlated tasks. Pattern matching technique is proposed in [52] to decompose the layout. However, it might not be able to work on large scale problem, hence limits the solution quality. Recently, a practical layout decomposition flow is proposed in [53] to address design needs for double patterning. They first detect the features associated with unresolvable conflict cycles for layout modification. The remaining design is then decomposed to minimize the number of stitches based on an ILP formulation. However, in their work, the number of unresolvable conflict cycles and splitting stitches are not optimized together, and conflict elimination technique is quite greedy.

In this chapter, we propose an algorithm to decompose layout for minimizing conflicts and stitches simultaneously. The proposed approach reduces

the conflicts by 87.6% with 33% less stitches than a greedy two phase decomposition flow. When compared to the state of art methodology [53], we are also able to achieve averagely 87.2% and 10% reduction on conflicts and stitches, respectively. Although our approach is comparatively slower, we can obtain coloring solutions for all the test cases within a few minutes. The runtime shows linear complexity with respect to problem size.

The main contributions of this chapter are as follows:

1. We propose a new grid model to enable bigger solution space than previous works [1, 2] and perform simultaneous conflict and stitch optimization.

2. We develop an ILP algorithm to minimize the number of conflicts and stitches for a high quality solution.

3. We propose three speed-up techniques (grid merging, independent component computing and layout partition) to improve the runtime and scalability of our algorithm. For layout partition, we identify and solve a coloring flip optimization problem to minimize the conflicts and stitches across the boundary of different partitions.

4. We discuss how to extend our proposed grid model to handle various splitting rules and design patterns in practice.

The rest of the chapter is organized as follows. Section 2.2 provides the preliminaries and motivates. In Section 2.3, we discuss the problem for-

mulation with related model and definitions. The basic ILP formulation is described in Section 2.4 with three speed-up techniques. The extensive discussion on grid model for practical design issues is presented in Section 2.5. Section 2.6 presents the experiment results and Section 2.7 concludes this chapter.

2.2 Motivation

The previous works insert stitches after coloring to resolve conflicts. Without planning possible splitting during coloring, it is hard to eliminate the conflict. Considering a layout in Fig. 2.1 (a), we have a coloring solution in Fig. 2.1 (b). During the splitting, the U feature should be cut into two parts to remove the conflict but we have to further check whether the splitting will result in another conflict like Fig. 2.1 (c). In such case, the coloring of the neighborhood features needs to be reconsidered to avoid unnecessary stitches like Fig. 2.1 (d) and enable optimal solution in Fig. 2.1 (e) or (f). This is a simple example, but as we can see, given the complexity of modern design, the two phase approach will have extreme difficulty handling the exploding consideration and producing high quality solution . This motivates us simultaneous conflict and stitch minimization during layout decomposition.

2.3 Problem Formulation

In this section, we will first motivate and introduce our grid model in Section 2.3.1. The basic terms will be defined in the following Section 2.3.2.

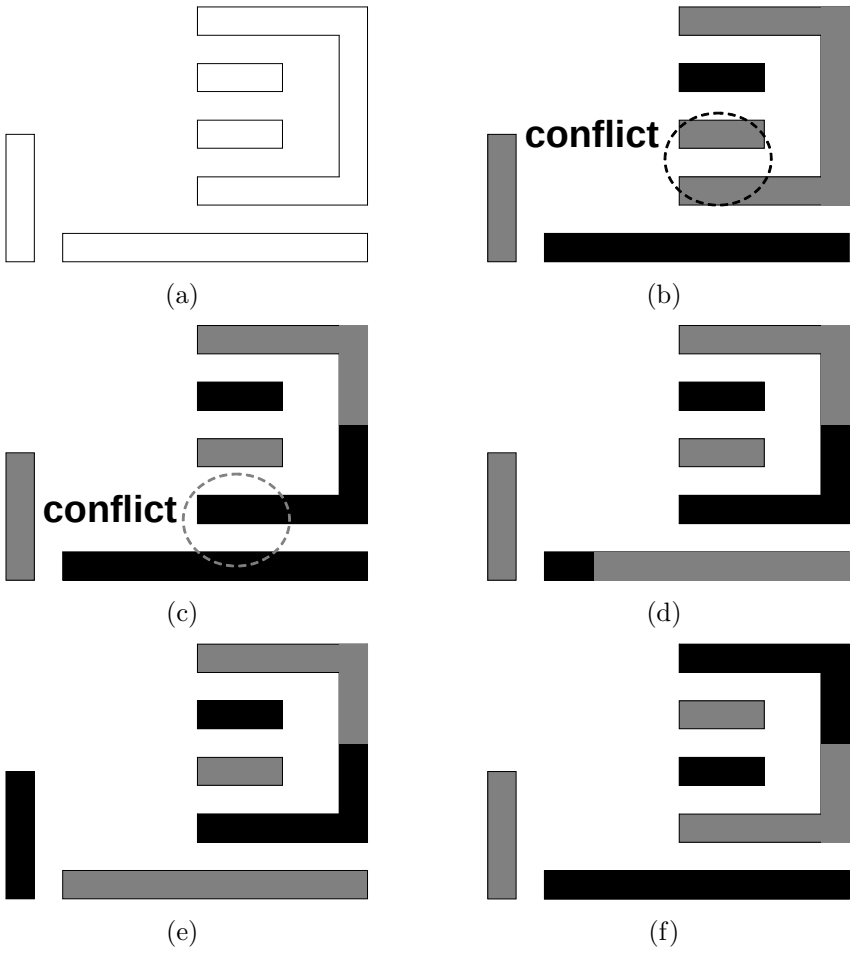


Figure 2.1: The shortcoming of two phase layout decomposition flow in previous works [1, 2]. An unplanned coloring will need much extra effort during splitting.

The formal problem definition will be described in the end.

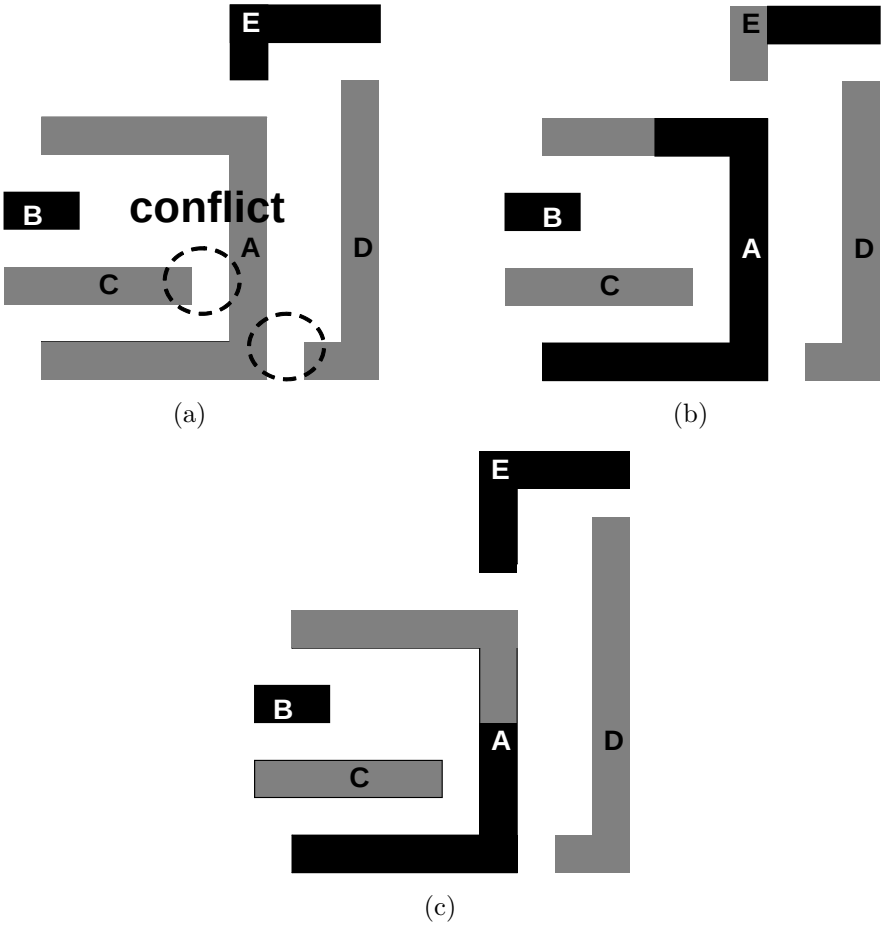


Figure 2.2: Different stitch candidates can lead to different solution qualities.

2.3.1 Grid Layout Model

Considering splitting during coloring is a challenging problem. First of all, the stitch configurations are highly correlated and all the potential locations need to be considered for global optimality. Fig. 2.2 (a) is a case with two conflicts. As we can see, two possible splitting choices on feature

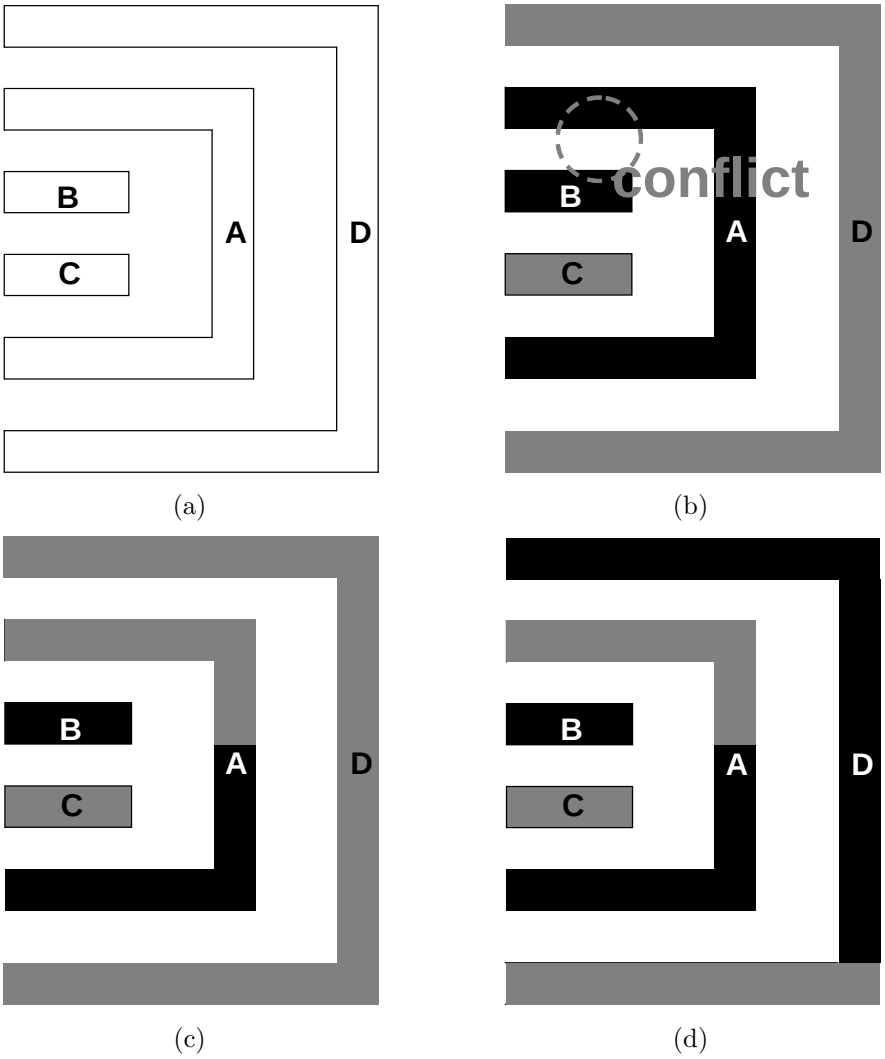


Figure 2.3: The difficulty of predicting where the splitting is needed

A lead to two different solutions, Fig. 2.2 (b) and (c). The first one has two stitches, where the latter one associates with only one. Moreover, we can even hardly predict where we could have a splitting due to some *chain* effect. For

example, the right most feature D is not expected to be cut in Fig. 2.3 (a) because it is only adjacent to one single feature A. However, given a coloring assignment as shown in Fig. 2.3 (b), feature A will be split to resolve the conflict between A and B like Fig. 2.3 (c). As a result, feature D also needs to be broken into two segments as shown in Fig. 2.3 (d).

In order to overcome these issues, we will map the whole layout into grids with its size to be half the pitch of the original design. Each grid is either empty or fully occupied by the pattern, and each occupied grid will be assigned one color. Therefore, any boundary between grids is a potential splitting location. This is shown in the Fig. 2.4. Essentially, we provide fine resolution for splitting options. This model is able to offer sufficient stitch candidates for all the features across the design in practice and the solution space is much bigger than previous works [1, 2]. The discretization is reasonable because a design follows underlying regular pitches in modern layout. Minimum coloring spacing min_{cs} is taken as two-grid size to double the spacing for each mask in this chapter and also subject to change according to given min_{cs}.

2.3.2 Terms and Problem Formulation

Before formulating our problem, we will first define the terms in the grid layout model.

Definition 1. occupied grid ***OG****: The grid filled by the layout.*

The OG must be assigned one of the two colors: GRAY and BLACK.

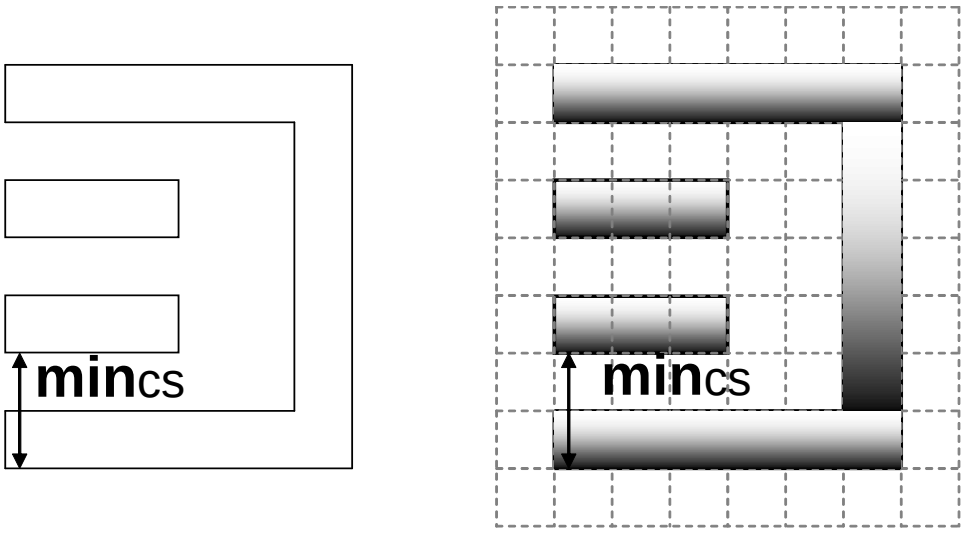


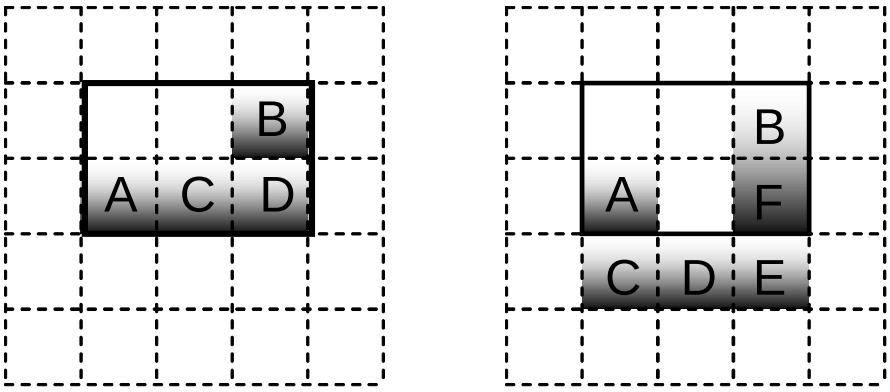
Figure 2.4: The proposed grid layout model.

Definition 2. blocking path ***BP****: Given two occupied grids OG_1 and OG_2, a blocking path is a path when*

1. It is fully composed of OGs and connects OG_1 and OG_2.
2. OG_1 and OG_2 are touching its two ending grids respectively but not belonging to this path.
3. This path is within the bounding box of OG_1 and OG_2.

The main usage of blocking path is to identify neighboring but locally isolated layout grids. These grids, even belonging to the same connection, need to be considered as different features, which could form a coloring conflict.

As shown in Fig. 2.5 (a), C-D is a blocking path for grid A and B. In another example Fig. 2.5 (b), C-F is not a BP for A-B, because not all of them are in the bounding box of A-B as the third rule defines. Some part of it (C-E) is beyond the box, and hence locally A-B can be considered as isolated.



(a) (b)

Figure 2.5: The concept of blocking path. The solid rectangle marks the bounding box.

Definition 3. potential conflict grid pair ***PCGP*** *and* potential stitch grid pair ***PSGP****: Given two occupied grids* OG_1 *and* OG_2*,*

1. *If the distance between* OG_1 *and* OG_2 *is less than* min_{cs} *and the two grids are not touching, they form a potential conflict grid pair.*

2. *If* OG_1 *and* OG_2 *are touching, they form a potential stitch grid pair.*

The distance between a pair of OGs is the minimum distance between any two points from the OGs. For example in Fig. 2.5 (b), the distance for

untouched B and C is $\sqrt{2}$ grid size due to two closest corners, which is smaller than min_{cs}. Therefore, they form a PCGP.

Definition 4. stitch grid pair ***SGP****: If the grids of a PSGP are assigned different colors, it is a stitch grid pair.*

Definition 5. conflict grid pair ***CGP****: If a PCGP is in the identical color, and there is no blocking path connecting them in the same mask, it is a conflict grid pair.*

The definition of SGP is straightforward as grids A and B shown in Fig. 2.6 (a). Fig. 2.6 (b) shows the normal CGP cases, where a PCGP is colored identically and unconnected. B-F and A are within the minimum coloring spacing. There are even no paths connecting them, not to mention blocking path. The rule one of Definition 2 is violated. As a result, any of B-F and A are a CGP.

There are also some special CGP cases that we need to further consider blocking path in order to avoid false recognition of lithography friendly pattern. If two non-touching grids are electrically connected through a blocking path, we should not consider them belonging to different features. The printability will not be an issue. As shown in Fig. 2.6 (c), grid A and B have a BP C-D in the same mask between them, so they do not form a CGP. It is indeed a normal jog, and can be printed well. In contrast, although there is a path C-F connecting A and B in Fig. 2.6 (d), C-E is out of their bounding box. In consequence, the path is not a blocking path. This violates the third rule of

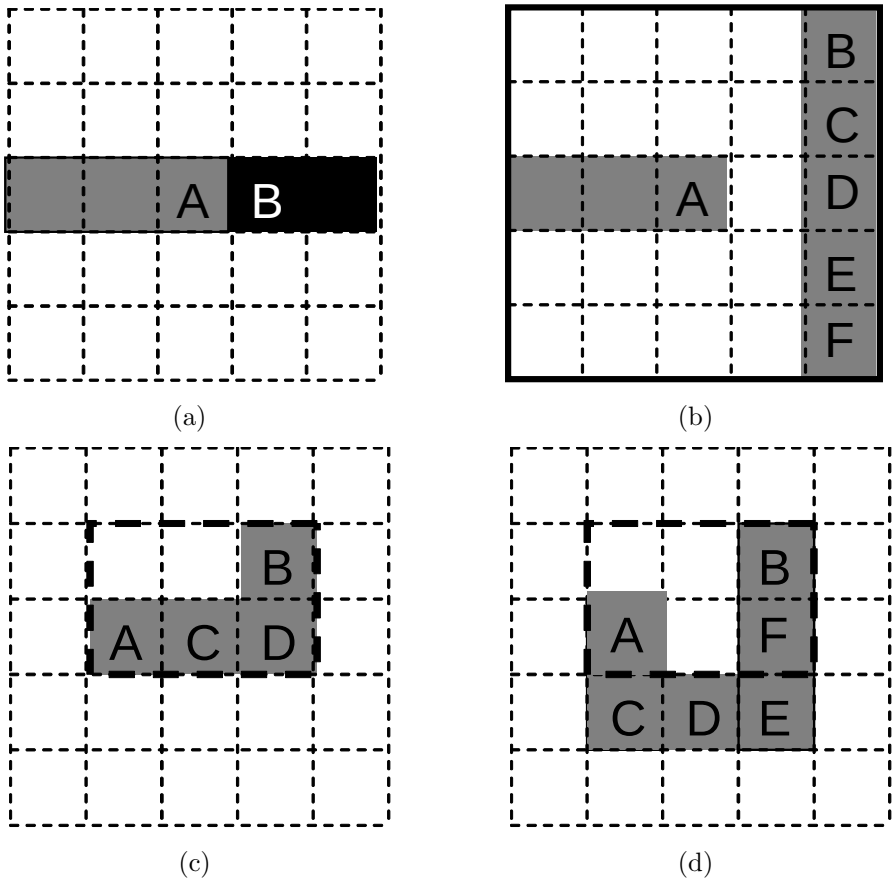


Figure 2.6: Stitch grid pair and conflict grid pair. The dash box in (c) and (d) is the bounding box of A and B.

Definition 2, so grid A and B form a CGP. In this case, A and B are in fact locally isolated but neighboring within the bounding box. This configuration is a typical U shape pattern, and would have weak printability.

2.3.3 Problem Description

In our work, we use the number of SGPs and CGPs as the cost, which assigns higher weight to the grids that are associated with more conflicts/stitches. Formally, we formulate the layout decomposition optimization problem as follows:

Problem Formulation : *Given a grid layout, color it into two parts (GRAY and BLACK). The primary objective is to minimize the number of CGPs and the second objective is to minimize the number of SGPs.*

We prefer a solution with less CGPs than one with smaller number of SGPs but more CGPs, because a layout with non zero CGPs is essentially not manufacturable and a solution with less CGPs reduces expensive redesign effort.

2.4 Algorithm

In this section, we will present our ILP based layout decomposition algorithm. The entire flow is shown in Figure 2.7. After mapping the design to grid model, we will process the grids and formulate the basic ILP formulation. Since the timing complexity for ILP is very high, we will then propose three speed-up techniques by either eliminating unnecessary variables or divided the whole problem into several smaller ones. Finally, the layout decomposition for the entire design can be obtained by merging the subproblem solutions. For better solution reunion, we formulate a problem of coloring flipping optimiza-

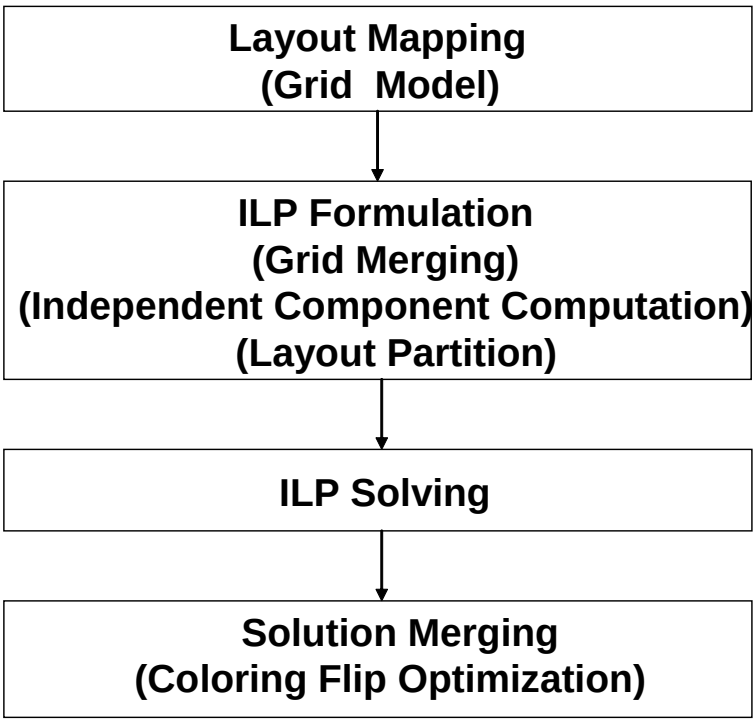


Figure 2.7: The overall layout decomposition flow.

tion through ILP.

2.4.1 Basic ILP Formulation

To better present our method, we first describe the notation in Table 4.1. The simultaneous coloring and splitting optimization can be formulated as follows:

$$min(\sum_{s_{ij,mn} \in SP} s_{ij,mn} + \alpha \sum_{c_{pq,uv} \in CP} c_{pq,uv}) \tag{2.1}$$

Table 2.1: Notation for basic ILP formulation

$og_{i,j}$	occupied grid which i and j are its coordinates.
$x_{i,j}$	binary variable that denotes the color of $og_{i,j}$. $x_{i,j} = 1$ if the color is GRAY, otherwise, it is BLACK.
$s_{ij,mn}$	binary variable $s_{ij,mn} = 1$ if $og_{i,j}$ and $og_{m,n}$ is a SGP.
$c_{pq,uv}$	binary variable $c_{pq,uv} = 1$ if $og_{p,q}$ and $og_{u,v}$ is a CGP.
SP	the set of PSGPs.
CP	the set of PCGPs.
$P_{pq,uv}$	the set of BPs connecting $og_{p,q}$ and $og_{u,v}$.
$p^k_{pq,uv}$	the k_{th} BP connecting $og_{p,q}$ and $og_{u,v}$.
$n^k_{pq,uv}$	the number of grids in $p^k_{pq,uv}$.
$g^k_{pq,uv}$	binary variable $g_{pq,uv} = 1$ if $p^k_{pq,uv}$ is a GRAY BP.
$b^k_{pq,uv}$	binary variable $b_{pq,uv} = 1$ if $p^k_{pq,uv}$ is a BLACK BP.

subject to

$$x_{i,j} + (1 - x_{m,n}) \leq 1 + s_{ij,mn} \quad \forall s_{ij,mn} \in SP \tag{2.2}$$

$$(1 - x_{i,j}) + x_{m,n} \leq 1 + s_{ij,mn} \quad \forall s_{ij,mn} \in SP \tag{2.3}$$

$$\sum_{x_{e,f} \in p^k_{pq,uv}} x_{e,f} \leq (n^k_{pq,uv} - 1) + g^k_{pq,uv} \quad \forall p^k_{pq,uv} \in P_{pq,uv} \tag{2.4}$$

$$\sum_{x_{e,f} \in p^k_{pq,uv}} (1 - x_{e,f}) \leq n^k_{pq,uv}(1 - g^k_{pq,uv}) \quad \forall p^k_{pq,uv} \in P_{pq,uv} \tag{2.5}$$

$$\sum_{x_{e,f} \in p^k_{pq,uv}} (1 - x_{e,f}) \leq (n^k_{pq,uv} - 1) + b^k_{pq,uv} \quad \forall p^k_{pq,uv} \in P_{pq,uv} \tag{2.6}$$

$$\sum_{x_{e,f} \in p^k_{pq,uv}} x_{e,f} \leq n^k_{pq,uv}(1 - b^k_{pq,uv}) \quad \forall p^k_{pq,uv} \in P_{pq,uv} \tag{2.7}$$

$$x_{p,q} + x_{u,v} \leq 1 + c_{pq,uv} + \sum_k g^k_{pq,uv} \quad \forall c_{pq,uv} \in CP \tag{2.8}$$

$$(1 - x_{p,q}) + (1 - x_{u,v}) \leq 1 + c_{pq,uv} + \sum_k b^k_{pq,uv} \quad \forall c_{pq,uv} \in CP \tag{2.9}$$

The objective function (3.1) is to minimize the weighted summation of SGPs and CGPs. Parameter α is used to tune the relative importance between SGP and CGP and can be set to ensure the priority of CGP elimination. All the PCGPs and PSGPs are pre-determined by examining the neighboring grids for each OG.

Constraints (4.6) and (4.7) are used to identify SGP from PSGP. According to the definition of SGP, we need to know whether the PSGP grids have opposite colors. Whenever $x_{i,j}$ and $x_{m,n}$ have opposite values, the left hand side of one of the constraints will be two. As a result, $s_{ij,mn}$ must be assigned one to satisfy the constraints, which detects a SGP.

The usage of Constraints (4.2)-(2.9) is to determine whether a PCGP forms a CGP. Identifying CGP takes more effort. Besides checking the colors of PCGP, we need to know whether there is a blocking path in the same mask. All the possible BPs $P_{pq,uv}$ can be easily enumerated by depth first search on the occupied grids within the bounding box. We can investigate their coloring using Constraints (4.2)-(4.5). The corresponding binary variable $g^k_{pq,uv}/b^k_{pq,uv}$ will be true only if the grids of some blocking path are in the same mask. Constraints (2.8) and (2.9) evaluate the conditions for CGP. A conflict will be reported only if PCGP grids are assigned same color and the possible BPs $g^k_{pq,uv}/b^k_{pq,uv}$ do not exist.

Let n_{og} be the number of occupied grids, the basic formulation contains at most $O(n_{og})$ variables. The constraints are specified for detecting either PSGPs or PCGPs. Suppose there are n_{sp} PSGPs and n_{cp} PCGPs, the complexity of n_{sp} is $O(n_{og})$. n_{cp} is linearly related to n_{og}, but quadratically proportional to min_{cs}. The complexity of constraints due to PSGPs is $O(n_{sp})$. The constraint number for PCGPs is linear proportional to n_{cp}. It is also exponentially related to min_{cs}, which results from the enumeration of blocking paths. Although this formulation shows exponential complexity in terms of min_{cs}, when we fix the value of min_{cs} as the pre-setting for layout decomposition, the number of variables and constraints is quadratic with respect to n_{og}.

The proposed integer linear formulation can minimize the number of conflicts and stitches simultaneously. However, because ILP is NP-complete, it is not affordable to directly apply a basic ILP formulation for large modern designs.

2.4.2 Speed-Up Techniques

In this section, we will discuss three speed-up techniques. The clustering methodology is applied in grid merging to reduce the number of variables and constraints. In contrast, the key idea of the other two techniques is to use a divide and conquer algorithm to convert the problem into smaller subproblems.

2.4.2.1 Grid Merging

In the proposed grid model, we aim to provide very fine resolution for stitch candidates. This may be over skilled under certain situations.

Consider the layout segment L in Fig. 2.8 (a) with unit grids A-B-C-D. Only the two ending points A and D may have coloring interaction with other layout objects besides L. B and C can be considered as *isolated* to some extent. Because there are no occupied grids outside A-B-C-D which are touching them or within min_{cs} of their boundary. Therefore, it is not possible for B or C to form a stitch or conflict with other layout apart from the grids of segment L.

We can make advantage of above property to reduce problem size by combining this type of connected grids into a big super grid. As graphically shown in Fig. 2.8 (b), B and C can be treated as a united grid T. This is equivalent to enforce B and C the same color. It will not deteriorate the conflict and stitch optimization. For this super grid, it does not have any chance to form a conflict or stitch with surrounding grids other than its two adjacent grids A and D.

The elimination of internal splitting candidates is not a problem for solution quality. For any optimized solution obtained under original grid model with internal stitches, it can be mapped to one solution in the merged model with the stitch propagated to its ending grids, such as from Figure. 2.8 (c) to Figure. 2.8 (d).

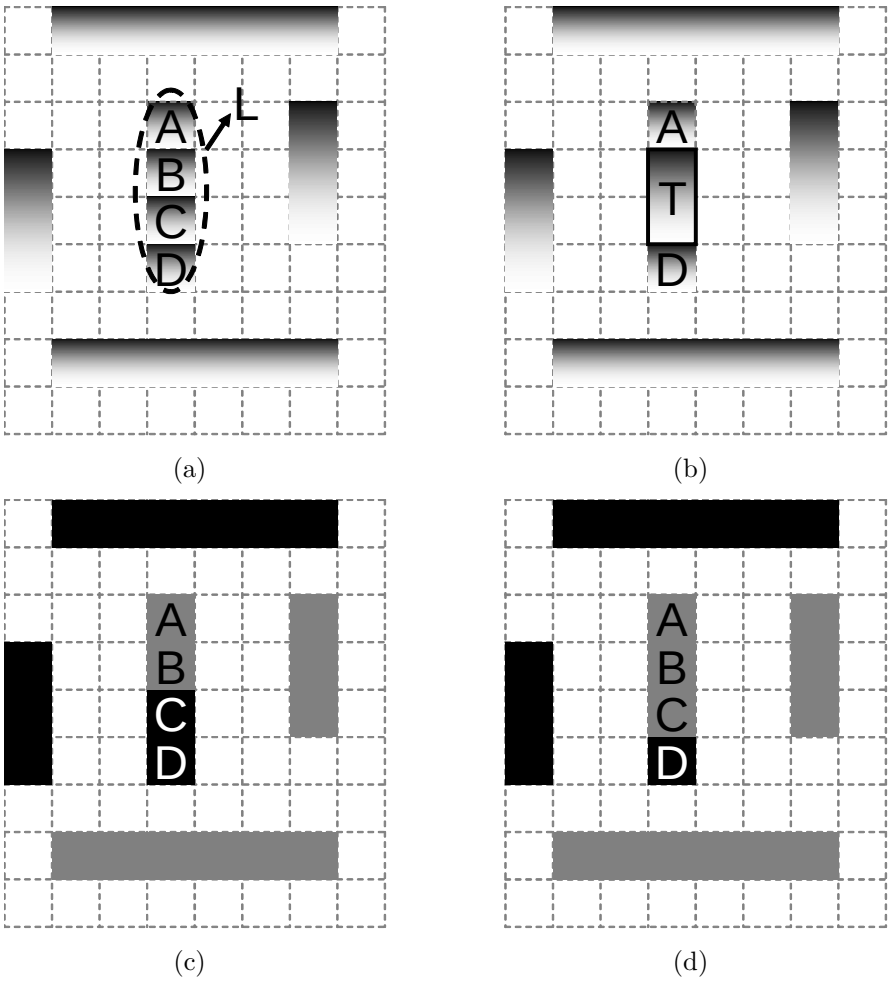


Figure 2.8: The main idea of grid merging.

2.4.2.2 Independent Component Computation

We propose independent component computation for reducing the ILP problem size without losing optimality. In real layout, we observe many *isolated* occupied grid clusters, i.e. there are no PSGPs or PCGPs formed between

them. Therefore, we can break down the whole design into several independent components as shown in Fig. 2.9, and apply a basic ILP formulation for each one. The overall solution can be taken as the union of all the components without affecting the global optimality. The runtime of ILP formulation scales down dramatically with the reduction of the variables and constraint.

Our independent component finding algorithm is given in Algorithm 1. The timing complexity of this algorithm is $O(V + E)$, which V is the total number of the OGs and E is the total number of PSGPs and PCGPs.

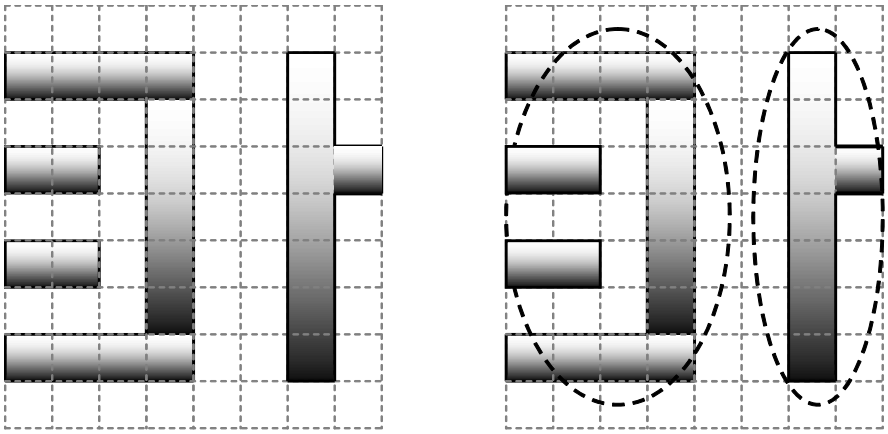


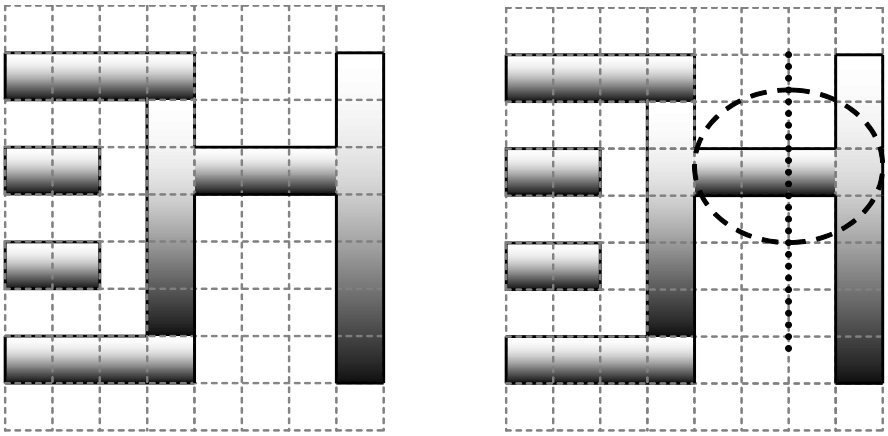
Figure 2.9: An example of breaking big layout into two independent components, having no interacted PSGPs/PCGPs and marked by the dash circle

Algorithm 1 Independent Components Finding

Require: The grid layout

Ensure: The independent components, having no PSGPs/PCGPs between any pair of components

1: Build a graph G(V,E), $V \in \phi, E \in \phi$.
2: **for** each OG $og_{i,j}$ **do**
3: Create one graph node $v_{i,j}$.
4: **end for**
5: **for** each PSGP/PCGP ($og_{i,j}$, $og_{m,n}$) **do**
6: Create one edge between $v_{i,j}$ and $v_{m,n}$.
7: **end for**
8: Perform the depth first search on the graph G to find the independent components.
9: Map the graph nodes in each component back to OGs $og_{i,j}$ and return.



(a) (b)

Figure 2.10: An example of layout partition. The dotted line cuts the layout into two parts while the dash circle marks PCGP and PSGP locations across the boundary of the two partitions

2.4.2.3 Layout Partition

Some components may still have prohibitive problem size even after independent component computation. Our heuristic is to divide a big component into several small connected partitions and perform an ILP approach for each one, indicated in Fig. 2.10. Different from the independent component computation, there will be some PSGPs/PCGPs between different partitions. Although we solve each partition by ILP, the united solution does not guarantee to be optimal for the whole component in terms of ILP objective since the partition boundaries are not considered in the optimization.

In order to minimize the loss of global optimality, we need to partition the circuit with as few as possible cuts while ensuring that each partition can be efficiently solved by ILP. Balanced min-cut partition method is applied in our work. We first construct a graph G which is the same as in independent component computing. For each vertex (OG), we assign a weight as its edge degree plus one, taking into account the number of both variables and constraints it associates with. A threshold W_t is pre-defined for the maximum node weight summation we allow for each partition. The number of partitions can be calculated as $\lceil \frac{W}{W_t} \rceil$, where W is the total vertex weight of G. Suppose W is 10000 and W_t is 3000, the component will be partitioned into 4 parts.

2.4.3 Solution Merging

After solving the solution for each component/partition, we need to merge the coloring assignment as a whole. While it is trivial to combine

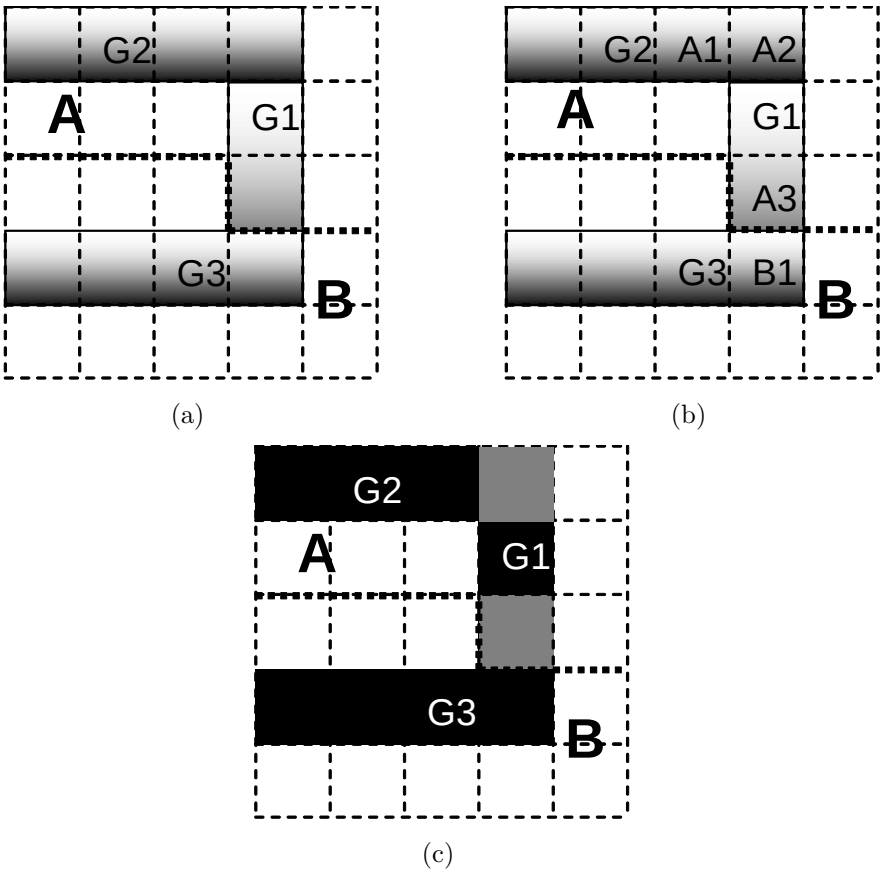


Figure 2.11: The "internal" and "external" concepts. The wide solid line is the boundary of different partitions.

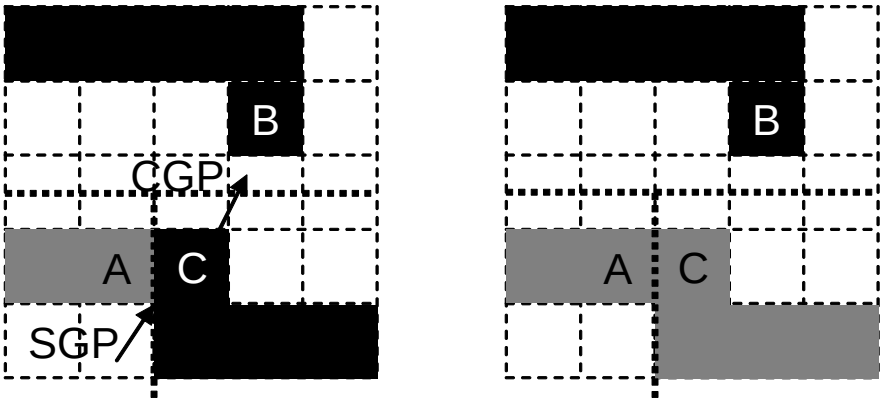
the solutions for smaller independent components, there comes a *coloring flip optimization* problem when we try to merge the solutions of all the partitions for the bigger components with partitioning applied.

In layout partition, the PSGPs and PCGPs for each partition can be divided into two disjoint subsets. The internal stitch/conflict grid pairs

$PSGP^i$s/$PCGP^i$s, and external ones $PSGP^e$s/$PCGP^e$s. If the associated grids, which are needed for identifying whether a $PSGP/PCGP$ is a SGP/CGP, are all within the same partition, this $PSGP/PCGP$ belongs to $PSGP^i$s/$PCGP^i$s, otherwise, it is considered as a $PSGP^e/PCGP^e$. Similarly, during the unitization, the SGPs and CGPs for each partition can be categorized as SGP^is/CGP^is and SGP^es/CGP^es. SGP^is/CGP^is are from $PSGP^i$s/$PCGP^i$s, and SGP^es/CGP^es are from $PSGP^e$s/$PCGP^e$s.

As illustrated in Fig. 2.11 (a), there are two partitions A and B. Suppose we are considering two $PCGP$s, (G_1, G_2) and (G_1, G_3), (A_1, A_2) and (C_1, C_2) are their additional associated grids respectively for correctly identifying a CGP, indicated by Fig. 2.11 (b). (G_1, G_2) is a $PCGP^i$ because the grids which are related to (G_1, G_2, A_1, A_2) are all in partition A. In contrast, (G_1, G_3) is a $PCGP^e$ while (G_1, A_3) belongs to partition A and (G_3, B_1) is in partition B. Similarly, in one possible coloring configuration in Fig. 2.11 (c), (G_1, G_2) is a CGP^i and (G_1, G_3) is a CGP^e.

During the solution union, it is possible to reduce the number of SGP^es/CGP^es by flipping the coloring of some partition. More importantly, such flipping will not change the status of SGP^is/CGP^is. In detail, it will not introduce new SGP^is/CGP^is, and any existing SGP^i/CGP^i will not go away as well. Based on the above definition, the related grids for identifying a SGP^i/CGP^i are in a single partition. Their coloring will be either flipped or not simultaneously. The conclusion of whether the respective $PSGP^i/PCGP^i$ is a SGP^i/CGP^i will not be changed.



(a) (b)

Figure 2.12: Different coloring flips have distinct numbers of SGPs/CGPs across the boundaries, marked by the dot lines.

The effect of coloring optimization is illustrated in Fig. 2.12, which has three partitions. The coloring merging in Fig. 2.12(a) produces one SGP and one CGP across the boundaries. If we flip the coloring of partition C from the BLACK to GRAY, it becomes a SGP/CGP free assignment in Fig. 2.12(b). To optimize the flipping scheme, we define *coloring flip optimization* as follows.

Coloring Flip Optimization: *Given a number of partitions and their coloring solutions for one independent component, choose the best flipping scheme to minimize total cost of* SGP^e *and* CGP^e*, which cross the boundaries among all the partitions.*

Because the number of partitions is usually not large, we also use an ILP formulation to solve this problem. The relevant notation can be found in

Table 2.2.

Table 2.2: The notation for coloring flipping problem

f_i	binary variable $f_i = 1$ if partition i flips its coloring.
$f^0_{i,j}$	binary variable $f^0_{i,j} = 1$ if both partitions flip or do not flip the coloring.
$f^1_{i,j}$	binary variable $f^1_{i,j} = 1$ if only one partition between i and j flips its coloring.
$s^{e0}_{i,j}$	the number of stitches between partition i and j if both flip or do not flip the coloring.
$c^{e0}_{i,j}$	the number of conflicts between partition i and j if both flip or do not flip the coloring.
$s^{e1}_{i,j}$	the number of stitches between partition i and j if only one partition flips its coloring.
$c^{e1}_{i,j}$	the number of conflicts between partition i and j if only one partition flips its coloring.

The formulation is as follows:

$$min \sum (f^0_{i,j}(s^{e0}_{i,j} + \alpha c^{e0}_{i,j}) + f^1_{i,j}(s^{e1}_{i,j} + \alpha c^{e1}_{i,j})) \quad \forall i, j \tag{2.10}$$

subject to

$$f_i + f_j \leq 1 + f^0_{i,j} \tag{2.11}$$

$$(1 - f_i) + (1 - f_j) \leq 1 + f^0_{i,j} \tag{2.12}$$

$$f_i + (1 - f_j) \leq 1 + f^1_{i,j} \tag{2.13}$$

$$f_j + (1 - f_i) \leq 1 + f^1_{i,j} \tag{2.14}$$

$$f^0_{i,j} + f^1_{i,j} = 1 \tag{2.15}$$

Our objective function (2.10) is to minimize the number of SGP^e and CGP^e. The same α as basic ILP formulation in Section. 2.4.1 is used for balancing the cost. For each pair of partitions, there are two cases: (1) only one of them

is flipped; (2) flipping both or none of them. We can easily pre compute the cost for each case, stored as $(s^{e0}_{i,j} + \alpha c^{e0}_{i,j})$ or $(s^{e1}_{i,j} + \alpha c^{e1}_{i,j})$.

Constraints (2.11) and (2.12) specify the case if both or neither of the partitions flip their coloring. Constraints (2.13) and (2.14) specify the case if only one of two partitions flip the coloring. Only one case can happen and this is formulated as Constraint (2.15).

It should be noted that, in our implementation, we do not explicitly impose Constraint (2.15). Instead, we substitute f^1_{ij} by $(1 - f^0_{ij})$ in (2.10)-(2.14) based on (2.15). This helps further reduce the number of variables and constraints.

2.5 Grid model for practical design issues

In this section, we will present how our proposed grid model can handle various splitting rules and design patterns in Section 2.5.1 and Section 2.5.2 respectively.

2.5.1 Practical Splitting Rules

Various manufacturability issues could impose many practical constraints on the locations of the stitches. Our grid model can be extended to satisfy these requirements. In the following, we will mainly focus on two major DPL-related guidelines, minimum width and minimum overlapping requirements.

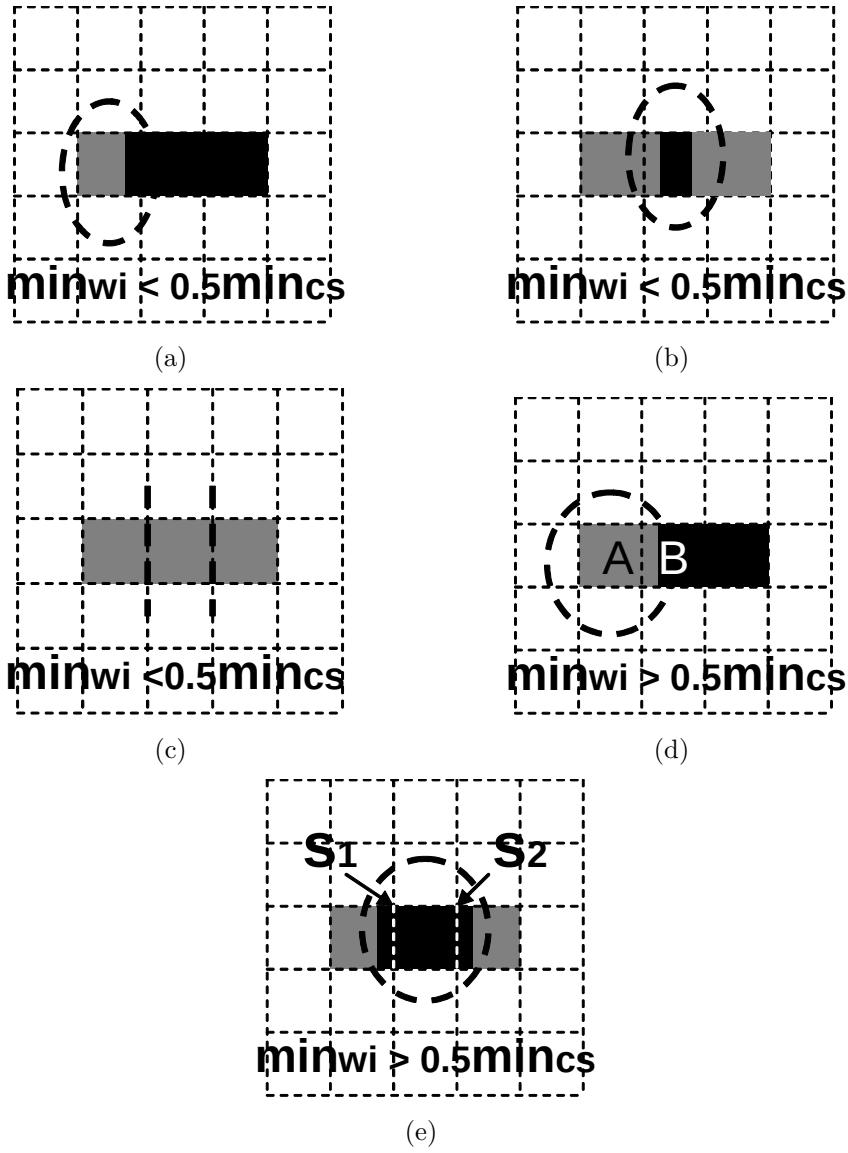


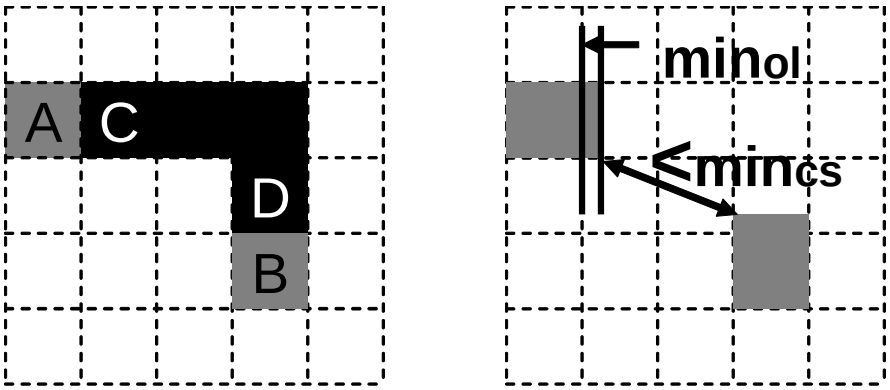
Figure 2.13: The grid model can handle minimum width requirement.

2.5.1.1 minimum width violation

The min_{wi} violation can be resulted from careless splittings as Fig. 2.13 shows. It could be located in the ending parts of polygons like Fig. 2.13 (a) and (d), or created by two close stitches as Fig. 2.13 (b) and (e) show.

In most process technology, min_{wi} is smaller than or equal to the half pitch, $0.5min_{cs}$, which is illustrated in Fig. 2.13 (a) and (b). Our grid model can successfully avoid these extra constraints implicitly. By only allowing splitting on the boundary of the grids as shown in Fig. 2.13 (c), the resulting small layout segments from splitting will be bounded from lower side by one grid size, i.e. $0.5min_{cs}$. The minimum width rule will be automatically satisfied, and there will be no pitfalls when we work on grids.

For the technology which has larger than one grid width min_{wi}, additional constraints can be augmented into our ILP formulation to ensure minimum width rule. We assume min_{wi} is still less than two-grid width here just for illustration purpose, and similar ideas can be applied for even larger min_{wi} requirement. For the example in Fig. 2.13 (d), we can enforce the coloring of grid A and B identical to avoid minimum width violation. We are also able to specify constraints to eliminate the situation resulting from adjacent stitches as Fig. 2.13 (e) indicates. A pair of stitches, S_1 and S_2, within one grid distance will be forbidden to exist simultaneously.



(a) (b)

Figure 2.14: The grid model can handle minimum overlap requirement.

2.5.1.2 minimum overlapping margin

The possible min_{ol} violation comes from the extra extension over the splitting locations, which may result in additional coloring conflict, such as the case from Fig 2.14 (a) to Fig 2.14 (b). A and B initially do not form a PCGP based on the definition in Section 2.3.2, although they are in the same color. On the existence of some possible splitting, the extended metal could bring them into a distance smaller than min_{cs}, which causes a coloring conflict.

This issue does exist in the process with 5-6nm overlay error. To encounter this problem, when we are extracting PCGPs, the extra extension needs to be included for calculating distance between two grids. As the example in Fig 2.14, when A and C or B and D have different colors, the overlay error should be considered.

On the other side, benefiting from possible further improvement on optical engineering, the overlap margin may not be a problem for our grid model in most cases. Remind that we are performing optimization based on the grids. For any pair of grids which do not form a PCGP, at least one of x and y dimensional distance will be two grid size, min_{cs}. As the research works show [51], min_{ol} will be possibly controlled below 3nm. With such a small overlap margin, the diagonal distance between A and B will still be larger than min_{cs}.

2.5.2 Non-grid-mappable Layout

The grid model not only works on regular designs, it can also be extended to handle non-grid-mappable layout.

2.5.2.1 Off-grid Layout

In deep sub-micro technology, although on-grid patterns are commonly favorable, there still exist off-track wires on lower layer metals, as illustrated by Fig. 2.15. (a). Pattern A is not aligned with the grid lines. Under such case, we are not able to apply our grid-based formulation directly.

To resolve this issue, if a point has any layout object, we will assign a binary grid variable for it. This is denoted as "relaxed grid mapping". As the example Fig. 2.15. (b) shows, grids A1 and A2 will both be considered occupied. Moreover, to exclude false detection, additional consideration will be required when we formulate our mathematical programming,

First of all, we need to pay extra effort to check whether a pair of grids are within min_{cs} or connected, which is one crucial factor for determining PCGP or PSGP. Instead of using the grid number based measurement as in Section. 2.3.2, we have to work on the distance or connection information of the underlying physical layouts.

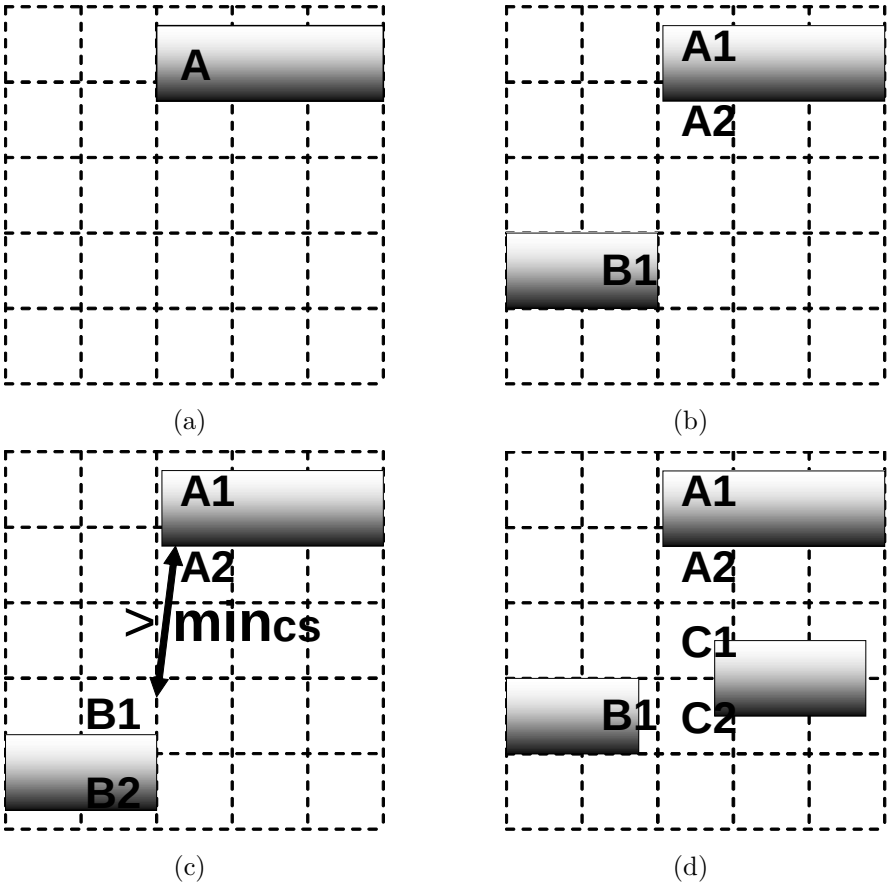


Figure 2.15: The grid model can handle off-grid layout.

Fig. 2.15. (b) and (c) show the need for checking min_{cs} condition for

PCGP by distance, not the number of grids. In Fig. 2.15. (b), the distance between grids A2 and B1 is one grid unit, smaller than two-grid unit threshold. This is consistent with the fact that the distance between related patterns is smaller than min_{cs}. They indeed form a PCGP. On the other side, in Fig. 2.15. (c), although grids A2 and B1 are away from each other by one grid, the distance of underlying design objects is no less than min_{cs}. They are not a PCGP. If determining the distance only by grid based unit, we will draw false conclusion. Similarly, it is the right way to determine whether two grids are linked by checking the layouts instead of grid occupancy status. Fig. 2.15. (d) shows an example where A1 and B1 are actually not connected. Grid based judgment will falsely consider they are linked together because A1, B1, C1 and C2 are all occupied grids.

Moreover, we also need to exclude the unfeasible stitch locations resulting from "relaxed grid mapping". For the stitch candidate between A1 and A2 in Fig. 2.15. (b), since the splitting will cause minimum width violation, it should be forbidden.

2.5.2.2 Fat Wire

When dealing with fat wires, we can map them into multiple grids. Although this will increase the complexity of the ILP formulation because of the dense clustering of occupied grids. However, we can apply previously proposed grid merging technique to reduce the problem size. Practically, a great portion of grids inside the wide wire can be merged.

2.6 Experimental Results

In our benchmarks, eight industrial designs are scaled down to 32nm. The metal1 for each test case is used for the experiments, which is one of the most troublesome layers in terms of double patterning lithography. The detailed information is shown in Table 5.1. The first column "ckt" denotes the circuit name, "area" is the chip area in terms of um^2, "grid array size" shows the number of rows by the number of columns in our layout grid array. "#OG", "#PCGP" and "#PSGP" give the number of OGs, PCGPs and PSGPs respectively.

We implement our algorithm in C++ and test on Intel Core 3.0GHz Linux machine with 32G RAM. Moreover, we use glpk [54] as our ILP solver and hMetis [55] for min cut partition. The threshold W_g for each partition is 1500. We study different α settings in the ILP objective function. As shown in Fig. 2.16, when we start to increase α with higher penalty on conflict, the number of CGPs/SGPs drops/climbs obviously. After certain value, it has little effect, because the ILP formulation has reached its best point to reduce conflicts. In our work, we set α as 10 for all the benchmarks.

2.6.1 Result Comparison

We implement two different layout decomposition algorithms for comparison. To be fair, the same conflict interpretation defined in Section 2.3.2 will be applied in our experiments. In detail, although two features belong to the same net, as long as they are locally isolated, they could still result in a

Table 2.3: The test cases.

ckt	area	grid array size	#OG	#PCGP	#PSGP
C1	89	294x294	6670	21215	5926
C2	160	395x395	15710	48007	14143
C3	207	450x450	20496	63403	18461
C4	292	534x534	33497	105641	30314
C5	422	642x642	53998	172826	49167
C6	540	726x726	68820	214527	62387
C7	747	854x854	101431	323890	92493
C8	1028	1002x1002	142535	447441	129172

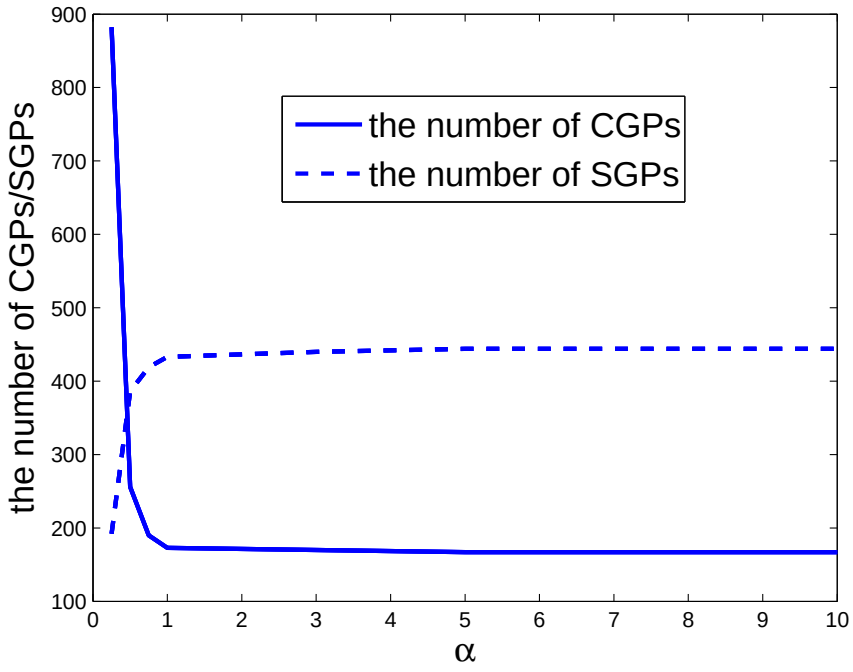


Figure 2.16: The performance of our algorithm with different α values.

conflict.

We first prepare a greedy two-phase layout decomposition flow for comparative study, which adopts construct-and-fix methodology as in the previous works [1, 2]. We first color all the layout features sequentially. Each feature will be assigned to a color which can minimize the current number of conflicts. In the second phase, the violations are detected and corrected by inserting stitches. This is done by flipping the coloring of conflict segments, which basically splits certain features. Finally, the decomposition solution is mapped back to grids for comparison.

As the second comparative method, we also implement a design methodology based on the previous ILP-based work [53]. The conflict cycle will be removed iteratively first, followed by an ILP formulation to minimize the number of stitches. We are not able to compare with [53] directly because some of our main objectives are different. In their work, the unresolved conflict cycle is used for judging the indecomposable patterns, while we apply much finer metric, conflict pair grid. To resolve the issue, as the last step, we perform an additional grid based greedy coloring run for the detected unresolved conflict cycles. The decomposition results will be mapped into grids in the end.

The detailed comparison is shown in Table 2.7. Under "Two phase approach", "1CGP" is the number of CGPs after the first step coloring and "uCGP" is the number of unresolved CGPs after inserting stitches. "CGP" under "Previous ILP-based work" and "Our algorithm" shows the final unresolved CGPs. We also list the results of "Previous ILP-based work" when

the conflict cycle removal iteration is set as 1 and 5, which are reported in columns with postfix name "Ite. 1" and "Ite. 5" respectively. For all the three approaches, "SGP" is the final number of stitch grid pairs and "CPU" is the runtime by second. "total" is the total number of all the testcases, and "ratio" is the average of their individual ratios.

Although "Two phase approach" is much faster, our algorithm significantly outperforms its results in terms of quality. "Two phase approach" can indeed eliminate the number of CGPs by averagely 52% after inserting stitches. However, lack of the careful planning, their coloring in the first step produces very poor starting solution, and there are a big amount of unresolved conflicts left after possible splitting. In contrast, our simultaneous method can averagely reduce the number of unresolved conflict grid pairs by about 87.6% with 33% less stitch grid pairs.

When compared to the previous ILP-based work[53], we can also achieve averagely 87.2% conflict and 10% stitch reduction. "Previous ILP-based work" only greedily eliminates the troublesome conflict cycle without global picture in mind. Although a little better than "Two phase approach", their approach generates much degraded results than our algorithm in terms of conflict. On the other side, because it applies ILP to optimize the stitch number, their splitting decision is close to our simultaneous optimization result. However, because "Previous ILP-based work" also considers coloring and splitting separately, its stitch number is still 10% more than ours. Also, from the breakdown of the solutions by different number iterations, we can see that iterative con-

flict removal can help improve results but still not enough due to the lack of global view.

In DPL, zero CGPs is desired in final tape out but the high complexity of modern designs makes it almost a must to go though tedious *design-decomposition-redesign* iterations. Our simultaneous flow with much higher quality solution can reduce expensive redesign effort as well as the number of iterations, which may eventually converge to a *clean* design much more quickly. Runtime for layout decomposition is not an issue as long as it is affordable.

2.6.2 Efficiency

The naive implementation of basic ILP formulation has prohibitive problem size, and it is not able to finish any benchmark in one day. Comparatively, our algorithm effectively reduces the runtime. In Table 2.7, the column "CPU" under "Our algorithm" shows that we can obtain the solution in a few seconds. For the biggest benchmark, it takes a little more than one minute. Fig. 2.17 also shows the scalability of our algorithm, and the runtime grows linearly with the number of occupied grids in the design. Moreover, our acceleration techniques sacrifice little optimality.

Next, we will show the effectiveness of our grid merging technique. We achieve the same number of conflict and stitch number for all the testcases with and without this option while independent component computing and layout partition are still applied. Fig. 2.18 also illustrates the runtime comparison. The number on the bar is the exact CPU time in terms of second. As it is

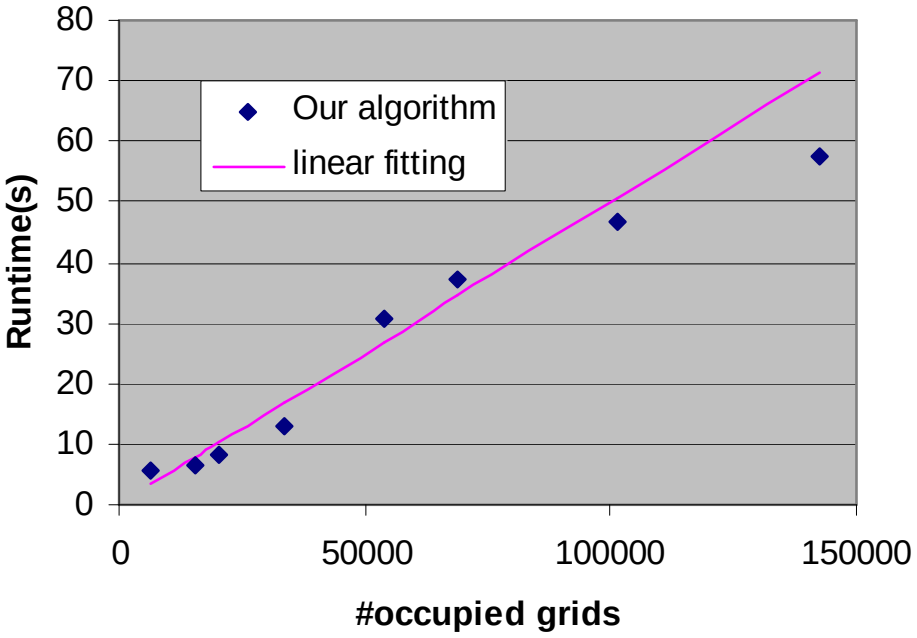


Figure 2.17: The runtime of our algorithm vs. number of occupied grids.

indicated, we can achieve approximately 19% speed-up. This mainly comes from the reduction of variables and constants in the mathematical formulation.

Table 2.4 lists the statistics on the independent components. "#InComp" is the total number of independent components. "#w/o partition" and "%w/o partition" respectively show the number and ratio of independent components, which are under partition threshold value W_t. As we can see, most components can be directly handled by ILP without performing layout partition and losing any optimality.

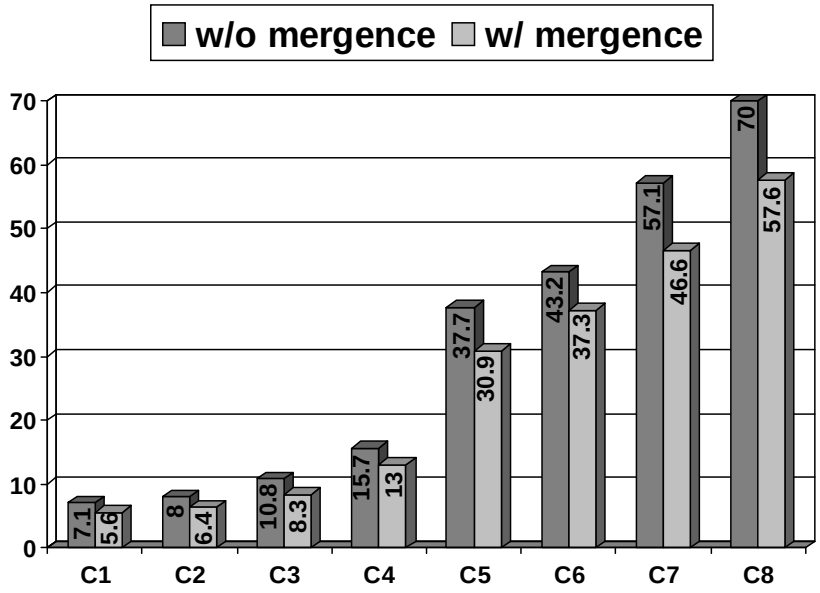


Figure 2.18: The CPU times of our algorithm with and without grid merging techniques

Table 2.4: Statistics on the independent components

ckt	#InComp	#w/o partition	%w/o partition
C1	181	178	98.3%
C2	362	357	98.6%
C3	688	681	99.0%
C4	838	824	98.3%
C5	1088	1070	98.3%
C6	1442	1420	98.5%
C7	1977	1951	98.7%
C8	3179	3147	99.0%

Table 2.5: ILP formulation statistics

ckt	reduced problem size		coloring flipping	
	$\#max_v$	$\#max_c$	$\#max_v^{cl}$	$\#max_c^{cl}$
C1	804	1333	2	4
C2	867	1445	2	4
C3	873	1435	2	4
C4	904	1469	3	12
C5	911	1499	2	4
C6	902	1478	3	8
C7	921	1511	3	12
C8	923	1522	4	20

Table 2.6: Results on coloring flip optimization

ckt	Without coloring flip				With coloring flip			
	CGP_{lp}	SGP_{lp}	CGP_{lp}^e	SGP_{lp}^e	CGP_{lp}	SGP_{lp}	CGP_{lp}^e	SGP_{lp}^e
C1	28	21	1	5	27	20	0	4
C2	18	22	9	4	12	20	3	2
C3	16	22	2	5	14	19	0	2
C4	37	70	10	11	31	66	4	7
C5	121	172	105	22	36	171	20	21
C6	65	98	13	20	55	90	3	12
C7	79	105	33	23	55	92	17	10
C8	108	142	79	28	88	127	59	13
total	472	652	252	118	318	605	106	71
ratio	1	1	1	1	0.75	0.92	0.30	0.60

Table 2.5 further shows the statistics on our ILP problem size. "$\#max_v$" and "$\#max_c$" respectively give the maximum number of variables and constraints of the basic formulation with three proposed reduction techniques applied. Moreover, "$\#max_v^{cl}$" and "$\#max_c^{cl}$" list the maximum number of variables and constraints respectively of ILP formulation, which is applied in

the coloring flip optimization.

As we can see from Table 2.5, the maximum ILP size is well controlled by the layout partition through the tuning threshold parameter W_g. W_g explicitly sets the upper bound for total number of grids, SGPs and CGPs within each sub problem. Therefore, the number of variables and constraints can be implicitly ensured in a reasonable range. Moreover, Table 2.5 indicates the coloring flip optimization has relatively very small problem size, and hence can be handled with little effort.

2.6.3 Coloring Flip Optimization

Table 4.3 shows the improvement when coloring flip is applied to merge solutions. This optimization will only be applied to relative bigger independent components, which require proposed layout partition technique to further reduce problem size. Therefore, in Table 4.3, we only list the statistics for these bigger components in the respective benchmarks. The conflict and stitch number from smaller components without layout partitioning applied are not included.

In Table 4.3, "CGP_{lp}" and "SGP_{lp}" denote the total number of CGPs and SGPs for the independent components which adopt layout partition. The percentage of this type of components is very small, as shown in Table 2.4. However, their conflict and stitch number have relatively much bigger portion when compared to the respective data under column "Our algorithm" in Table 2.7.

"CGP_{lp}^{e}" and "SGP_{lp}^{e}" are the number of corresponding external conflict and stitch grid pairs. The results show that there are outstanding "CGP_{lp}^{e}" and "SGP_{lp}^{e}" for further optimization. "with coloring flip" can reduce CGP_{lp}^{e} and SGP_{lp}^{e} by 70% and 40%, about 25% and 8% for total CGPs and SGPs. This experiment demonstrates the necessity of coloring flip optimization and the effectiveness of our ILP based approach. The CPU time difference between "Without coloring flip" and "With coloring flip" is very tiny and not listed.

2.7 Summary

In this chapter, we have developed a double patterning aware layout decomposition flow for simultaneous conflict and stitch minimization. Our approach is featured by grid layout model and integer linear programming. We also propose three speed-up techniques: grid mergence, independent component computation and layout partition. We can successfully handle the piratical guidelines for layout splitting.

Table 2.7: Result comparison

	Two phase approach				Previous ILP-based work [53]							Our algorithm		
ckt	1CGP	uCGP	SGP	CPU(s)	CGP(Ite. 1)	SGP(Ite. 1)	CGP(Ite. 5)	SGP(Ite. 5)	CGP	SGP	CPU	CGP	SGP	CPU(s)
C1	401	272	70	0.2	413	98	412	98	412	98	0.8	110	88	5.6
C2	1765	939	389	0.4	1089	287	1029	283	1015	282	1.7	160	220	6.4
C3	1799	779	416	0.5	774	206	735	199	720	198	1.8	129	175	8.3
C4	4232	2084	620	0.6	2143	469	1998	463	1972	459	2.9	171	452	13.0
C5	8125	4408	1325	1.0	3886	1078	3503	1056	3478	1059	4.9	367	1001	30.9
C6	9052	4625	1621	1.2	5082	1297	4761	1282	4731	1291	5.2	607	1112	37.3
C7	13607	5551	2753	1.8	6415	1831	5653	1811	5530	1817	7.6	606	1651	46.6
C8	18975	9223	3038	2.4	9805	2599	9050	2503	8941	2510	11.8	949	2271	57.6
total	57956	27881	10232	8.1	29607	7865	27141	7695	26799	7714	36.7	3099	6970	205.7
ratio	16.6	8.1	1.5	0.043	9.55	1.13	8.76	1.10	7.9	1.12	0.18	1	1	1

Chapter 3

Double Patterning Friendly Routing with Redundant Via Consideration

3.1 Introduction

Double patterning lithography [29] is considered as a most likely solution for 32nm/22nm technology. In DPL, the original layout is decomposed into two masks (colors), e.g., BLACK and GRAY. However, this introduces the challenges of minimizing conflict and stitch. Most researches [53] [35] focus on post layout decomposition but it may be not enough for poor designs, especially due to complex 2D patterns in lower metal layers. DPL-friendliness is needed from design side. Recently, Cho et al [4] proposed the first DPL-friendly detailed routing for highly decomposable routing.

Another key yield improvement technique is the redundant via insertion [56, 57]. As VLSI technology scales, the design complexity increases dramatically with six or more metal layers. The number of vias, which are used to connect the routes from adjacent layers, has been increasing at a tremendous place. A via may have open failure due to various reasons such as random defects, electromigration, cut misalignment, and/or thermal stress induced voiding effects. Figure 3.1 shows an electromigration-induced earlier failure

with void in via. While a completely failed via could leave nets open in circuit and result in wrong functionality, partial failure can introduce unexpected delay due to the increase of the via resistance. This significantly degrades chip performance and reduces the manufacturing yield.

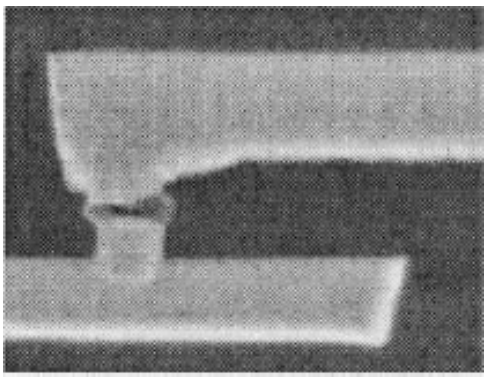


Figure 3.1: A SEM cross section of a failed open via [3].

Redundant via [56, 58–63] (double via) has been widely applied in major foundries [64] to reduce the negative impacts due to via failure. The essential idea is to enhance a single via manufactuability/reliability by adding an additional via adjacently as a fault-tolerant backup, while the minimum spacing rules are still obeyed. Redundant vias can typically lead to 10-100x lower failure rate [57] than single via, and 6% increase in good wafer yield [56].

This insertion of redundant via could introduce complexity in DPL compliance. The challenge comes from the metal which is used to cover the via and the redundant via in both layers. It is commonly referred as extra metal. Besides not violating the minimum spacing min_{sp} rule, minimum coloring

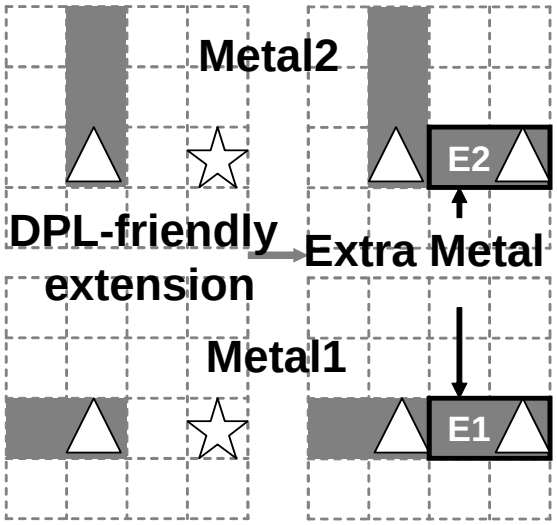


Figure 3.2: This figures illustrates the redundant via and extra metal it introduces. A via or redundant via goes through the corresponding triangles in metal1 and metal2. The star denotes the feasible redundant via candidate without violating min_{sp}. E1 and E2 are the extra metals, enclosed by solid rectangles.

spacing min_{dp} needs to be satisfied as well to avoid conflicts. Fig. 3.2 shows a motivational example, where the top figures are metal2 and the bottoms are for metal1. As Fig. 3.2 indicates, E1 and E2 are the extra metals. To minimize the number of stitches, we prefer assigning them the same color as the metal the via touches in corresponding layer, which is referred as *stitch-free extension*. However, this may cause conflicts due to the coloring assignment of existing layout. In Fig. 3.3, the *stitch-free extension* will introduce a conflict in both metal1 and metal2. To resolve this issue, as Fig. 3.4 illustrates, we can flip the coloring of the extra metal E2 by introducing an additional stitch.

However, it is not always possible to remove the conflict, such as the one in metal1. In such case, we need to modify the layout and move one of the three features.

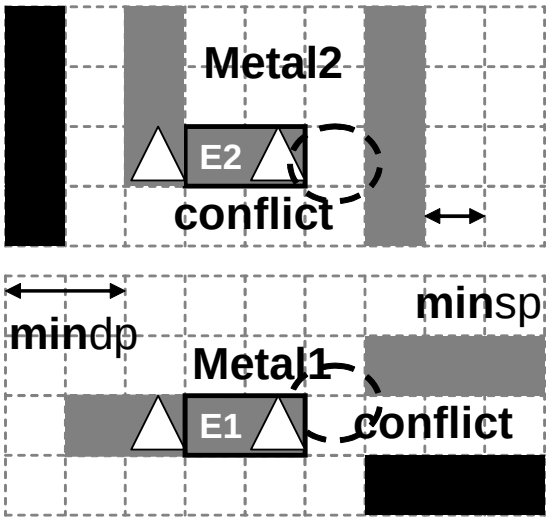


Figure 3.3: This figures illustrates redundant via DPL-compliance problem.

In this chapter, we consider DPL and redundant via insertion together. We develop two algorithms, *post-routing DPL-aware insertion* and *DPL-friendly routing with redundant via consideration*, to take into account redundant via DPL-compliance. Experimental results show that, compared to a DPL-aware optimization flow without redundant via consideration, we can improve insertion rate by 43% while still achieving zero coloring conflicts. Moreover, we can reduce the number of vias and stitches by 9% and 17% respectively.

The rest of the chapter is organized as follows. Section 3.2 introduces

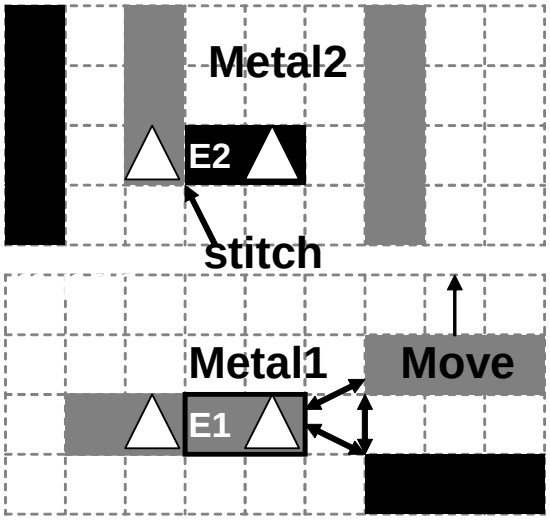


Figure 3.4: This figures illustrates that it is not easy to fix the redundant via introduced conflict.

basic definitions and previous works. Section 3.3 and 3.4 will discuss our two algorithms respectively. Section 3.5 presents the experiment results and Section 3.6 concludes this chapter.

3.2 Preliminaries and Previous Work

3.2.1 Preliminaries

Our algorithm works on routing grids. The min_{sp} and min_{dp} for metals and via cuts are taken as one-grid and two-grid size respectively, but they can be easily extended to other values. Only metal1 and metal2 are in our discussion, because the detailed routing and DPL are mainly for low metal layers.

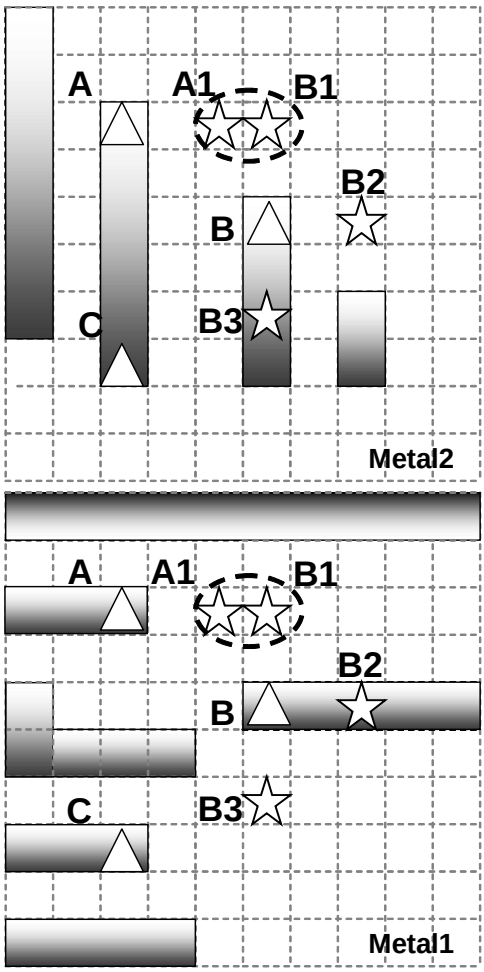


Figure 3.5: This figure illustrates the basic concepts of *feasible* redundant via candidate, *dead* and *alive* via. The triangle denotes via or inserted redundant via and the star denotes the *feasible* redundant via candidate.

We explain some key concepts with help of Fig 3.5-3.7. If a redundant via candidate for a via does not violate min_{sp} in the existing layout, we refer it as *feasible*. If a via does not have any *feasible* candidate, it is *dead*, otherwise

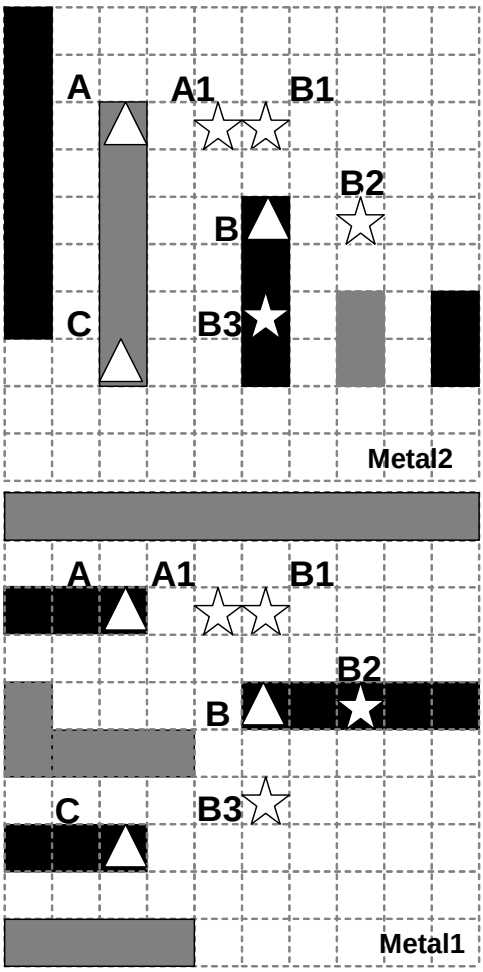


Figure 3.6: This figure shows a layout coloring, which has troubling to insert DPL-friendly redundant vias.

alive. If two *feasible* candidates of different vias can not be inserted at the same time due to min_{sp} constraints, there is an *external conflict* between them. As Fig. 3.5 shows, there are three vias A, B, and C. Via C is a *dead* via. A and B are *alive*, and their *feasible* candidates are A1 and B1-B3 respectively. There

is an *external conflict* between A1 and B1. Instead, we can select A1 and B2 as redundant via solutions without violating min_{sp}. However, under DPL, when min_{dp} is additionally considered, the *feasible* candidates may result in conflicts or extra stitches. We refer to a *feasible* candidate as *DPL-feasible*, if its *stitch-free extension* configuration will not cause conflicts under the pre determined layout and coloring. Basically, a *DPL-feasible* candidate has a DPL-friendly, conflict and stitch free, redundant via solution. If a via does not have any *DPL-feasible* candidate, it is *DPL-dead*, otherwise *DPL-alive*. With the coloring assignment in Fig. 3.6, if the same solutions A1 and B2 are selected, it will produce two *coloring conflicts* as Fig. 3.7 illustrates when *stitch-free extension* is applied. A is a *DPL-dead* via because its only *feasible* candidate A1 is not *DPL-feasible*.

3.2.2 Previous work on DPL-friendly routing

To improve layout decomposability in metal layers, Cho [4] et al. proposed a DPL-friendly simultaneous routing and decomposition using *coloring path* and *color shadow* techniques. A two bit-variable is introduced for each grid to keep track of the colorability information, which is one of the four states $\{BG,\ B\overline{G},\ \overline{B}G,\ \overline{BG}\}$. As a preprocessing, all the routing blockages will be colored by a layout decomposition algorithm and *color shadow* will be performed to generate the starting grid state map for all the routing grids. In *color shadow*, the state of surrounding grids within min_{dp} distance from the colored layout will be assigned. For example, grids near a GRAY grid will

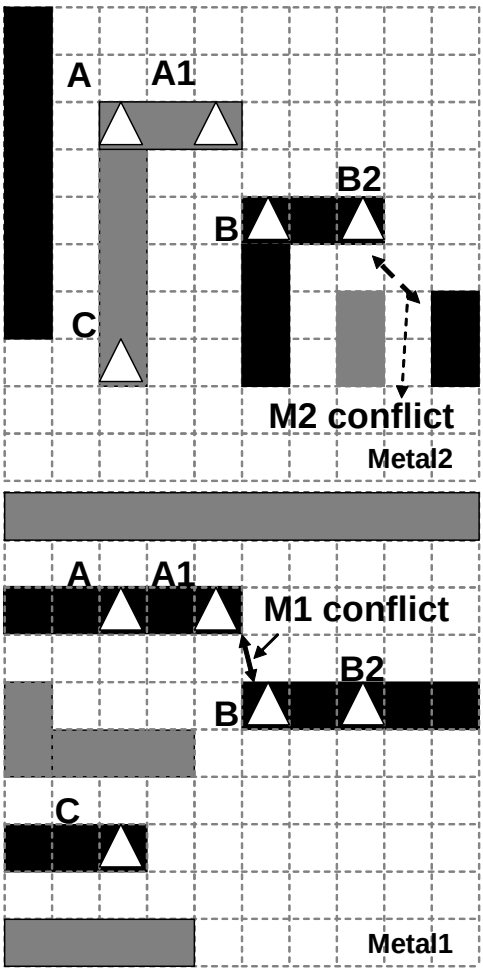


Figure 3.7: This figures illustrates the basic concepts of *DPL-feasible* candidate, *DPL-dead* and *DPL-alive* via.

have either $B\overline{G}$ or $\overline{BG}$ states.

When each net is to be routed, other than the traditional costs, they also apply DPL awareness penalty, based on current routing solution and grid states. As an example, Fig. 3.8 shows the existing configuration, and DPL-

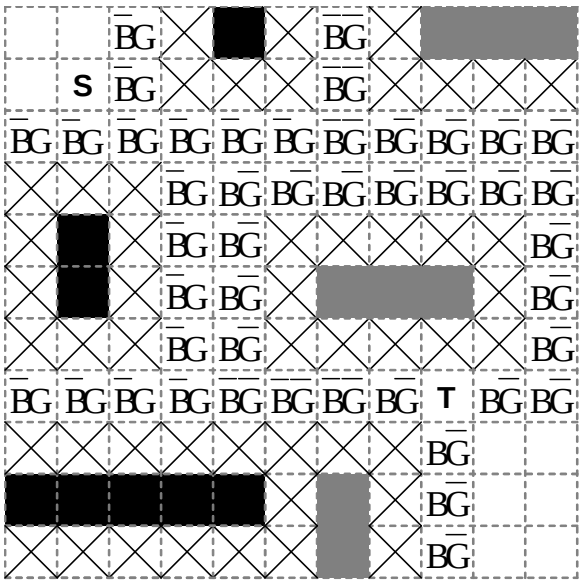


Figure 3.8: This example shows the coloring configuration of existing grids, right before routing a new net. The objects are layering in metal1 by default if not specially notated. The checked boxes are the blockages due to min_{sp}. Except BG, the state is shown in the grid.

friendly routing will be performed for net S-T. Assuming the path in Fig. 3.9 is picked due to possible *conflict and stitch free* wiring, the grids along the path will be colored through *coloring path* according to its grid states. In *coloring path*, a grid with $B\overline{G}$ and $\overline{B}G$ will be deterministically colored as BLACK and GRAY, and there will be a grid of conflict if the state is $\overline{BG}$. For a grid in BG state, which has the freedom to be in either BLACK or GRAY, it will be assigned to the nearest color along the path to minimize the number of stitches. After coloring a path, we will apply *coloring shadow* for it as well. Fig. 3.10 shows the colored new route with updated state map. These three

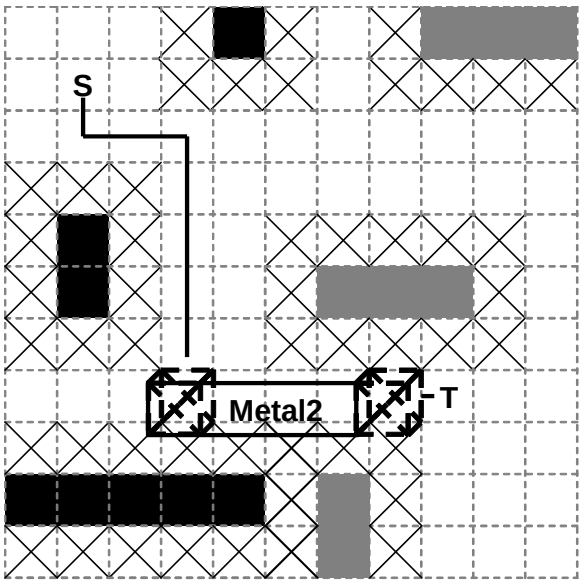


Figure 3.9: This example shows one possible DPL-friendly routing path for net S-T, which is conflict and stitch free.

steps, DPL-friendly routing, *coloring path*, and *coloring shadow*, repeat for all the nets.

However, the work [4] did not handle redundant via DPL compliance. As an example, Fig. 3.11 shows the routing result for new net S-T in Fig. 3.10 plus detailed metal2 information. Suppose via C has a single *feasible* redundant candidate C1, and A1-A2 and B1-B2 are the *feasible* ones for A and B respectively. Under the existing coloring assignment, as Fig. 3.12 illustrates, A2/B2 is not *DPL-feasible* because of the $\overline{BG}$ grid locations in metal1. A1 and C1 will also be in trouble as Fig. 3.12 shows. C1 is not *DPL-feasible* due to the coloring of M2 in the new S-T path. On the other side, A1 will have conflict

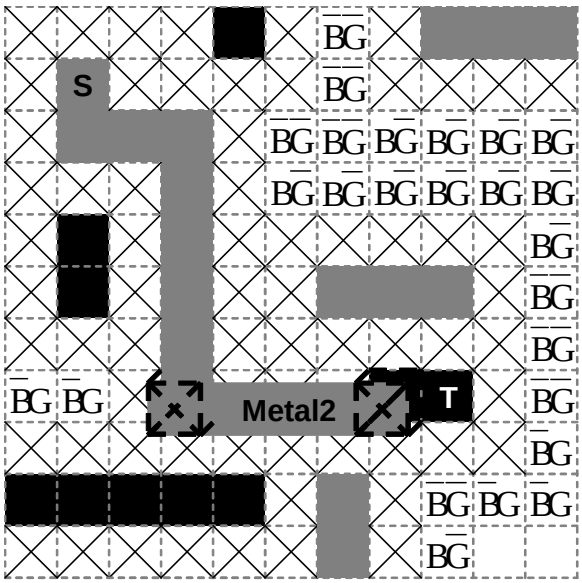


Figure 3.10: This example illustrates the step of *coloring shadow* in [4].

with the existing layout in metal2 if its configuration is *stitch-free extension*. In this case, both A and C are *DPL-dead*. If we simply flip the coloring of the entire M2 in the S-T routing path to save *DPL-feasible candidates* A1 and C1, it will make B *DPL-dead* in turn. As another solution, M2 might be split to make all vias *DPL-alive* but at a cost of an extra stitch. This example illustrates the fact that, their algorithm has difficulty producing high quality solution for both redundant via insertion and DPL.

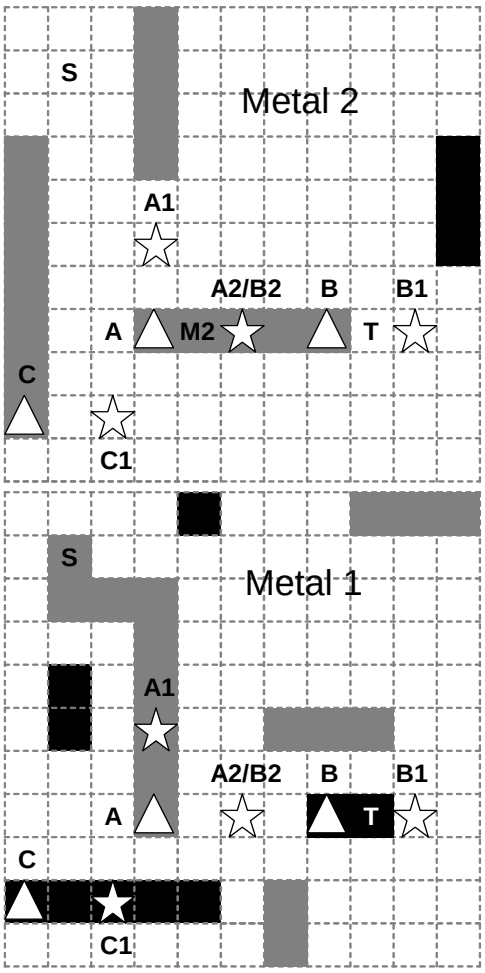


Figure 3.11: This example shows one layout, which the previous work [4] has difficulty handling redundant via DPL-compliance.

3.3 Post Routing DPL-Aware Redundant Via Insertion

In this section, we will first handle the redundant via DPL-compliance in post routing insertion stage. The layout and coloring will not be modified

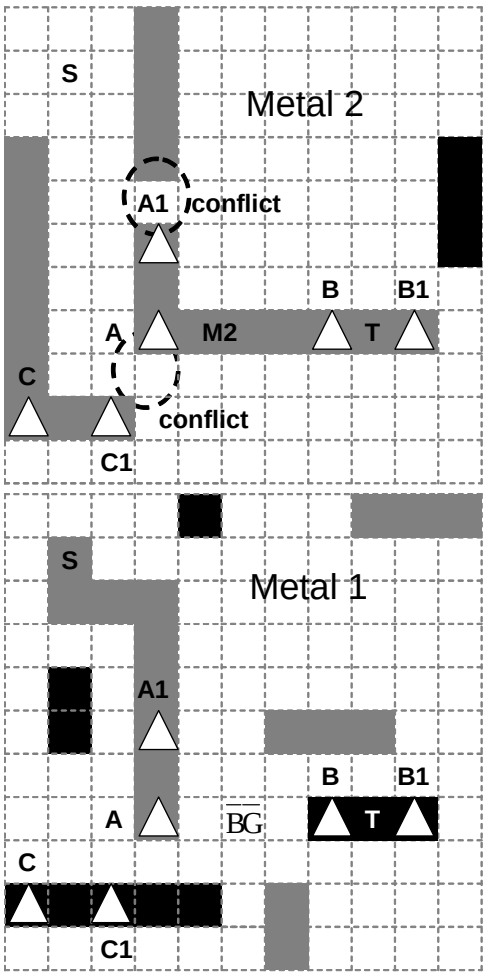


Figure 3.12: This figures shows the conflicts introduced by redundant via insertion, based on the previous work of [4]

to honor the existing optimization result, but the stitch will be applied to resolve coloring conflicts. We do not allow any coloring conflict resulting from insertion because it is not manufacturable. The formal problem statement is as follows.

Definition 6. post-routing DPL-aware redundant via insertion: *Without modifying the existing layout and coloring assignment, maximize the redundant via insertion rate while introducing as few as possible stitches and zero coloring conflicts.*

Inspired by the work [56], we formulate an integer linear programming algorithm to perform insertion and coloring at the same time. To enable simultaneous optimization, for each *feasible* redundant via candidate of the via shown in Fig. 3.13 (a), we first define four types of *potential configuration*s depending on the coloring of the extra metal. Fig. 3.13 (b)-(e) show the examples for the *feasible* candidate on the right side, where (b) is the *stitch-free extension* case while the other three all result in stitches by flipping extra metal colors in one or both layers. We disable the configuration that flips the coloring of existing routing wires to honor pre determined solution. That is to say, the configuration similar to Fig. 3.13 (f) will be forbidden because the coloring of existing metal2 is flipped. In our redundant via solutions, only the above four *potential configuration*s are included for maintaining the uniformity of extra metal coloring. Other configurations with some stitches inserted in the middle of the extra metal can be easily extended.

Our post-routing DPL aware redundant via insertion algorithm is based on integer linear programming, which is inspired by the work [56]. The notations are listed in Table 4.1. The formulation is as follows:

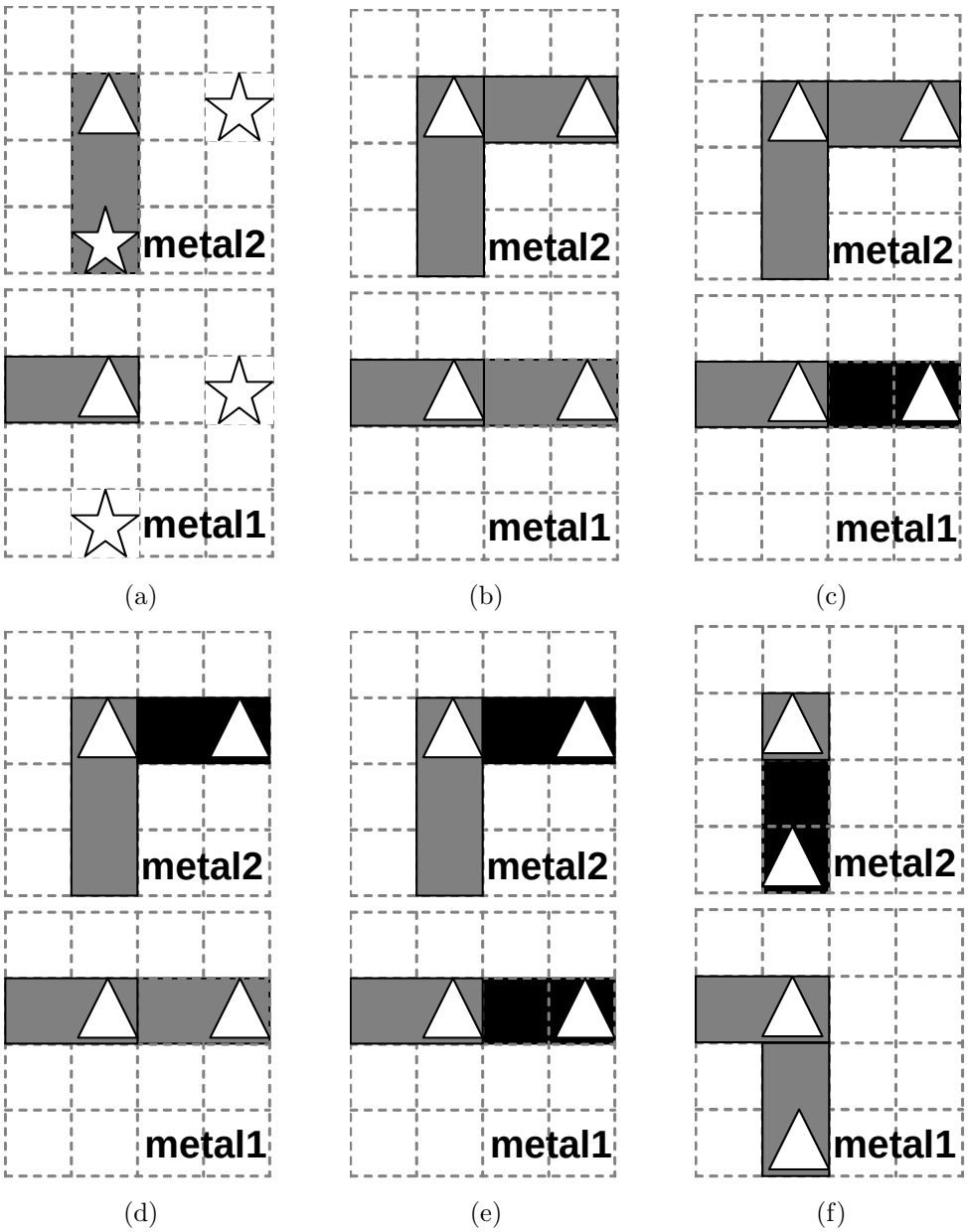


Figure 3.13: This example illustrates the *potential configuration*s for redundant via insertion in DPL.

Table 3.1: Notation

v_i	the ith via with at least one feasible redundant via locations
V	the set of all v_i
f_{ij}	the jth feasible redundant via location of v_i
F	the set of all f_{ij}
r_{ij}	binary variable. $r_{ij} = 1$ if f_{ij} is chosen
R	the set of all r_{ij}
(f_{ij}, f_{mn})	f_{ij}, f_{mn} are a pair of feasible redundant candidates for different vias v_i and v_m $(i \neq m)$, which can not be inserted simultaneously due to *external conflict*
EF	the set of all (f_{ij}, f_{mn})
p_{ij}^k	the kth *potential configuration* of f_{ij}
P	the set of all p_{ij}^k
s_{ij}^k	number of stitches p_{ij}^k will introduce
o_{ij}^k	binary variable. $o_{ij}^k = 1$ if p_{ij}^k is chosen
(p_{ij}^k, p_{mn}^l)	a pair of p_{ij}^k, p_{mn}^l for different vias v_i and v_m $(i \neq m)$ will cause coloring conflicts to each other
ECP	the set of all (p_{ij}^k, p_{mn}^l)
CP	the set of all p_{ij}^k, which will cause coloring conflict with existing layout and coloring assignment
B_{ij}	the set of best *potential configuration*s for f_{ij} which have minimum number of stitches and do not cause coloring conflict with existing layout and coloring assignment
NFP	the set of all f_{ij} whose p_{ij}^k is NOT involved in any ECP

$maximize$:

$$\sum_{\forall r_{ij} \in R} r_{ij} - \lambda \sum_{\forall p_{ij}^k \in P} s_{ij}^k \cdot o_{ij}^k \tag{3.1}$$

where

$$\sum_{f_{ij} of v_i} r_{ij} \leq 1 \quad \forall v_i \in V \tag{3.2}$$

$$r_{ij} + r_{mn} \leq 1 \quad \forall (f_{ij}, f_{mn}) \in EF \tag{3.3}$$

$$\sum_{p_{ij}^k of f_{ij}} o_{ij}^k = r_{ij} \quad \forall f_{ij} \in F \tag{3.4}$$

$$o_{ij}^k = 0 \quad \forall p_{ij}^k \in CP \tag{3.5}$$

$$o_{ij}^k + o_{mn}^l \leq 1 \quad \forall (p_{ij}^k, p_{mn}^l) \in ECP \tag{3.6}$$

$$o_{ij}^k = 0 \quad \forall p_{ij}^k \notin B_{ij} \cup CP, \quad \forall f_{ij} \in NFP \tag{3.7}$$

The objective function (3.1) is to maximize the insertion rate while minimizing the number of stitches the *potentical configuration* introduces. λ is used to tune the relative importance between stitches with respect to insertion rate.

Constraint (3.2) implies that at most one *feasible* candidate will be picked for each via. The *external conflict* is avoided by applying Constraint (3.3). Constraint (3.4) guarantees only one *potential configuration* will be selected for each inserted redundant via location. Constraint (3.5) and (3.6) are used to prevent the coloring conflict due to insertion. Constraint (3.5) disables the *potential configuration* which will cause coloring conflict with the existing layout, and Constraints (3.6) is used to avoid the situation when a pair of redundant vias cause coloring conflict between each other. Constraint (3.7) prunes the sub optimal *potential configuration*s of certain *feasible* candidates. If a candidate can not have conflict with existing layout or other redundant vias, we will not pick its *potential configuration*s which do not have minimum number of stitches. We also adopt speed-up techniques similar to the work [56].

3.4 DPL-friendly Detailed Routing with Redundant Via Consideration

The post-routing DPL-aware redundant via insertion may not be enough for DPL-unfriendly designs. If we only do DPL-friendly routing as [4], we might end up with a lot of *DPL-dead* vias, which do not have *conflict and stitch free* redundant via solutions. Therefore, it is in high demand to consider redundant via and DPL together during routing. Our formal problem statement is as follows.

Definition 7. DPL-friendly Detailed Routing with Redundant Via Consideration: *Given an input netlist, perform simultaneous routing and coloring to minimize the number of* DPL-dead *vias while maintaining highly decomposable wiring path and other design objectives.*

In our algorithm, we will first present *via color shadow* in Section 3.4.1 to remove *DPL-dead* via during routing. Moreover, for larger optimization space and better solution quality, we propose an *equivalent transformation* in Section 3.4.2. Finally, the entire detailed routing algorithm will be presented in Section 3.4.3.

3.4.1 Via Color Shadow

Eliminating *DPL-dead* vias during routing is much more difficult than dealing with *dead* ones as in [65]. Besides min_{sp}, the coloring of the layout has to be taken into account as well. A via could have *DPL-feasible* redundant

candidate in one coloring assignment but is *DPL-dead* in another. Therefore, we need to have different costs to predict and penalize the *DPL-dead* vias, according to various coloring configurations. Meanwhile, similar to removing *dead* via, when routing a new net, we should prevent a *DPL-alive* via of existing nets from becoming *DPL-dead* as well as avoiding generating *DPL-dead* via in itself. In this subsection, we will propose *via color shadow* to resolve above issues and enable during-routing *DPL-dead* vias elimination.

Algorithm 2 via color shadow

Require: a colored path p
Ensure: *DPL-dead* via avoidance cost assignment
1: **for** each *DPL-alive via* v in p **do**
2: **for** each *DPL-feasible* candidate rvc of v **do**
3: **for** each $g \in Gt$, which Gt denotes all the available nearby routing grids within min_{dp} from rVc **do**
4: **if** when g is $BLACK$, it will have conflict with the *stitch-free extension* of rvc **then**
5: $g.Vcost(B)$ += f_1
6: **end if**
7: Similar when g is $GRAY$
8: **end for**
9: **end for**
10: **end for**
11: **for** each $v \in V$, which V denotes all the available via locations within min_{dp} from p **do**
12: **if** v will be *DPL-dead* when the coloring of its up and down grids ($v.ch$, $v.cl$) is $(BLACK, BLACK)$ **then**
13: $v.Vcost(B, B)$ += C_2
14: **end if**
15: Similar when ($v.ch$, $v.cl$) has other coloring configurations in Fig. 3.14
16: **end for**

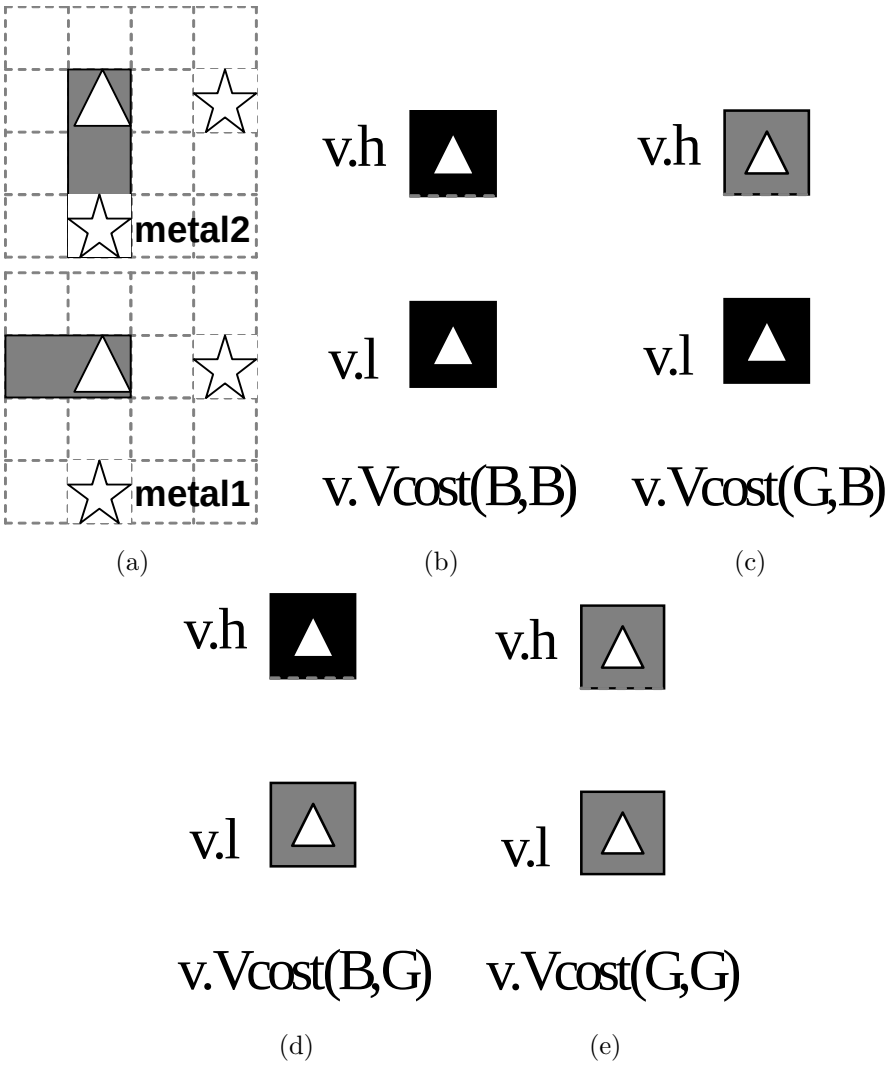


Figure 3.14: This example illustrates the four coloring configurations of via v and its costs v.$Vcost(v.ch, v.cl)$. $v.h$ and $v.l$ denote the up and down grids v links to respectively.

Our algorithm is presented in Algorithm 3. First of all, we should avoid hurting the *DPL-alive* vias in the existing layout when routing and coloring the new nets. We propose a penalty pair ($g.Vcost(B)$, $g.Vcost(G)$) for each

grid g in line 1-10. The value of the pair reflects the cost of coloring the grid as BLACK and GRAY respectively. The cost will be higher if more *DPL-feasible* candidates of the existing vias will be killed due to the grid state. Meanwhile in line 11-16 , we would like to avoid generating a *DPL-dead* via when routing a new net. To determine whether a potential via v is *DPL-dead*, we need to know the coloring of the grids it links in both layers. Suppose the higher and the lower grids which v links are $v.h$ and $v.l$ respectively, and their colors are $v.ch$ and $v.cl$, we introduce four costs $v.Vcost(v.ch, v.cl)$ for each possible via v in the routing graph. These are to account for the potential *DPL-dead* cases, when ch and cl can be either BLACK(B) or GRAY(G).

Our algorithm is presented in Algorithm 3. First of all, we should avoid hurting the *DPL-alive* vias in existing layout when routing and coloring the new nets. We propose a penalty pair ($g.Vcost(B)$, $g.Vcost(G)$) for each grid g. The value of the pair reflects the cost of coloring the grid as BLACK and GRAY respectively. The cost will be higher if more *DPL-feasible* candidates of existing vias will be killed due to the grid state. In line 1-9, after we find a path and assign its coloring, we will update $g.Vcost(B)/g.Vcost(G)$ of related grids as described. The cost function f_1 is defined as $\frac{C_1}{n}$ where n is the number of *DPL-feasible* candidates each *DPL-alive* via has and C_1 is a constant. The reason we do not explicitly modify the state of the grid is that, killing a *DPL-feasible* candidate does not necessarily result in a *DPL-dead* via, there could be other candidates available. Alternating the grid state results in overwhelming constraints and decreases the routing flexibility.

Algorithm 3 via color shadow

Require: a colored path p
Ensure: *DPL-dead* via avoidance cost assignment
1: **for** each *DPL-alive via* v in p **do**
2: **for** each *DPL-feasible* candidate rvc of v **do**
3: **for** each $g \in Gt$, which Gt denotes all the available routing grids within min_{dp} from rvc **do**
4: **if** when g is $BLACK$, it will have conflict with the *stitch-free extension* of rvc **then**
5: $g.Vcost(B)$ += f_1
6: **end if**
7: Similar when g is $GRAY$
8: **end for**
9: **end for**
10: **end for**
11: **for** each $v \in V$, which V denotes all the available via locations within min_{dp} from p **do**
12: **if** v will be *DPL-dead* when the coloring ($v.ch$, $v.cl$) of its up and down grids is $(BLACK, BLACK)$ **then**
13: $v.Vcost(B, B)$ += C_2
14: **end if**
15: Similar when ($v.ch$, $v.cl$) has other coloring configurations
16: **end for**

Meanwhile, we would like to avoid generating a *DPL-dead* via when routing a new net. To determine whether a potential via v is *DPL-dead*, we need to know the coloring for the grids it links in both layers. Suppose the higher and lower grids which v links to are $v.h$ and $v.l$ respectively as in Fig 3.14(a), and their colors are $v.ch$ and $v.cl$. We introduce four costs $v.Vcost(v.ch, v.cl)$ for each possible via v in the routing grid. These are to account for the potential *DPL-dead* cases, when ch and cl can be either BLACK(B) or GRAY(G), as Fig 3.14(b)-(e) indicates. In line 10-14 of Al-

gorithm 3, we update $v.Vcost(v.ch, v.cl)$ of any possible via location within min_{dp} from this path. If its certain configuration in Fig 3.14 is *DPL-dead*, the corresponding cost will be increased by a constant value C_2.

The proposed *DPL-dead* via avoidance costs can be penalized as in Algorithm 4. The penalty $g.Vcost$(B)/$g.Vcost$(G) will be applied to the grids with a state except BG in a straightforward way as described in Line 1-3. $B\overline{G}/\overline{B}G$ grid will be punished by $g.Vcost(B)/g.Vcost(G)$, while $\overline{BG}$'s cost is the summation of $g.Vcost(B)$ and $g.Vcost(G)$ to completely keep away the redundant via from coloring conflicts. Similarly, when routing through one via v, we can penalize $v.Vcost(v.ch, v.cl)$ according to the states of its $v.h$ and $v.l$ grids as shown in Line 4-12. However, it is tricky to deal with these costs for BG grid because of a chicken and egg problem. BG can be assigned either BLACK or GRAY. If we do not consider its exact coloring during routing as in [4], we have no idea which one out of these proposed costs should be penalized. We can certainly apply some estimated cost for BG grid but the solution quality will be deteriorated. This motivates us to perform simultaneous coloring and routing for BG grids.

3.4.2 Equivalent Transformation

In the previous work [4], when a grid is BG, its exact coloring will not be considered during routing. It will be picked as either BLACK or GRAY greedily after the path is determined. This narrows the optimization space. Moreover, as discussed above, we also need detailed coloring information for

Algorithm 4 penalize *DPL-dead* via avoidance cost

Require: the current grid x along with its adjacent grid d in the searching direction, and routing cost *cost*

1: **if** $d.state == B\overline{G}$ or $\overline{B}G$ or $\overline{BG}$ **then**
2: $cost$ += $d.Vcost(B)$ or $d.Vcost(G)$ or $(d.Vcost(B)+d.Vcost(G))$
3: **end if**
4: **if** x and d are not in the same layer, which forms a via v **then**
5: identify higher and lower grids $v.h$ and $v.l$ from x and d
6: **if** $(v.h).state == B\overline{G}/\overline{B}G$ and $(v.l).state == B\overline{G}/\overline{B}G$ **then**
7: $cost$+=$v.Vcost((v.h).state, (v.l).state)$
8: **end if**
9: **if** $(v.h).state == \overline{BG}$ or $(v.l).state == \overline{BG}$ **then**
10: $cost$+=$(v.Vcost(B,B)+v.Vcost(B,G)$ $+v.Vcost(G,B)+v.Vcost(B,B))$
11: **end if**
12: **end if**

BG grid on the fly to better eliminate *DPL-dead* vias. Therefore, we would like to have certain *look ahead* capability to distinguish different situations, when the BG grid is in BLACK or GRAY.

Our idea is to replace each BG grid with two equivalent grids, as Fig. 3.15 illustrates, which are *brothers* to each other. These grids have the state of $B\overline{G}$ and $\overline{B}G$ respectively, and are used to track the different cases when this original BG is colored as BLACK/GRAY. During routing, we will apply penalty on the equivalent grids rather than the original BG grid.

The underlying routing graph is changed accordingly as well, as shown in Fig. 3.15 (b). Both the new grids link to the grids the BG one connects. Because the two equivalent grids are physically same, in any found path, only

one of them should be there. To ensure this, first of all, in the updated routing graph, there should be no routing edge between the equivalent grids.

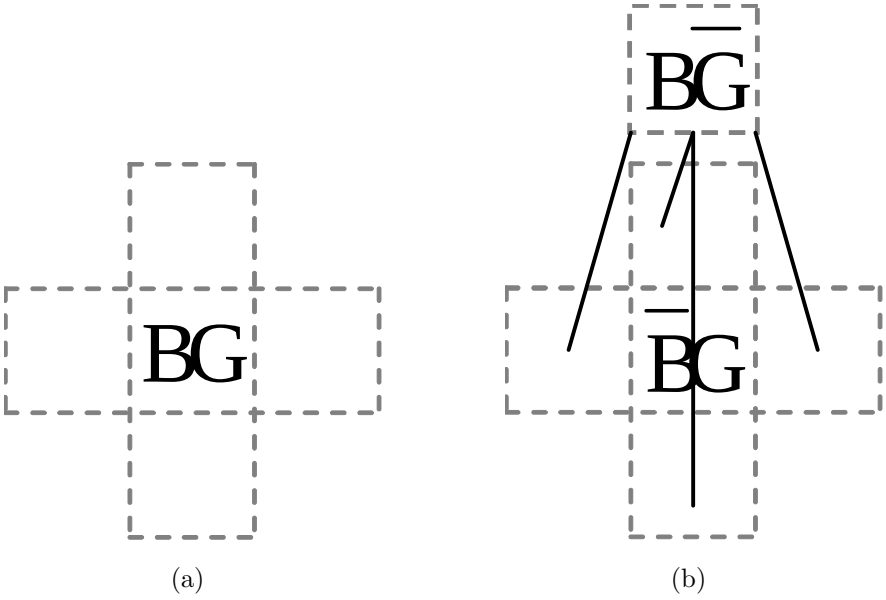


Figure 3.15: This example illustrates equivalent transformation. The solid line means there is a routing edge available between two grids.

Our idea is to replace each BG grid with two equivalent grids, as Fig. 3.15 illustrates, which are *brothers* to each other. These two grids have the state of $B\overline{G}$ and $\overline{B}G$, and are used to track the different cases when this original BG is colored as BLACK and GRAY respectively. During routing, we will apply penalty on the equivalent grids rather than the original BG grid.

The underlying routing graph is changed accordingly as well, as shown in Fig. 3.15 (b). Both the new grids link to the grids the BG one connects. Because the two equivalent grids are physically same, in any found path, only

one of them should be there. To ensure this, first of all, in the updated routing graph, there should be no routing edge between the equivalent grids.

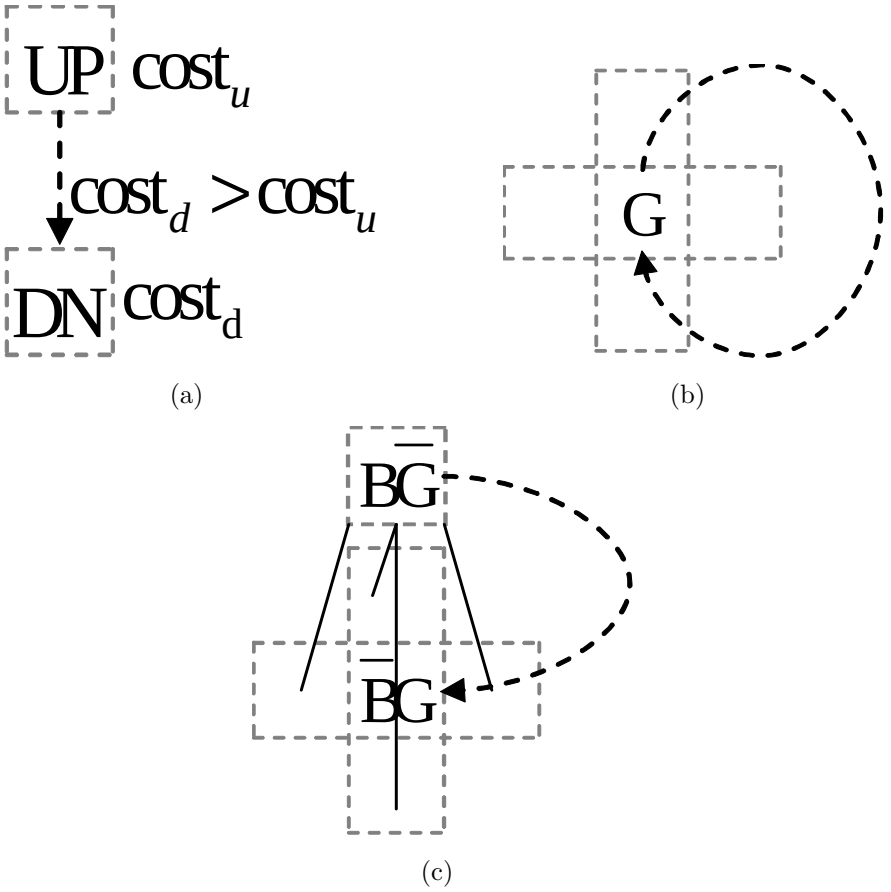


Figure 3.16: This example illustrate the illegal loop problem. The bold dash line cycle is the path going through a set of grids.

However, there will be an *illegal loop problem* coming from above *equivalent transformation*. During the path propagation we may still generate a

route linking the equivalent grids through a set of other grids as Fig. 3.16 (c). This configuration is illegal because the selection of $B\overline{G}$ and $\overline{B}G$ is exclusive as we just discuss. Unfortunately, this situation does happen in our routing algorithm. First, as Fig. 3.16 (a) indicates, because the penalty for each routing edge is positive in our case, for any wiring path, $cost_d$ associated with downstream grid DN must be higher than $cost_u$ of its upstream grid UP. For normal grid, the self loop Fig. 3.16 (b) can be avoided. The reason is that suppose this happens, some grid G must be the downstream grid of itself. It is a contradiction that the cost of G is higher than the cost of itself, according to above discussion. However, in Fig. 3.16 (c), the illegal loop occurs in proposed transformation because the equivalent grids account for different coloring cases and their costs do not have to be the same value. Suppose the current cost in $\overline{B}G$ is much higher than $B\overline{G}$, without taking care of it, the maze routing could produce a path with strictly increasing cost from $B\overline{G}$ to $\overline{B}G$.

Our solution for this problem is simple but effective. Whenever we would like to update a cost of an equivalent grid during routing, we will trace back to see whether its ancestors contain the *brother* one. If so, it returns true and this updating request will not be executed.

3.4.3 Detailed Routing

Based on the two techniques discussed above, we will present our DPL-friendly detailed routing with redundant via consideration. The algorithm is shown in Algorithm 5. When we start to route a net, the equivalent transfor-

Algorithm 5 DPL-Friendly Detailed Routing with Redundant Via Consideration

Require: current state map and unrouted net n with source s and target t

1: perform *equivalent transformation* for s and t if applicable
2: a priority queue $Q = s$ or its equivalent grids
3: **while** Q is not empty **do**
4: x = dequeue from Q
5: **if** if $x == t$ or its equivalent grids **then**
6: break
7: **end if**
8: perform *equivalent transformation* for adjacent grids for x if applicable
9: **for** each adjacent grid d of x **do**
10: $cost = (x.cost+1+Acost)$//wirelength and A* search cost
11: **if** $(d.state, x.state) == (B\overline{G}, \overline{B}G)$ or $(\overline{B}G, B\overline{G})$ **then**
12: $cost$ += α // to discourage a stitch on routing
13: **end if**
14: **if** $d.state == \overline{BG}$ **then**
15: $cost$ += β // to discourage decomposition failures
16: **end if**
17: **if** x and d not on the same layer **then**
18: $cost$ += γ// to discourage the via
19: **end if**
20: *penalize DPL-dead via avoidance cost*
21: **if** $d.cost > cost$ and *illegal loop* checking returns false **then**
22: $d.cost = cost$; $d.prev = x$
23: **end if**
24: **end for**
25: **end while**
26: *coloring path* for found route p
27: *reverse transformation*
28: *coloring shadow* and *via cost shadow*

mation is first performed if the state of source/target grid is BG, as in line 1. In line 8, during maze routing, we will also perform *equivalent transformation* if any adjacent grid of current grid x has BG state. Therefore, the

grid can only possibly be either $B\overline{G}$, $\overline{B}G$ and $\overline{BG}$ in path propagation phase. The following lines 10-19 are the DPL-aware cost and other objectives as in previous work [4]. In line 20, we will apply the DPL-dead via avoidance cost as described in Algorithm 4. In line 21, when we try to update the cost of a grid, we have to make sure illegal loop problem will not happen, as discussed in 3.4.2. Line 26 is the *coloring path*, same as [4] but the coloring of every grid can be deterministically assigned because no BG grid is here. Line 27 is used to clear up the transformation and restore the original routing graph. If neither of the equivalent grids is on the found path, we will simply map back from Fig. 3.15 (b) to Fig. 3.15 (a). Otherwise, the one the routing passes will be kept, and the *brother* grid along with its connecting routing edge will be released. *Coloring shadow* and *via cost shadow* are performed next in Line 28 to update the grid states and *DPL-dead* via avoidance cost.

3.5 Experimental Result

In our benchmarks, four 65nm ASIC designs are scaled down to the 32nm technology node for experiments. The detailed information for each test case is shown in Table 5.1. The first column denotes the circuit name. "nets" shows the number of nets and "area" is the chip area in terms of um^2. "grid size" shows the number of rows by the number of columns in terms of routing grid.

We implement our algorithm in C++ and test on Intel Core 3.0GHz Linux machine with 32G RAM. Glpk [54] is applied as the solver for inte-

Table 3.2: The test cases.

ckt	nets	area	grid size
C1	200	0.37k	600*600
C2	500	1.02k	1000*1000
C3	1000	2.01k	1400*1400
C4	2000	4.10k	2000*2000

ger linear programming. Our *post-routing DPL-aware insertion* is denoted as **POSTDPL**. We study the different λ settings in ILP formulation shown in Fig 3.17. As λ is decreased, we are able to achieve higher insertion rate by allowing more stitches. After certain value, it has little effect. In our work, we set λ as 0.4. On the other side, our *DPL-friendly routing with redundant via consideration* is referred as **DPRRV**. The parameters α, β and γ are set as the same as the previous work [4]. α and γ are 9 and 6 respectively while $\beta >> 10$. For C_1 and C_2 in *via cost shadow*, we also conduct similar parameter study as shown in Fig 3.18. Increasing both parameters can effectively reduce the number of *DPL-dead* vias until reaching certain saturating values. We set both as 100.

For comparative reason, we also implement a DPL-aware optimization flow without considering redundant via compliance **DPR+POST**. In **DPR**, *DPL-friendly routing* [4] is first performed. For fair comparison, we also apply the techniques in [65] to reduce the number of *dead* vias. The reason is that in our routing algorithm **DPRRV**, *via cost shadow* can eliminate *dead* vias as a side effect. The *DPL-dead* via it primarily removes is a super set of *dead*

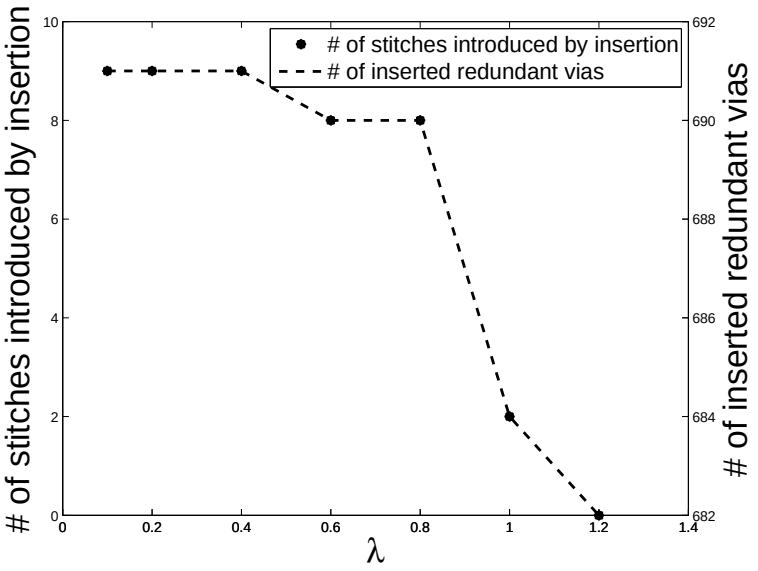


Figure 3.17: The performance of our algorithm with different λ values.

via. In the following post-routing via insertion **POST**, we first apply [56] to maximize the redundant via insertion rate. For each inserted one, we will then try to select the best *potential configuration* to eliminate the resulting coloring conflict with minimum stitches introduced. After this, if there are still coloring conflicts due to insertion, we will remove the corresponding redundant vias.

Table 3.3-5.3 presents our experiment results. For all the tables, "WL" is the total wirelength of metal1 and metal2 with the unit *um*. "CPU" is the total runtime by second. "via" and "DPDV" are the number of vias and *DPL-dead* vias respectively, and "DPDV%" is the ratio of "DPDV' over "via". "ST(v)" is the number of additional stitches caused by post-routing via

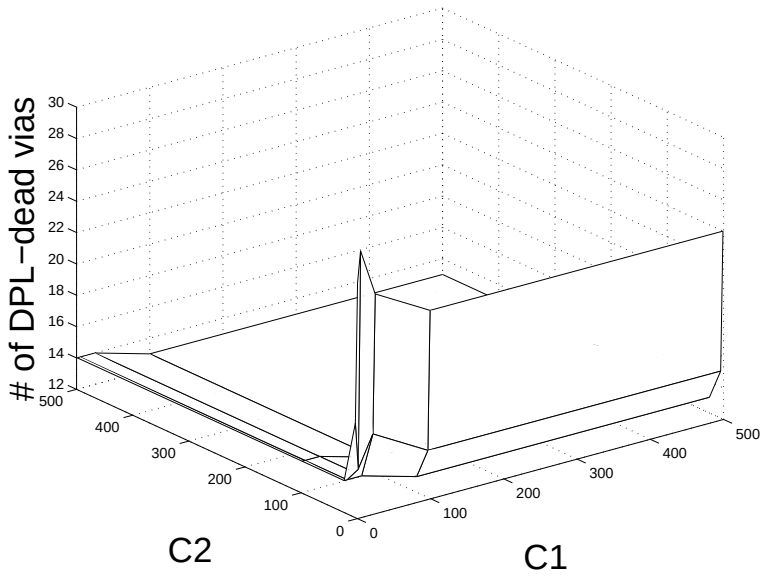


Figure 3.18: The performance of our algorithm with different C_1 and C_2 values.

insertion. "ST(t)" is the total number of stitches in the final layout, including both routing wires and redundant vias. "rv" shows the number of inserted redundant vias without causing any coloring conflict. "rv%" is the insertion rate, which is the ratio of "rv" over "via". The "total" row shows the sum or average for all four test cases. "ratio" row is calculated with respect to corresponding "total" item in Table 3.3. All of our experiments achieve 100% routability and zero coloring conflicts.

3.5.1 The importance of redundant via consideration in DPL

Table 3.3 shows the result of DPL-aware optimization flow without considering redundant via compliance, **DPR+POST**. **DPR** only focuses on obtaining zero coloring conflicts and minimizing the number of stitches for

routing wires. However, as we find out, it produces many *DPL-dead* vias, i.e, there are a large portion of vias, averagely 6.9%, which do not have *conflict and stitch free* redundant via solutions. Therefore, if we simply apply **POST** to insert redundant vias, our insertion rate is only 69.4% averagely when no resulting coloring conflict is allowed. These experimental results demonstrate the strong demand for considering redundant via insertion and DPL together.

Table 3.3: **DPR+POST**: The DPL flow without considering redundant via

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	12	313	26	8.3	4	41	209	66.8
C2	1.67	64	757	74	9.8	3	86	519	68.6
C3	4.06	184	1004	64	6.3	13	293	696	69.3
C4	8.95	680	2062	123	6.0	21	565	1447	70.2
total	15.35	940	4136	287	6.9	41	985	2871	69.4
ratio	1	1	1	1	1	1	1	1	1

3.5.2 Post-routing DPL-aware redundant via insertion

To find better DPL-friendly redundant via solution, we first replace **POST** by our approach **POSTDPL** in post-routing phase after applying **DPR**. This is denoted as **DPR+ POSTDPL** and the results are shown in Table 5.2. By performing simultaneous insertion and coloring, **POSTDPL** can increase the insertion rate to 94.1% averagely without causing any coloring conflict. Compared to **DPR+POST**, although we introduce a few more stitches due to insertion, the number is relative small compared to the total

number of stitches in the final layout. Moreover, the runtime overhead is very little. On the other side, because we also apply **DPR** without redundant via DPL-compliance in this experiment, a significant number of *DPL-dead* vias are still there. This implies a noticeable improvement space. It is highly desirable to eliminate these unfriendly vias from design side.

Table 3.4: **DPR+POSTDPL**: Result for post-routing DPL-awareness insertion

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	12	313	26	8.3	3	40	288	92.0
C2	1.67	66	757	74	9.8	9	92	691	91.3
C3	4.06	188	1004	64	6.3	19	299	950	94.6
C4	8.95	683	2062	123	6.0	29	573	1962	95.2
total	15.35	949	4136	287	6.9	60	1004	3891	94.1
ratio	1	1.01	1	1	1	1.46	1.02	1.36	1.36

3.5.3 DPL-friendly routing with redundant via consideration

To eliminate those vias which do not have *DPL-feasible* redundant via solution during design, we will further replace **DPR** by **DPRRV** before applying **POSTDPL**. This is referred as **DPRRV+POSTDPL**. As expected, our approach achieves averagely 89% less *DPL-dead* vias. The insertion rate is improved to 99.3% averagely, with zero conflicts and 73% less stitches introduced. Moreover, we are able to reduce the number of total stitches in final layout by 17%, including both routing wires and redundant vias. The reason is that by doing *equivalent transformation*, the coloring of BG grid is planned during our detailed routing. It has higher possibility to conduct global opti-

mization. Because of the same reason, we are also able to reduce the number of vias by 9%, while **DPR** has to use more vias to resolve potential coloring conflicts. As the last mention, all these improvements in our approach are obtained with little overhead on wirelength and runtime.

Table 3.5: **DPRRV+POSTDPL**: Result for DPL-friendly Post-Routing and detailed routing with redundant via consideration

ckt	WL	CPU	via	DPDV	DPDV%	ST(v)	ST(t)	rv	rv%
C1	0.67	17	283	6	2.1	2	34	279	98.6
C2	1.68	90	654	13	2.0	4	61	640	97.9
C3	4.10	277	930	3	0.3	2	240	928	99.8
C4	9.03	750	1915	9	0.5	3	482	1908	99.6
total	15.48	1134	3782	31	0.8	11	817	3755	99.3
ratio	1.008	1.21	0.91	0.11	0.12	0.27	0.83	NA	1.43

3.6 Summary

In this chapter, I propose the first work to consider DPL and redundant via together. To make design manufacturable in DPL, we should not insert a redundant via if it will cause a coloring conflict. I have developed two algorithms, *post-routing DPL-aware insertion* and *DPL-friendly routing with redundant via consideration* to taken into account redundant via DPL-compliance in a construct-by-correction and correct-by-construction manner respectively. Experimental results show that, compared to a DPL-aware optimization without redundant via consideration, redundant via insertion rate is improved by 43% while still achieving zero coloring conflicts. Moreover, we can even reduce the number of vias and stitches by 8% and 17% respectively.

Chapter 4

Wire Spreading Enhanced Double Patterning

4.1 Introduction

In Double Patterning Lithography (DPL), conflict and stitch minimization are two main challenges. Many researches focus on post-routing mask decomposition. A novel flow is proposed in [53] to optimize splitting locations with ILP. Xu et al. [66] present an efficient graph reduction based algorithm for stitch minimization, and Yang et al. [67] propose a fast partition-based approach. In these works, conflicts are eliminated in a greedy way. To enable simultaneous conflict and stitch minimization, ILP is adopted in [35, 68] with different feature pre-slicing techniques. Xu et al. [69] propose a matching based decomposer that handles the same optimization problem as [35, 68].

There are also several DPL-aware optimization works from design side. Cho et al. [4] propose a correct-by-construction DPL-friendly routing with built-in layout decomposer. The idea is extended by [70] with enhancement of lazy evaluation and with-in net optimization. In [36], the DPL awareness and redundant via insertion are considered together during routing. Hsu at al. [71] propose a simultaneous layout migration and decomposition for standard cell design, which aims to minimize stitch number and layout area together. Be-

cause the spacing between the features are considered dynamically during coloring, their approach suffers from run time overhead, which is not suitable for large-scale layout modification.

In this chapter, we present WIre Spreading enhanced Decomposition of Masks in double patterning lithography(WISDOM). The chip area is fixed in our work. After initial Decomposition Graph (DG) construction, we create a set of wire spreading candidates. The DG is updated then to model layout decomposition problem together with these potential WSCs. Our main technical contributions are as follows:

1. We develop an integer linear programming formulation from DG to simultaneously minimize the number of conflicts, stitches and amount of layout perturbation.

2. We prove that ILP formulae only needs to be conducted on a subgraph of DG, which is the union of all the odd-cycles. We also develop an efficient algorithm to calculate such subgraph in a polynomial time.

3. We propose coloring-independent group computing to reduce ILP problem size without losing optimality. We also develop suboptimal solution punning technique to simplify each coloring-independent group by pre computing optimal solutions for underlying substructures.

The rest of the chapter is organized as follows. Section 4.2 provides the preliminaries and problem formulation. The detailed algorithm is described

in Section 4.3, and the speed-up techniques are discussed in Section 4.4. Section 4.5 presents the experiment results and Section 4.6 concludes this chapter.

4.2 Preliminary and Formulation

4.2.1 Wire Spreading for Decomposability

Our key idea of performing wire spreading for decomposition of masks is to push layout segments away for more flexible coloring. This helps reducing the number of conflicts and stitches.

Given a routed design, wire spreading can be used for eliminating unresolvable coloring conflicts. Fig. 4.1 (a) shows a three-way conflict cycle between features A-B-C, where any two of them are within minimum coloring space. Moreover, no stitches can be inserted while satisfying minimum overlapping margin requirement. In consequence, there is no way to produce conflict-free solution by mask decomposition algorithm alone. However, if part of feature C is spread to break the coloring constraint between A and C as Fig. 4.1 (b), we can easily resolve this problem. For more restricted situation, as Fig. 4.1 (c) illustrates, it is not possible to completely push C away from feature A or B beyond min_{cs}, due to surrounding fixed layout objects. In such case, we could still spread polygon C for creating a legal splitting location, as indicated by Fig. 4.1 (d).

It is also possible to spread wire to reduce the number of stitches. For the example in Fig. 4.2 (a), initially, feature C is not splittable, and two stitches are required on A and D for resolving conflicts. If we spread the route C as

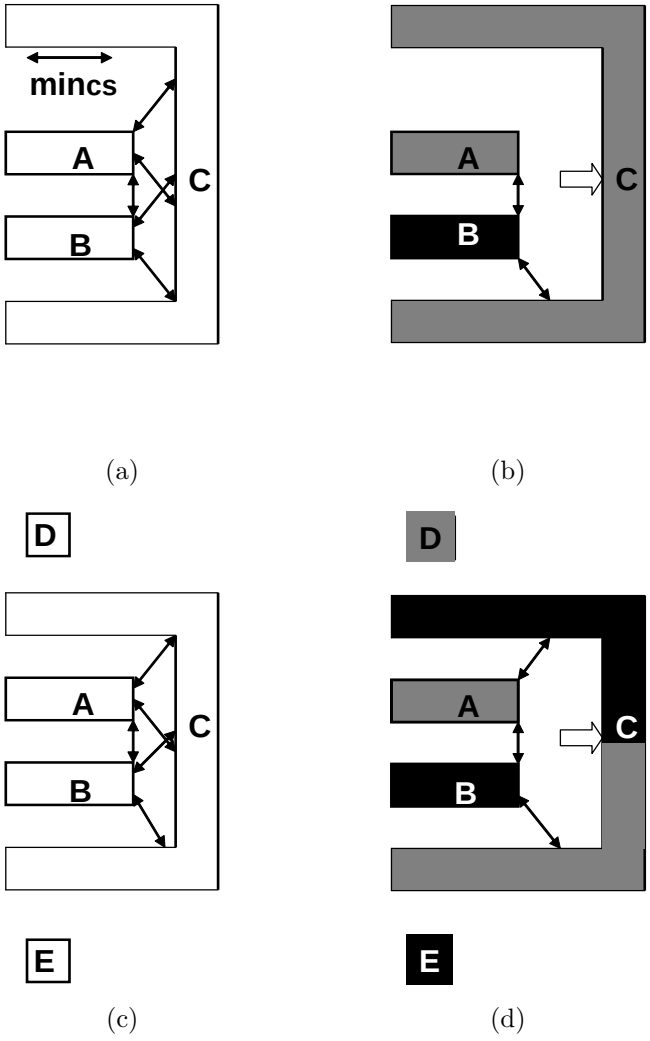
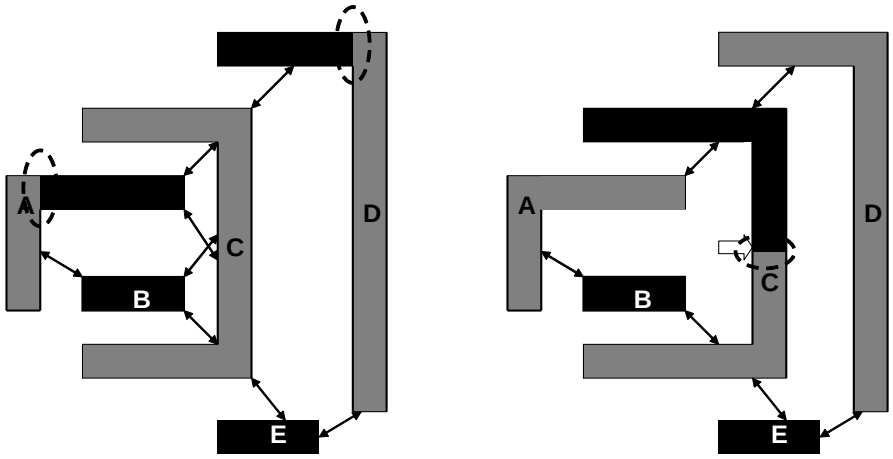


Figure 4.1: Wire spreading to eliminate conflict. Assume only feature C can be moved.

Fig. 4.2 (b), only one stitch is needed.

As it can be seen from Section 4.2.1, with help of wire spreading, the



(a) (b)

Figure 4.2: Wire spreading to reduce stitch. Assume only feature C can be moved.

solution space of DPL decomposition can be extraordinarily high. On the other hand, as nanometer designs are mostly grid-based, that restricts wire spreading to its nearby discrete routing grids. Therefore, we will preprocess the initial decomposition graph and generate a library of Wire Spreading Candidates (WSC)s to improve decomposability (more detailed description to be presented in Section 3.2). Mask assignment is performed together with these candidates as options. As examples, Fig. 4.1 (b)/(d) and Fig. 4.2 (a) are simple WSCs for Fig. 4.1 (a)/(c) and Fig. 4.2 (f), respectively.

With this library of WSCs, our optimization problem is formally defined as follows:

Problem Formulation: Given a layout, perform mask decomposition

with pre-computed WSCs as design modification options. The goal is to minimize the number of conflicts and stitches, while introducing as less layout perturbation as possible.

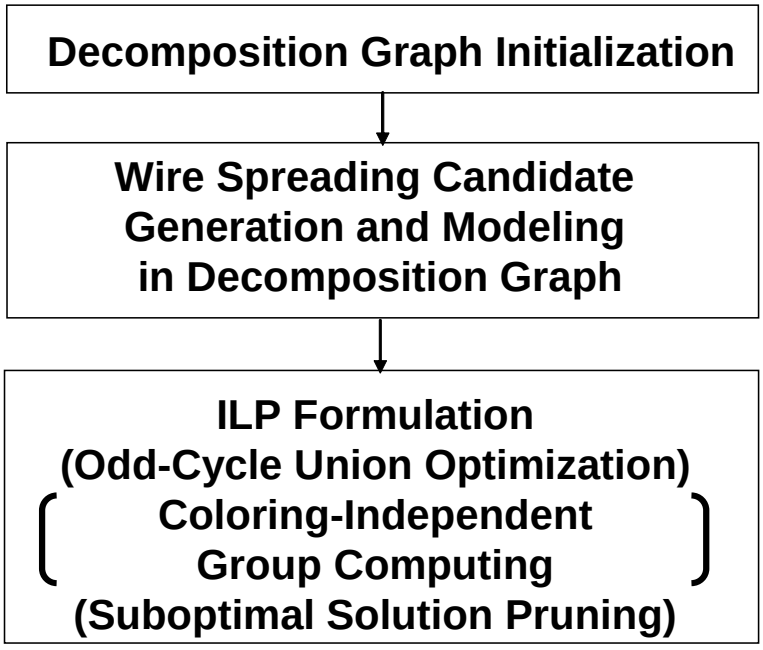


Figure 4.3: The overview of WISDOM

4.3 Basic Algorithms for WISDOM

In following two sections, we will present our WISDOM algorithm. The entire flow is shown in Figure 4.3. First, the input mask will be processed to construct an initial Decomposition Graph (DG). Next, we will generate a set of WSCs, which are expected to improve design decomposability. The DG will

be updated to take into account these potential configurations. To achieve high quality decomposition result as well as low amount of wire distortion, an integer linear programming algorithm is proposed to consider layout coloring and wire spreading candidates simultaneously. Since ILP is NP-Complete and its complexity grows exponentially with respect to problem size, we then propose three speed-up techniques to improve the efficiency of the basic ILP formulation in Section 4.3.3.

4.3.1 Decomposition Graph Initialization

For simplification purpose, we adopt a flow similar to [68] to construct an initial DG for modeling mask decomposition problem. Other approaches as in [35, 66] are also flexible to apply. There are two kinds of edges: *Conflict Edge* (**CE**) and *Stitch Edge* (**SE**). If and only if two nodes(polygons) are connected by CE/SE and in same/different masks, it results in a conflict/stitch.

The key steps of our construction method are briefly reviewed with the help of Fig. 4.4. Fig. 4.4 (a) shows the irregular polygons for original layout. If two polygons are within min_{cs}, there is a CE between them, marked by a dash line in Fig. 4.4 (b). The node projection, proposed in [53], is then performed, where projected segments are highlighted by bold curves of Fig. 4.4 (c). Based on projection result, all the legal splitting locations are computed next. The corresponding rectangles are split with SE added, which updates DG as Fig. 4.4 (d).

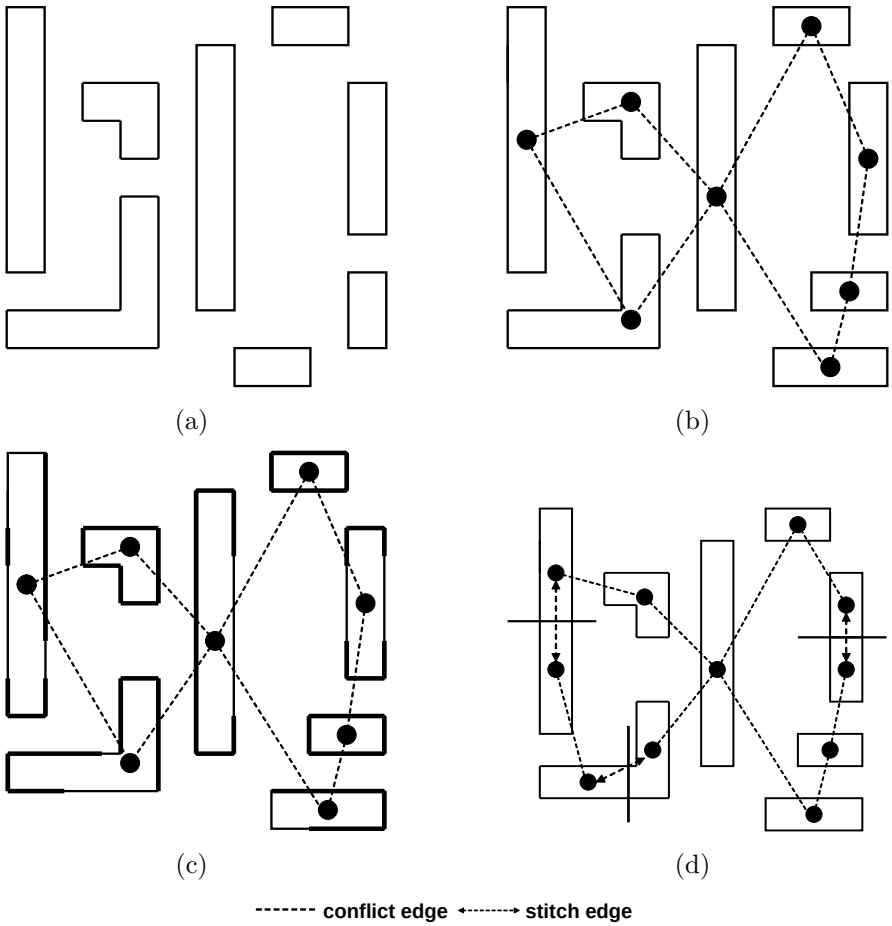


Figure 4.4: Initial decomposition graph construction

4.3.2 WSC Generation and Modeling

As the next step, we will generate a set of DPL-friendly WSCs, and model them in initial DG. There are two new types of edges introduced in the decomposition graph : *Conflict Elimination Edge* (**CEE**) and *Splitting Creation Edge* (**SCE**).

Each desired layout modification corresponds to a WSC, which will be constructed sequentially and independently based on original design. A *search region* is defined to avoid changing design too much. All the moved segments must be completely within their corresponding search regions, and the connectivity should be maintained. We also allow ripple movement of multiple wires. No design rules should be violated, and timing critical nets/vias are fixed. If the distance of two features in the original layout is larger than min_{cs}, this relationship should also hold after certain WSC is applied. This ensures no new coloring constraints are introduced in terms of double patterning lithography. It should be noted that, other user-defined conditions can be easily incorporated.

In the following, we present the key steps for WSC generation and modeling.

4.3.2.1 Spreading to Eliminate Conflict Edges

First, for each CE, we try to find a WSC, moving apart associated polygons beyond min_{cs}. This relaxes layout coloring constraints, since these two features can be assigned to the same mask consequently.

If such a WSC is available, we will change the status of the corresponding conflict edge to Conflict Elimination Edge (CEE). For the original layout in Fig. 4.5 (a), suppose there is a WSC like Fig. 4.5 (b) eliminating its conflict edge i-k, the DG is transformed to Fig. 4.5 (c).

During decomposition, if two nodes are connected by a conflict elimina-

tion edge and assigned into same masks, there is a conflict. However, different from conflict edge, if the corresponding WSC of this CEE is applied, this conflict can be removed.

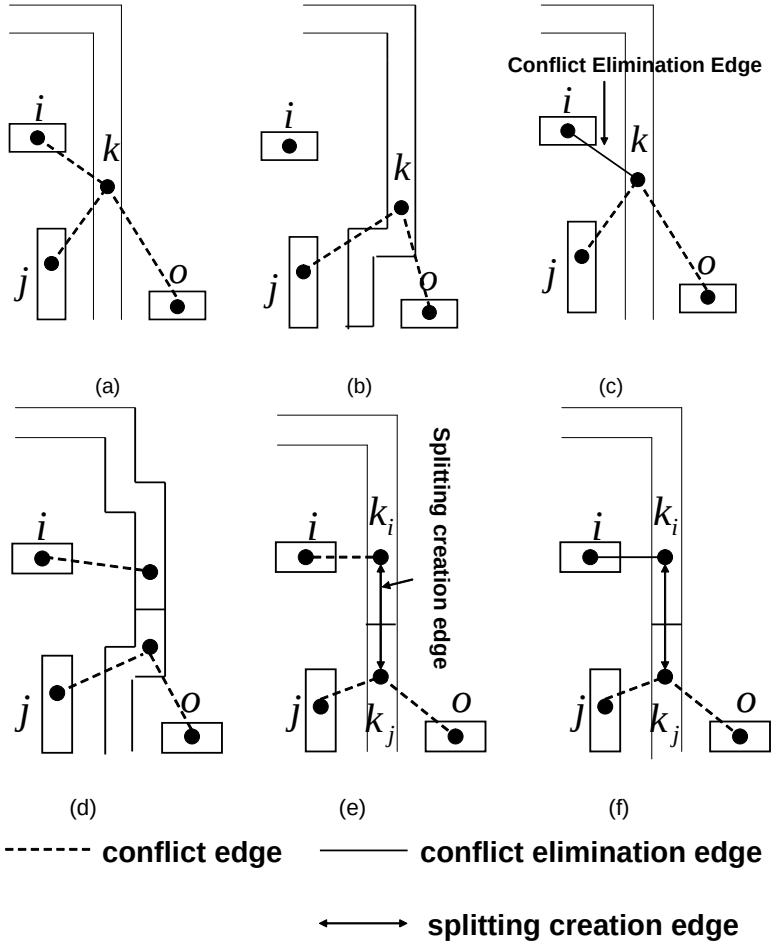


Figure 4.5: Wire spreading candidate modeling and decomposition graph updating.

4.3.2.2 Spreading to Create Splitting Locations

Then, we also detect these WSCs, which can create new potential stitch locations on initially unsplittable polygons. This provides larger solution space for double patterning mask decomposition.

As illustrated by the same example of Fig. 4.5 (a), suppose a new splitting location can be created on k by modifying layout as Fig. 4.5 (d). To model this WSC, polygon k is first split into two touching features k_i and k_j. An Splitting Creation Edge (SCE) is then added between k_i and k_j to indicate that this new potential stitch location results from wire spreading. The conflict edges between (i, k) and (j, k) are replaced by (i, k_i) and (j, k_j) respectively. Other edges connecting to k will be directed to either k_i or k_j, such as o-k_j in Fig. 4.5 (e).

During coloring, the two nodes linked by splitting creating edge should be in same color by default. Only if its corresponding WSC is applied, a stitch can be introduced and they can be assigned into different masks.

4.3.2.3 Non-compatible WSCs

Since we generate each WSC independently, they may not be applied in the same time. Two WSCs are *non-compatible*, if resulting in any of following problems:

1. Design rule violation or new conflict edge is introduced.
2. The same polygon is modified in distinct way.

Fig. 4.5 (f) illustrates the second case, with both WSCs in Fig. 4.5 (b) and (d) modeled. Obviously, these WSCs change polygon k differently, and hence only one can be picked.

After finding out all the WSCs, we will create a list of non-compatible WSC pairs based on above definitions.

4.3.3 ILP Formulation

To achieve good trade-off among conflict, stitch and layout perturbation minimization, in this section, we will formulate an integer linear programming to perform simultaneous mask decomposition and modification. The set of pre defined WSCs are the only available design pertubation configurations. Our ILP is different from [35, 53], because their formulations are not considering layout modification for decomposability improvement and simultaneous optimization. To better present, some notations are first listed in Table 4.1.

The co-optimization problem can be formulated as follows:

$$min(\alpha \sum_{e_{ij} \in E} c_{ij} + \epsilon \sum_{t_{ij} \in T} s_{ij} + \sum_{e_{ij} \in E^{cee}} m_{ij}^{cee} p_{ij}^{cee} + \sum_{e_{ij} \in E^{sce}} m_{ij}^{sce} p_{ij}^{sce}) \tag{4.1}$$

subject to

$$x_i + x_j <= 1 + c_{ij} \quad \forall e_{ij} \in E^{ce} \tag{4.2}$$

$$(1 - x_i) + (1 - x_j) <= 1 + c_{ij} \quad \forall e_{ij} \in E^{ce} \tag{4.3}$$

$$x_i + x_j <= 1 + c_{ij} + m_{ij}^{cee} \quad \forall e_{ij} \in E^{cee} \tag{4.4}$$

Table 4.1: Notation

r_i	The i_{th} layout polygons
x_i	binary variable denoting the coloring of r_i $x_i = 0$ if the color is GRAY, otherwise it is BLACK
e_{ij}	a conflict edge, a conflict elimination edge or a splitting creation edge between r_i and r_j
t_{ij}	r_i and r_j are touching each other, connected by a stitch edge or splitting creating edge
E^{ce}	the set of conflict edges
E^{cee}	the set of conflict elimination edges
E^{sce}	the set of splitting creation edges
E	the set of e_{ij}
T	the set of t_{ij}
c_{ij}	binary variable c_{ij} =1 when there is a conflict between r_i and r_j
s_{ij}	binary variable s_{ij} =1 when there is a stitch between r_i and r_j
m_{ij}^{cee}	binary variable $m_{ij}^{cee} = 1$ if and only if WSC for $e_{ij} \in E^{cee}$ is applied
m_{ij}^{sce}	binary variable $m_{ij}^{sce} = 1$ if and only if WSC for $e_{ij} \in E^{sce}$ is applied
p_{ij}^{cee}	layout modification cost, when m_{ij}^{cee} =1
p_{ij}^{sce}	layout modification cost, when m_{ij}^{sce} =1
$y_{ij,mn}^{cc}$	$(i,j) \neq (m,n)$. WSCs for $e_{ij} \in E^{cee}$ and $e_{mn} \in E^{cee}$ are not compatible.
Y^{cc}	the set of $y_{ij,mn}^{cc}$
$y_{ij,mn}^{ss}$	$(i,j) \neq (m,n)$. WSCs for $e_{ij} \in E^{sce}$ and $e_{mn} \in E^{sce}$ are not compatible.
Y^{ss}	the set of $y_{ij,mn}^{ss}$
$y_{ij,mn}^{cs}$	$(i,j) \neq (m,n)$. WSCs for $e_{ij} \in E^{cee}$ and $e_{mn} \in E^{sce}$ are not compatible.
Y^{cs}	the set of $y_{ij,mn}^{cs}$

$$(1 - x_i) + (1 - x_j) <= 1 + c_{ij} + m_{ij}^{cee} \quad \forall e_{ij} \in E^{cee} \tag{4.5}$$

$$x_i + (1 - x_j) <= 1 + s_{ij} \quad \forall t_{ij} \in T \tag{4.6}$$

$$(1 - x_i) + x_j <= 1 + s_{ij} \quad \forall t_{ij} \in T \tag{4.7}$$

$$m_{ij}^{sce} = s_{ij} \quad \forall e_{ij} \in E^{sce} \tag{4.8}$$

$$m_{ij}^{cee} + m_{mn}^{cee} <= 1 \quad \forall y_{ij,mn}^{cc} \in Y^{cc} \tag{4.9}$$

$$m_{ij}^{sce} + m_{mn}^{sce} <= 1 \quad \forall y_{ij,mn}^{ss} \in Y^{ss} \tag{4.10}$$

$$m_{ij}^{cee} + m_{mn}^{sce} <= 1 \quad \forall y_{ij,mn}^{cs} \in Y^{cs} \tag{4.11}$$

The objective function (4.1) is to minimize the weighted summation of conflicts and stitches as well as layout perturbation. The weights of α and ϵ are user-defined parameters, for assigning relative importance between these matrices. The layout penalty costs p_{ij}^{cee} and p_{ij}^{sce} are associated with respective conflict elimination and splitting creation edge.

Constraints (4.2)-(4.3) identify a conflict if two features connected by a conflict edge are in the same color. Constraints (4.4)-(4.5) are applied for conflict elimination edge. If m_{ij}^{cee} is one, the corresponding WSC is applied. As a result, the two polygons connected by e_{ij} are moved beyond min_{cs}, and they can be in the same mask without introducing conflicts. In such case, variable c_{ij} is always zero forced by the objective function. On the other side, when m_{ij}^{cee} is zero, its corresponding WSC is not used. Conflict will be detected based on the same logistic for the case of conflict edge as Constraints (4.2)-(4.3).

Constraints (4.6)-(4.7) are used to identify a stitch if two touching rectangles are colored differently. Constraint (4.8) follows the fact that any splitting creation edge e_{ij}^{sce} connects two touching rectangles t_{ij}. If and only if the stitch s_{ij} for t_{ij} is one, the corresponding wire spreading candidate is applied. Constraints (4.9)-(4.11) serve for the same purpose. If two WSCs are not compatible, at most one will be picked.

4.4 WISDOM Speedup Techniques

Since the time complexity of solving ILP is quite high in general, in this section, we propose three reduction techniques to simplify the decomposition graph without losing optimality.

Definition 8. odd/even cycle (OC/EC): *a cycle, whose total number of conflict edges and conflict elimination edges is odd/even.*

4.4.1 Odd-Cycle Union Optimization

Naively, ILP formulation would be performed on the entire decomposition graph. In this section, we will show that, it is sufficient to conduct ILP only on a subgraph of the DG, which is the union of all the odd-cycles. It will not lose optimality.

The overall flow is shown in Fig. 4.6. Since we are working on a much smaller graph, the CPU time of ILP solving is well reduced. Moreover, it only takes polynomial time to perform preprocessing and postprocessing steps, which details are discussed in Section 4.4.1.1 and 4.4.1.2 respectively. There-

fore, by taking the flow of Fig. 4.6, the coloring assignment can be effectively accelerated.

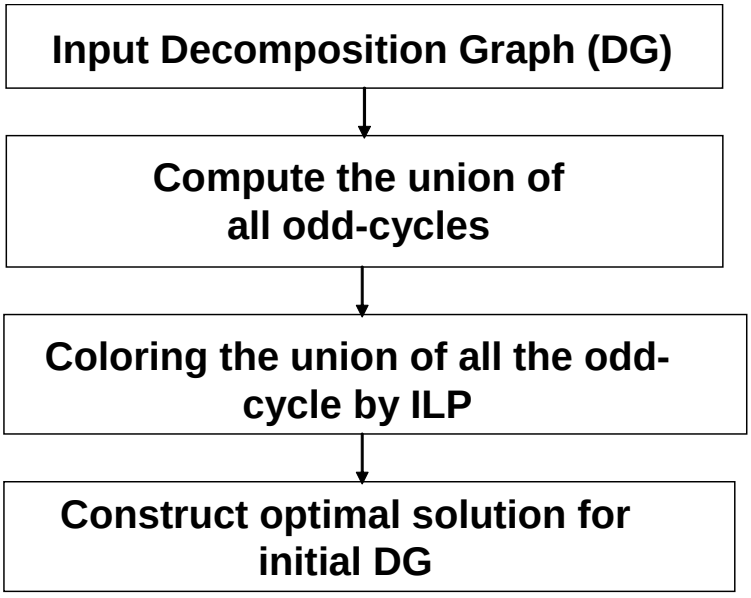


Figure 4.6: The flow of odd-cycle union optimization

4.4.1.1 Computing the union of all the odd-cycles

The naive calculation is to enumerate all the odd cycles and then find their union. This would be expensive since the number of OCs grows exponentially with respect to the size of DG. Our key idea is to make use of the concept of *cycle basis* and compute this union in a polynomial time. We do not have to dig out each individual OC.

Definition 9. cycle basis *and* base cycle: *Given a Decomposition Graph (DG),*

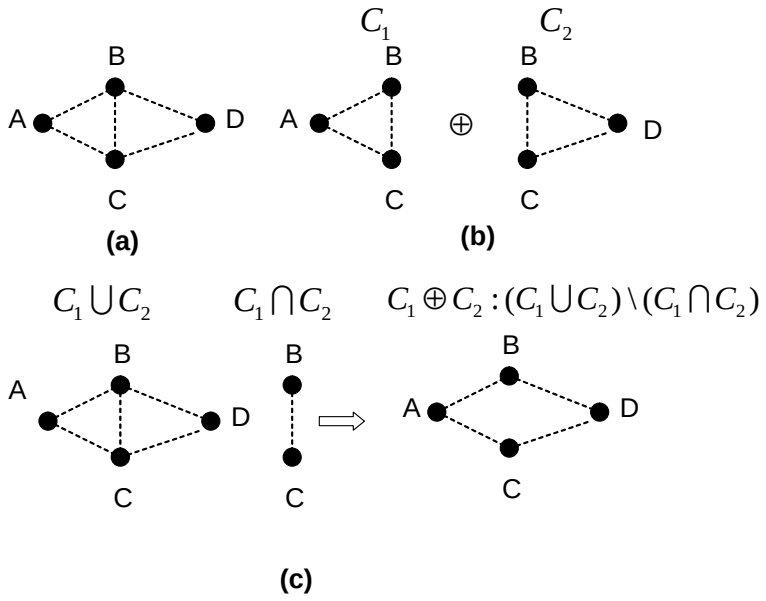


Figure 4.7: The concept of cycle basis

a subset of its cycles are called cycle basis, with each one called base cycle, if any cycle in DG can be generated by performing XOR operation on the cycles in this subset.

Definition 10. *XOR ($\oplus$) operation on two graphs is denoted as ($G_1 \oplus G_2$), which is the union of these graphs, minus their common edges. XOR ($\oplus$) operation on more than two graphs is computed as* $(((G_1 \oplus G_2) \oplus G_3) ... \oplus G_i)$.

As Fig. 4.7 (a) illustrates, there are three cycles. A-B-C-A and D-B-C-D could be a cycle basis, since the third loop A-B-D-C-A can be obtained by taking $\oplus$ on their edges.

Algorithm 6

Require: A decomposition graph, DG
Ensure: the union of all the odd-cycles: ODG
1: $OCB = \emptyset$
2: use depth-first search to calculate a cycle basis of DG: CB
//Line 3-11 compute one cycle basis for ODG.
3: move all the odd base cycles in CB into OCB
// now all the base cycles in CB are even cycles.
4: **while** OCB and CB share some common edges **do**
5: Make these common edges as COMMON
6: **for** any even base cycle EC in CB **do**
7: **if** EC have contain at least one COMMON edges **then**
8: move this EC from CB to OCB
9: **end if**
10: **end for**
11: **end while**
// OCB now is one cycle basis of the union of all the odd-cycles.
12: output the union of all the base cycles in OCB as ODG

Our efficient computation for odd-cycle union is given in Algorithm 6, with timing complexity of $O(N^3)$ time. Due to page limit, the detailed proof and analysis are skipped here.

4.4.1.2 Optimal solution construction for decomposition graph

After solving the union of all the odd-cycles by integer linear programming, one *optimal* coloring assignment needs to be constructed for original decomposition graph. Strictly speaking, *this constructed solution should have the same cost as the ILP assignment of DG, weighted by Objective (1).* The detailed steps are shown in Procedure 1, whose complexity is linear proportional to the size of graph. Its validity will be demonstrated when Theorem 4.4.1 is

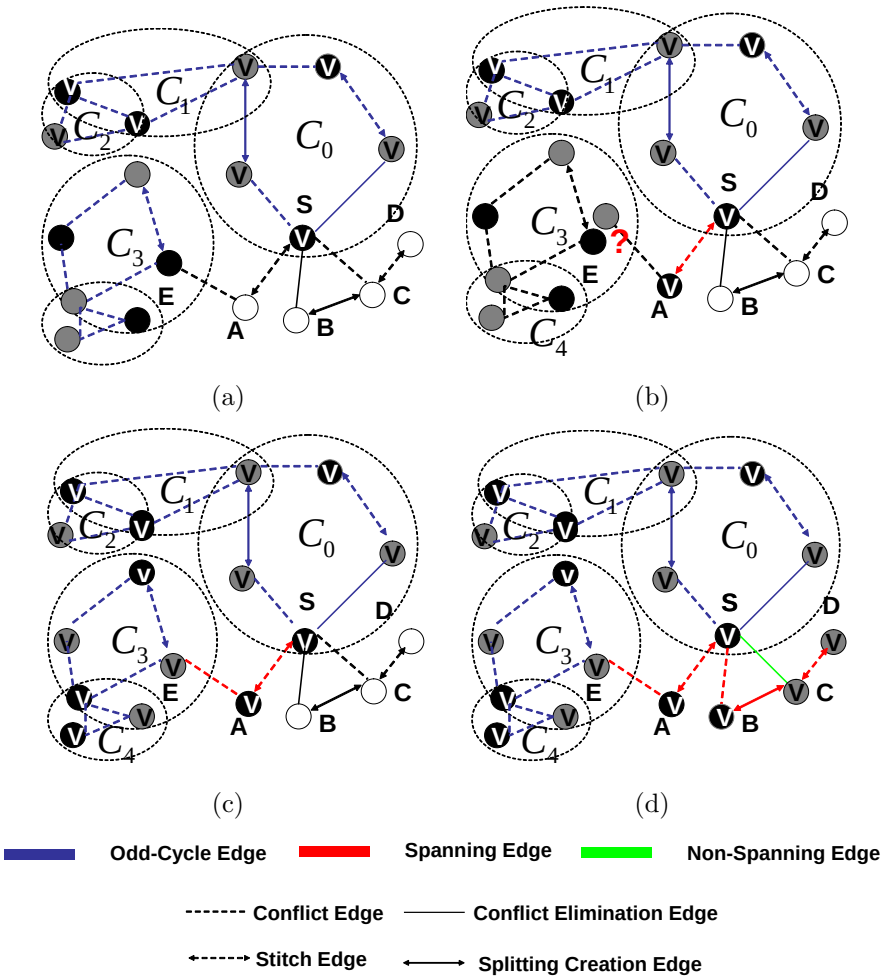


Figure 4.8: The procedure of solution construction for original decomposition graph. The symbol v inside each node denotes the state of visited. Blue, Red, Green are odd-cycle edges, spanning edges, and non-spanning edges respectively.

proven.

Procedure 1:

Step1: As Fig. 4.8 (a) illustrates, given a DG, assume we have obtained

a solution for the union of its odd-cycles C_0-C_4 by ILP, while other part of DG remains uncolored. All the edges in this union are marked as **Odd-Cycle-Edges (OCE)**.

Step2: Compute connected components for this odd-cycle union by depth first search, where no edge or vertices shared between different clusters. For example, there are two such components in Fig. 4.8 (a): (C_0, C_1, C_2) and (C_3, C_4).

Step3: Randomly pick one connected component, as (C_0, C_1, C_2) in Fig. 4.8 (a), and mark its nodes as visited sources. Then, start from these initial sources to traversal DG in a **depth-first manner** for exploring and colorings unvisited vertices. Each search phase consists in two steps: Step4 and Step5.

Step4: Given a current already-visited node s, we will explore its neighbors. If one of its edges leads to unvisited vertex t, we will mark edge s-t as **Spanning Edges(SPE)**. The mask of t will be assigned based on following *coloring propagation rule*s:

coloring propagation rule: if the type of edge s-t is a conflict edge or conflict elimination edge, we will assign the opposite color of s to t; otherwise, s and t will be on the same mask.

This rule ensures neither conflict/stitch nor layout modification will be introduced when a non-visited node is reached and colored through a SPE. As shown in Figure. 4.8 (b), the coloring of A can be determined as BLACK by

propagating the coloring of feature S through a stitch edge.

Step5: We will further check whether t is contained by some unvisited connected component, computed in *Step2*. If so, we will further assign the coloring of this entire component in one time, using its existing ILP solution. However, there might be a *coloring inconsistence problem*. As illustrated by Figure. 4.8 (b), after we propagate the solution of A, the color of E would be GRAY. However, E also belongs to a unvisited connected component (C_3, C_4), and it is assigned BLACK by ILP. Under such case, we can simply flip the existing ILP solution of this entire component (C_3, C_4) for maintaining coherence, shown by Figure. 4.8 (c).

Step6: The newly found t and CC will be marked as visited, and *Step* 4&5 will be recursively called on these nodes. □

Above steps can be repeated until all the nodes have been visited, as Figure. 4.8 (d) shows. All the OCEs and SPEs form a **depth first forest**. The edges which are in DG but not belonging to this forest are called **Non-Spanning Edges (NSE)**. When we explore the neighbors of an visited node s, these NSEs connect to other already-visited vertices. Edge C-S is an example of NSE. When we find node S from C, S has already been processed and colored.

Theorem 4.4.1. The solution generated by Procedure 1 is an optimal solution to the ILP formulation in (1) for the original decomposition graph .

Proof: Assume both original decomposition graph and its subgraph

of odd-cycle union are solved optimally by ILP, and their objective costs are ILP_{DG} and ILP_{OC} respectively. It is not difficult to see that ILP_{DG} should be no less than ILP_{OC}. Therefore, if the solution, constructed by Procedure 1, has the same cost as ILP_{OC}, weighted by Objective (1), it is actually an optimal coloring assignment for the original decomposition graph.

As described in Procedure 1, there are three types of edges: OCE, SPE, and NSE in the decomposition graph. First of all, it is easy to see, for all the OCEs, we preserve the same cost as their ILP solutions. In other words, these edges contribute a total penalty of ILP_{OC}. Secondly, for SPEs, since we follow *coloring propagation rule*s, there will be no extra conflicts, stitches or wire spreading involved, and the additional cost is zero. Therefore, in order to prove Theorem 4.4.1, we only need to show that NSEs will not introduce new penalty.

Without loss of generality, suppose NSE X-Y introduces a conflict on a conflict edge as in Fig. 4.9 (a), where its two connected nodes are in the same mask. In following few paragraphs, we will show that this configuration will never happen.

First of all, as Fig. 4.9 (a) illustrates, there must be a even cycle L formed by X-Y, and some edges in the depth-first forest of DG. From its definition in Section 4.4.1.2, this forest should be a connected component of DG and covers all the vertices, including X and Y. In consequence, there must be a path $X \leftrightarrow Y$ from X to Y in the depth-first forest, fully composed of OCEs and SPEs. This implies path $X \leftrightarrow Y$ and NSE X-Y form a loop L. This

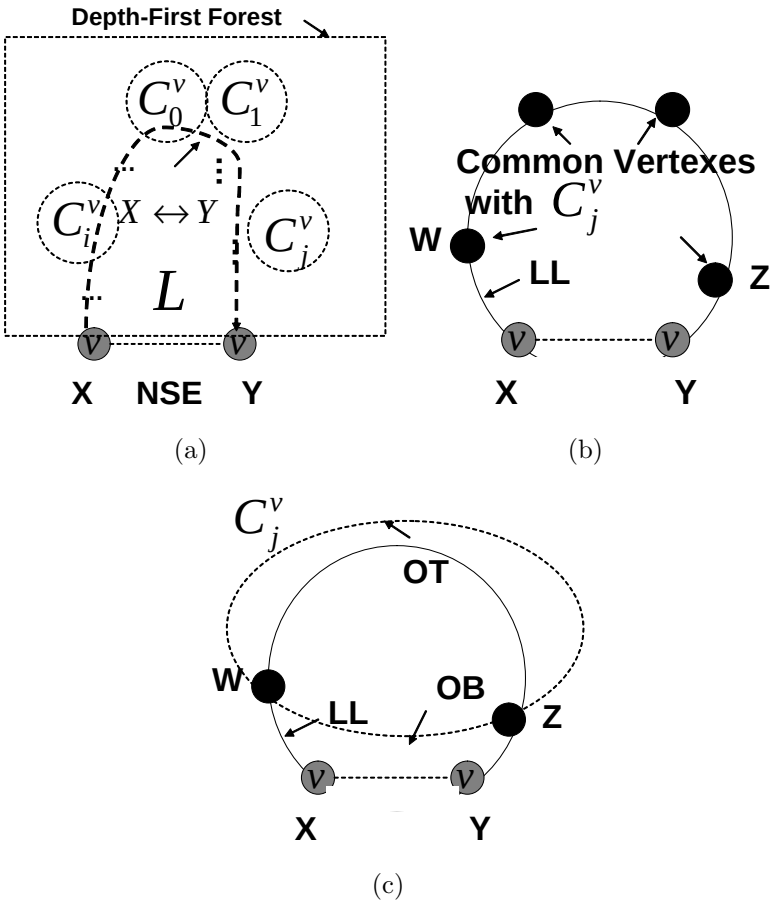


Figure 4.9: This figure explains no additional cost is introduced by non-spanning edge.

loop itself must be an even cycle, because it contains NSE X-Y which does not belong to any odd-cycle by definition.

Moreover, L and any OC can not have more than one common vertices. As Fig. 4.9 (b) illustrates, suppose L and one odd-cycle C_j^v share multiple common nodes, then these vertices divide L into several partial circular arcs.

Assume LL is the arc which contains edge X-Y and ends with two of common nodes, W and Z. Then, shown by Fig. 4.9 (c), C_j^v is also divided into two circular arcs OB and OT by W and Z. In consequence, it is not difficult to see that both W-LL-Z-OT-W and W-LL-Z-OB-W are loops. The total number of CEs and CEEs in these two cycles must have opposite parity, since the union of OT and OB is an OC C_j^v. As a result, one of W-LL-Z-OT-W and W-LL-Z-OB-W must be an OC. This implies, the edge X-Y belongs to an OC, and hence is a edge of OCE. This contradicts the fact that X-Y is a NSE.

Therefore, any OC and even cycle L share at most one vertex. In other words, any edge of L does not belong to any odd-cycle, and hence must be either a SPE or NSE. In consequence, partial arcs of L, path $X \leftrightarrow Y$ in Fig. 4.9 (a), should solely be composed by SPEs, given the truth that this path is part of depth-first forest. This implies, the mask assignments of all the nodes along the edges of $X \leftrightarrow Y$ follow *coloring propagation rules*. Combined with the fact that L is an even cycle, it is not difficult to derive that X and Y must be in the different masks. As a result, the conflict shown in Fig. 4.9 (a) can not exist.

Similarly, we can show that NSEs do not introduce any additional stitch and layout modification. In summary, Theorem 4.4.1 is proven.□

4.4.2 Coloring-Independent Groups

After finding the union of all the odd-cycles, we compute graphically-disjointed connected components as [35, 53, 68] for improving scalability. Fur-

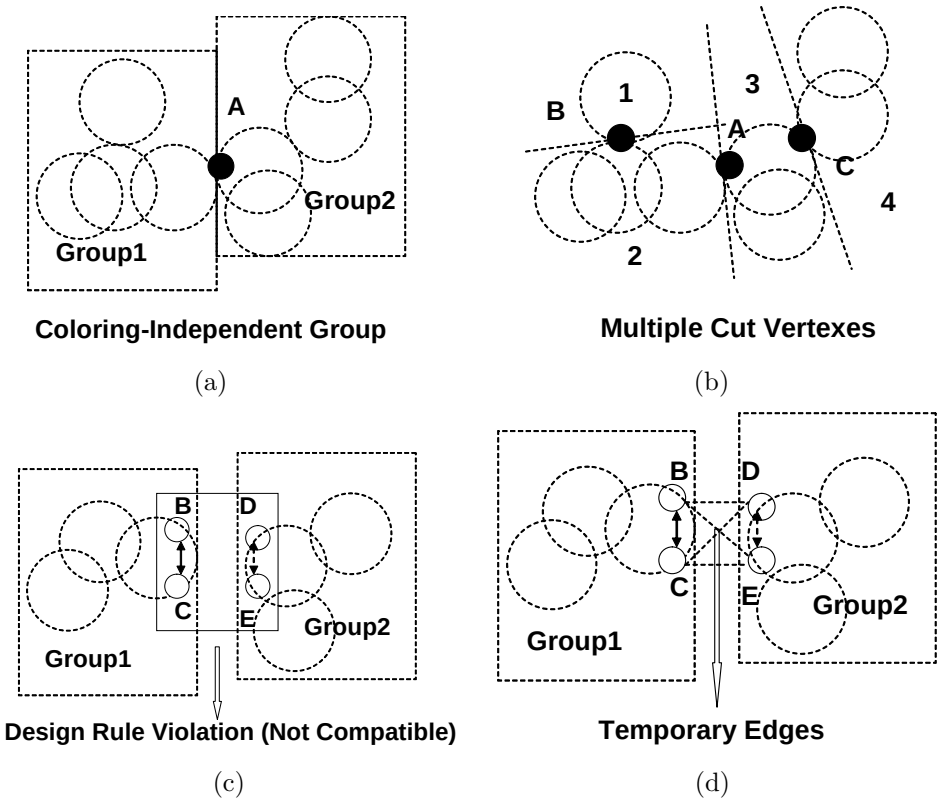


Figure 4.10: Figures (a) and (b) explain the concept of coloring-independent groups. Figures (c) and (d) explain how to handle wire spreading candidate.

thermore, we observe that, in terms of mask decomposition problem, each connected component may still be divided into several coloring-independent groups. This reduces ILP problem size to a greater extent.

Fig. 4.10 (a) shows a simple motivational example, which is a connected component. We observe that, node A is the only common vertex between group one and two, and these two groups can still be solved by ILP individ-

ually while maintaining optimality. The reason is that, after their respective optimal solutions are obtained, if the coloring of feature A from both groups are different, we could simply flip the solutions of one group without effecting overall solution quality.

In graph theory, such node as A is called cut vertex or articulation point, whose removal creates disconnected components. Generally, if there are multiple cut vertices, the initial graph can be decomposed into a chain of coloring-independent groups linked by these articulation points. As Fig. 4.10 (b) illustrates, four groups 1-2-3-4 are connected sequentially with cut vertices A-C as boundary nodes. Similar to the simple case of Fig. 4.10 (a), we can solve each group individually without losing optimality. Their solutions can be merged by appropriately flipping the coloring of certain groups. The detailed algorithm for finding all the cut vertices and related coloring-independent groups can be done in a polynomial time by depth-first search according to the work of [72].

4.4.3 Suboptimal Solution Pruning

Given any coloring-independent group, we can further simplify it by performing solution pruning for underlying substructures, *sequential path*, defined as follows:

Definition 11. sequential path (SP): an acyclic linked list of nodes is a sequential path, if

- Except two ending features, all the nodes must have a degree of two.

- Any WSC associated with this list and any other WSC outside are compatible.
- This list can not be totally included in another list which satisfy first two conditions.

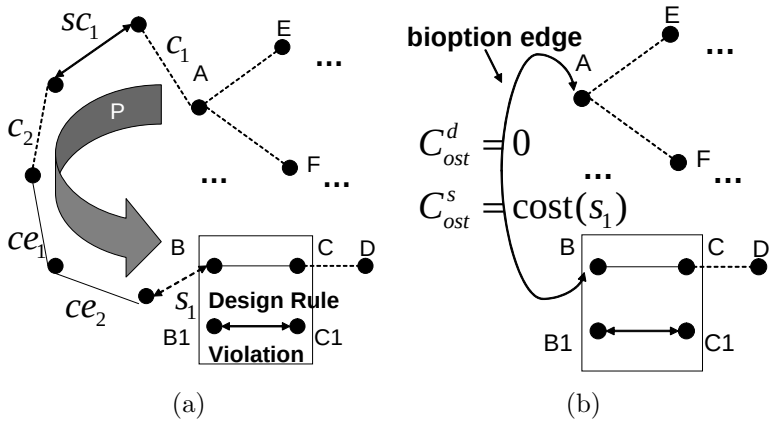


Figure 4.11: Suboptimal Solution pruning.

Based on the above definition, in Fig 4.11 (a), Path P is a sequential path with node A and B as ending features. It is composed of two CEs c_1-c_2, one SE s_1, one SCE sc_1 and two CEEs ce_1-ce_2. Ending node A can not be extended to E or F, because in that case, A becomes a internal vertex but has a degree of three. This violates condition 1. Edge B-C can not be further included in this SP as well since it is not compatible with splitting creating edge B1-C1, which is outside of path P. Condition 2 does not hold, then.

The nice property of SP is that, besides two ending nodes, each sequential path will not have coloring or design modification interaction with other parts of this graph. In other words, given coloring configurations of two ending nodes, the best decomposition of a SP can be uniquely determined.

There are only two possible configurations for each SP, depending on whether head and tail features have same color or not. As Fig 4.11 (b) shows, since there are four CEs/CEEs, when A and B are assigned into same mask, no stitches or layout modifications are needed for zero conflict solution. As a result, the best ILP penalty C^s_{ost} for path P is zero. On the other hand, if A and B are assigned into different masks, one of s_1, sc_1 and ce_1-ce_2 should be applied to resolve potential conflict. Assume the cost of s_1 is the smallest, we pick it as a local optima C^d_{ost}.

Therefore, graphically, we can replace this whole SP by a bioption edge, which only stores possible optimal costs, C^s_{ost} and C^d_{ost}. While formulating ILP problem, we simply apply following four equations to check whether node A and B are in same/different masks, where binary variable d_{ij} is zero/one.

$$x_A + (1 - x_B) <= 1 + d_{ij} \quad \forall e^b_{ij} \quad bioption \quad edge \tag{4.12}$$

$$(1 - x_A) + x_B <= 1 + d_{ij} \quad \forall e^b_{ij} \tag{4.13}$$

$$x_A + x_B <= 2 - d_{ij} \quad \forall e^b_{ij} \tag{4.14}$$

$$(1 - x_A) + (1 - x_B) <= 2 - d_{ij} \quad \forall e^b_{ij} \tag{4.15}$$

A expression of $C^s_{ost}(1 - d_{i,j}) + C^d_{ost}d_{i,j}$ will be added into Objective (4.1) to take into account corresponding optimal ILP penalty for the represented sequential path. Comparatively, in original ILP, for sequential path P, we need to introduce one additional binary variable for each internal vertex. Moreover, for every edge, at least two constraints out of Constraints(2)-(8) should be specified. As a result, by conducting suboptimal solution pruning, it reduces number of variables and constraints.

4.5 Experimental Results

We implement our algorithm in C++ and test on Intel Core 3.0GHz Linux machine with 32G RAM. OpenAcess2.2 [73] is used for interfacing with GDSII directly. Moreover, we choose glpk [54] as our solver for integer linear programming. ISCAS-85&89 benchmarks are scaled down and modified as our test cases. The metal one layer is used for experimental purpose, because it is one of the most trouble some layers in terms of double patterning lithography. The minimum width and spacing become 40nm. The minimum coloring space for double patterning is set as 65nm, and minimum overlapping margin for stitch insertion is 10nm.

The penalty p^{ce}_{ij} and p^{sc}_{ij} due to wire spreading are set proportional to the increased wirelength and number of jobs in their respective WSCs. Then we calculate the weight summation of all the WSCs, and set the parameter ϵ a bigger value than that weight. This ensures the priority of stitch minimization over design modification. Similarly, suppose there are at most n possible

stitches, we set parameter α larger than n times ϵ, which guarantees conflict elimination weights more than stitch reduction.

4.5.1 Statistics on Decomposition Graph

The detailed statistics of constructed decomposition graphs are shown in Table 5.1. The first column denotes circuit name. Columns "#ce" and "#se" under "initial DG" are the total number of conflict edges and stitch edges in initial decomposition graph. Columns "#cee" and "#sce" under "updated DG" show the respective number of conflict elimination and splitting creation edges, added in WSC generation and modeling step. "#cee" plus "#sce" equal to the total number of WSCs "#WSC". "total" is the summation number of all the test cases, and "ratio" is computed percentage of corresponding metrics.

From Table 5.1, we have WSCs which can eliminate 8% conflict edge and create 9% more stitch candidates. Although these percentages seem relatively small, however, since DPL layout decomposition has a ripple effect, it could remove more than one conflicts or stitches by just applying one WSC. On the other side, the increased graph size due to "#cee" and "#sce" would degrade the performance of ILP. As we show later, with proposed graph reduction techniques, this side effect is well encountered.

Table 4.2: Statistics on decomposition graph.

circuit	initial DG		updated DG		
	#ce	#se	#cee	#sce	#WSC
C432	1063	964	36	14	50
C499	2428	1437	78	88	166
C880	2464	2439	177	196	373
C1355	3101	3768	74	104	178
C1908	5109	5648	262	96	358
C2670	8750	8655	596	420	1016
C3540	10896	10864	850	768	1618
C5315	16049	15654	1112	670	1782
C6288	13389	11014	264	530	794
C7552	22516	23525	1453	1122	2575
S1488	5273	4284	499	428	927
S38417	69270	57204	6302	2908	9210
S35932	86540	58661	8553	7634	16187
S38584	170079	7140	14191	7764	21955
S15850	169147	124969	12422	8920	21342
total	586074	336226	46869	31662	78531
avg	1	1	0.08	0.09	-

4.5.2 Result Comparison

For comparison, we implement a post-routing mask decomposition algorithm [68], which is the extension of [53]. The metric *unresolvable conflict edge* they applied is actually consistent with our definition of conflict, which is described from another point of view. This algorithm is performed directly on the initial DG, which is obtained in the first step of our flow.

We can not compare the work of [35, 69] directly. In [35], they work on extremely fine metrics, conflict grid. In [69], their algorithm is designed

Table 4.3: Result Comparison

	[68]				WISDOM			
circuit	$cflt$	$stitch$	$WL(e^5)$	CPU	$cflt$	$stitch$	$WL(e^5)$	CPU
C432	55	11	2.781	0.27	48	14	2.784	0.09
C499	258	11	5.792	0.74	214	11	5.809	0.27
C880	125	105	2.920	0.62	32	83	2.925	0.2
C1355	82	89	86.790	0.66	31	102	86.810	0.32
C1908	99	346	14.440	1.91	55	343	14.462	0.4
C2670	254	749	23.730	3.13	49	655	23.857	1.08
C3540	472	643	30.162	3.38	67	619	30.350	0.73
C5315	413	1234	43.700	3.6	89	949	43.967	1.1
C6288	912	331	35.240	5.78	663	340	35.340	0.76
C7552	708	1544	62.300	4.5	166	1338	62.303	1.6
S1488	274	316	14.300	2.01	60	134	14.372	0.44
S38417	3866	868	184.000	24.88	2518	471	184.552	5.51
S35932	11731	1383	407.400	203.24	7006	875	409.240	14.22
S38584	11254	948	443.000	127.75	6635	1139	444.580	11.67
S15850	11579	3392	431.200	66.15	7198	2103	433.780	12.26
total	42082	11970	1787	448.62	24831	9176	1795	50.6
ratio	1	1	1	1	0.59	0.77	1.004	0.11

planar graph only, while our decomposition graph is not. We are also not able to compare with another DPL-driven post-routing layout modification work [71], because our targeted problem is different. In [71], they focus on technology migration for stand-cell library only, and area minimization is one of their objectives. We work on full-chip design perturbation, and the chip size is fixed.

Table 4.3 lists the comparison of decomposition results, where "cflt" and "stitch" are the number of conflicts and stitches in the colored layout. The column "WL" shows the wire length in terms of "nm". CPU is the

computational time in terms of "second", including both graph construction and ILP solving steps.

As we can see, our algorithm significantly outperforms [68] in terms of quality, which generates a solution with 41% and 23% reduction on conflict and stitch number respectively. The layout perturbation ratio is only 0.4%. With respect to runtime, we achieve 9X speed-up. For some benchmark, such as c432, WISDOM does produce more stitches. The reason is that in our experimental setting, conflict elimination is set as higher priority job over splitting reduction. The number of stitches could increase comparatively, to better remove the conflicts.

Although after WISDOM, there are several conflicts and stitches remaining in the design, we observe they are mainly resulting from the pins/vias in standard cells and highly-congested routing paths. While we search WSCs, most of these features are set as fixed for avoiding modifying design and timing too much. Thus, our WISDOM can be used in combination with high-level DPL-friendly design methodologies [4, 36, 70, 71].

4.5.3 Efficiency

We further study the effectiveness of various acceleration techniques in Table 4.4. "CPU(base)" shows the runtime of our algorithm without proposed speed-up approaches, in terms of "seconds". For fair comparison, the layout partition technique in [68] is applied in this baseline. Then, we add in each acceleration technique incrementally. In all the columns, "w/o" and

"w" show the respective data without and with certain method, and "CPU" is the resulting runtime after it is adopted. We double check simulation results, and ensure that no solution quality is lost by applying these graph reduction approaches.

The columns under "odd-cycle union optimization" show the graph statistics of initial DG and its odd-cycle union. The number of nodes and edges are reduced by 61% and 40% respectively in average. This reduction is significant in terms of ILP solving, which has exponential complexity with respective to problem size. As seen from Table 4.4, we achieve 2.7x speed-up.

The effectiveness of coloring-dependent group computing is investigated then with results listed under "+c-independent". "#CC" is the number of connected components, computed by the layout partition technique in [68]. By using cut vertex, we further divide each component to multiple coloring-independent groups, which can also be solved by ILP individually. The total number of such groups is shown in "#CG", which is averagely 20% more than "#CC". With this technique, the ILP problem size becomes smaller, and the runtime is further reduced by 37%.

Last, we apply the technique of suboptimal solution pruning, where the results are listed under "+suboptimal solution pruning". "#var" and "#con" are the total number of variables and constraints in the ILP formulations. As we can see, solution pruning technique can reduce their number by 61% and 53%, respectively. Therefore, the coloring assignment can be effectively accelerated by another 2.4X.

4.6 Summary

In this chapter, we have developed a wire spreading enhanced decomposition of masks algorithm for double patterning lithography. Our approach is featured by integer linear programming and efficient graph reduction techniques. The experimental results show 41% and 23% reduction on the number of conflicts and stitches respectively with 9x speed-up.

Table 4.4: The effectiveness of various acceleration methods

circuit	CPU(base)	odd-cycle union optimization					+c-independent			+suboptimal solution pruning				
		#nodes		#edges		CPU	#CC	#CG	CPU	#var		#con		CPU
		w/o	w	w/o	w		w/o	w		w/o	w	w/o	w	
C432	0.35	2350	715	3398	798	0.26	127	138	0.22	1596	792	1502	914	0.09
C499	1.2	3996	1539	7485	1926	0.78	164	180	0.55	3854	1933	3449	2013	0.27
C880	0.64	5525	2195	6562	2428	0.57	293	311	0.64	4864	1904	4604	2254	0.2
C1355	0.87	7893	3084	9633	7893	0.67	551	585	0.82	6482	2796	6291	3449	0.32
C1908	2.26	12122	4709	15321	12122	1.35	635	699	1.52	10366	4578	9828	5497	0.4
C2670	2.96	18784	8788	23875	18784	2.05	916	977	2.93	20078	8849	18765	10229	1.08
C3540	4.53	23878	10108	29925	23878	2.61	1196	1227	2.77	22784	9092	21469	10603	0.73
C5315	3.91	34556	15245	44201	17320	2.92	1779	1920	2.18	34650	14567	32424	17056	1.1
C6288	5.82	29192	8460	40968	9181	3.03	1266	1668	4.89	18368	8006	17239	9486	0.76
C7552	5.22	50650	21565	62801	24029	4.14	2716	2869	1.27	48066	20061	45441	23769	1.6
S1488	1.38	10538	4752	9771	5430	1.2	585	622	1.38	100680	4046	10050	4686	0.44
S38417	27.54	141535	47592	127928	52660	14.67	5791	6989	12.53	105342	44790	99055	5222	5.51
S35932	223.56	311931	139035	149018	152925	57.51	17123	20960	29.58	305906	131384	288123	147123	14.22
S38584	129.85	338771	117352	181101	131147	49.46	13241	16605	27.97	262388	114148	245135	126849	11.67
S15850	83.32	326454	132463	298576	147940	41.93	14960	18207	31.73	295942	123066	277157	137241	12.26
total	493.41	1318175	517602	1010563	608461	183.15	61343	73957	120.98	1241366	490012	1080532	506391	50.65
ratio	1	1	0.39	1	0.60	0.37	1	1.20	0.24	1	0.39	1	0.47	0.1

Chapter 5

Electronic Beam Stencil Design with Overlapped Characters

5.1 Introduction

Electronic Beam Lithography (EBL) is one of promising emerging technologies in sub-22nm regime. In EBL, the desired circuit patterns are directly shot into wafer, which overcomes the diffraction limit of light in current optical lithography system. However, the low throughput becomes its key technical hurdle. In conventional EBL system, Variable Shaped Beam (VSB), the layout is decomposed into a large number of rectangles, and each rectangle will be projected by one electronic shot. This would be extremely slow. As an improved EBL technology, Character Projection (CP) shoots complex shapes, **characters**, in one time, by putting them into a pre-designed **stencil**. However, only limited number of characters can be employed, due to the area constraint of the stencil. Those patterns, not contained by any character, are still required to be written by VSB.

Many methodologies have been proposed to design and select group of circuit patterns as characters of stencil for minimizing total projection time of both CP and VSB. In [74], frequently-used standard cells are greedily chosen

as characters, processed by CP technology. M.Sugihara at el. [75–78] employ integer linear programming to optimize the throughput, given a set of character candidates. Recently, EDA vendor D2S Inc [43] proposes improving stencil design from a new point of view, but with no detailed algorithm presented. They show that, in practice, when individual character/template is designed, blanking area is usually reserved around its boundaries. By sharing blanks between adjacent templates, more characters can be placed on the stencil than the regular design of Fig. 1.6 (d), better improving the throughput.

The work of [43] implies that, to fully minimize the total projection time of EBL, besides selecting appropriate characters as [74–78], their relative locations on the stencil should also be taken into account in the same time due to possible overlapping.

In the chapter, we will investigate on this new problem of electronic beam lithography stencil design with overlapped characters. One/two dimensional problem is researched separately, depending on whether the available overlapping space of characters is non-uniform in either/both horizontal and/or vertical directions. The main contributions of our work are stated as follows.

1. We co optimize the selection process of characters and their physical placements on stencil for effective EBL throughput improvement.

2. We propose a four-phase iterative refinement process to conduct one-dimensional stencil design optimization. A Hamilton-path based ap-

proach has been developed to solve single-row reordering optimally.

3. We develop a Sequential Pair (SP) based simulated annealing framework to optimize general two-dimensional stencil design. Two SP-related techniques have been proposed to ensure correct and fast character placement evaluation, and two specialized perturbation methods have been developed for robust solution improvement of simulated annealing process.

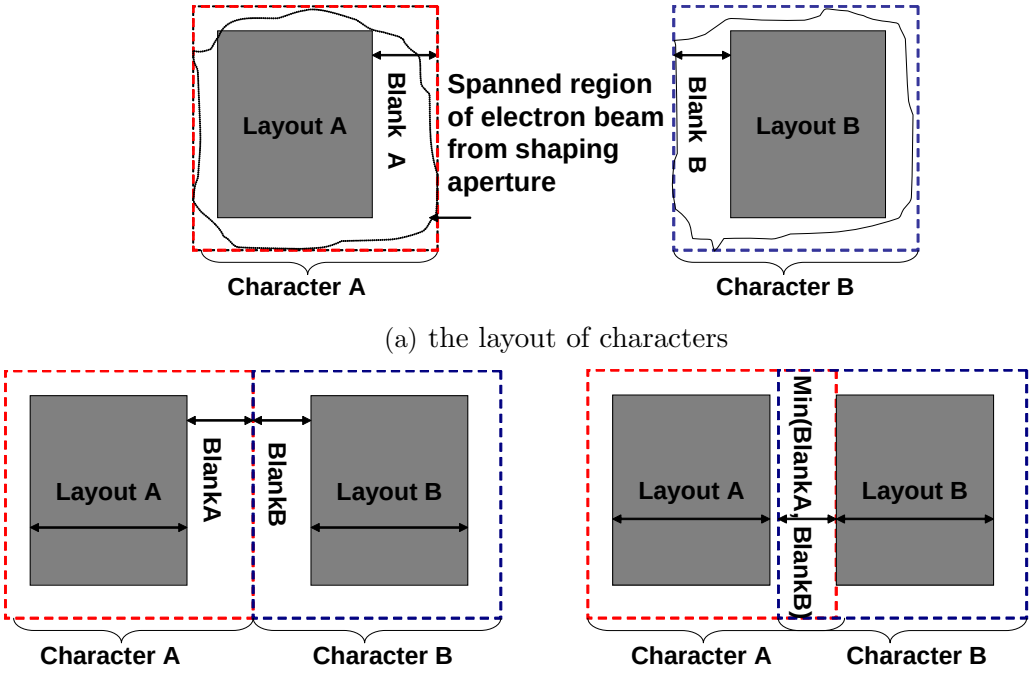
The rest of the chapter is organized as follows. Section 5.2 provides the preliminaries and problem formulation. The detailed algorithm is described in Section 5.3 and 5.4. Section 5.5 presents the experiment results and Section 5.6 concludes this chapter.

5.2 Preliminary and Problem Formulation

5.2.1 Overlapped Character

Various investigation [75–78] have been conducted on the optimization of character selection for EPL technology, where no intersection is allowed between templates on the stencil, as shown by Fig. 1.6 (d). Recently, the work of [43] shows that the design of stencil can be further improved by overlapping adjacent characters, which allows more templates to be put and increases the throughput.

As pointed out by [43], when individual character is designed, blanking space is usually reserved around its enclosed rectangular circuit pattern, shown by Fig. 5.1 (a). The reason is that, when the electron beam is scatted from the



(a) the layout of characters

(b) conventional stencil for (a)

(c) overlapped characters

Figure 5.1: Overlapped characters for improving the stencil densities.

shaping aperture of Fig. 1.5 (b), it could span larger area on the stencil than the layout to be printed. In order to avoid projecting any unwanted image, the white space should be preserved. These blanking areas offer great opportunity for character sharing.

Suppose the required white space around layout A and B are $BlankA$ and $BlankB$ respectively, in Fig. 5.1 (a). If the characters are conventionally aligned by edge as Fig. 5.1 (b), it results in a waste of area. The space between layout A and B is actually $BlankA + BlankB$, which is more than required for

both patterns. By contrast, we would greatly reduce the total area of character A and B by sharing an amount of $min(BlankA, BlankB)$ space. In this case, $max(BlankA, BlankB)$ white width is still reserved between layout A and B, which is sufficient for ensuring correct printing image.

5.2.2 Stencil Design Challenge

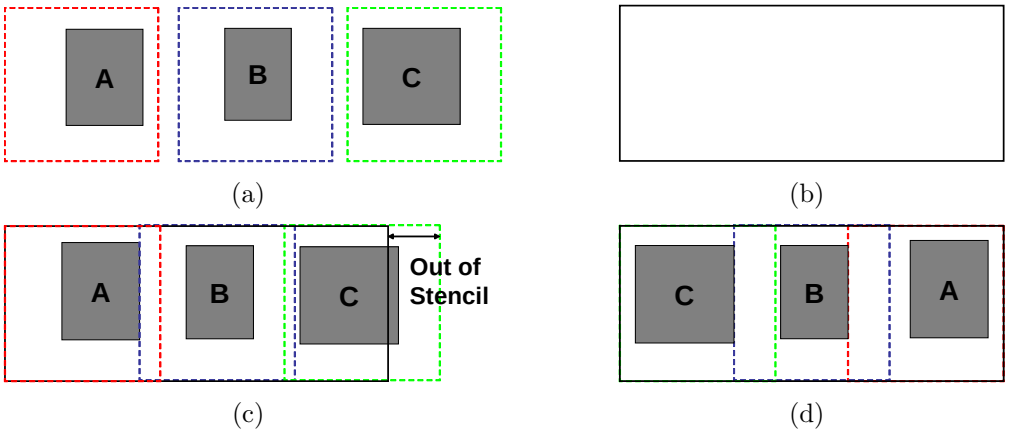


Figure 5.2: The main difficulty of stencil design with overlapped characters

The main challenge of stencil design with overlapped characters comes from the fact that, for each character, the amount of required blanking space is not uniform, strongly depending on its enclosed layout patterns. In consequence, for different placements of characters, the area reduction from template overlapping may vary a lot. Therefore, unlike the traditional design of Fig. 1.6 (d), the number of maximum allowable characters in the stencil is not fixed. To achieve high quality solution, the detailed physical placement

information of all the characters must be taken into account. This makes the problem of stencil design with overlapped characters not only different from but also more difficult than conventional non-overlapping one, addressed in [75–78].

As the example of Fig. 5.2 (a) illustrates, suppose there are three character candidates A-C, and we would like to pack them into a simple stencil of Fig. 5.2 (b) for minimum projection time. As easily seen, their blanking spaces are quite different. In conventional design where overlapping is not considered, at most two of them can fit. On the other side, when the blanking space is shared by adjacent characters, the result is correlated with the detailed physical implementation of stencil, and could be different from traditional design. If these three candidates are tried out by the order of A-B-C like Fig. 5.2 (c), only A and B can be put in. Pattern C is out of bound and has to be processed by VSB technique. This does not lead to higher throughput than conventional non-overlapped methodology. In contrast, if rearranged as C-B-A as Fig. 5.2 (d), all of these three patterns can be used as CP characters. Obviously, it is a better stencil optimization.

5.2.3 Problem Formulation

In this subsection, we will formulate the problem of EBL stencil design with overlapped characters.

Similar to previous work [75–78], we assume a set of character candidates have already been given. To model overlapping information, as Fig. 5.3

(a) illustrates, assume the blanking spaces of each candidate c_i, from left, right, top and bottom boundaries, are l_i, r_i, t_i and b_i, respectively. The orientation of these candidates are not allowed to be flipped, since it actually becomes a different template, as explained in [76]. When two candidates c_i and c_j are put adjacent to each other horizontally, their maximum allowed overlap is set as o_{ij}^H, which is $min(r_i, l_j)$ as shown by Fig. 5.3 (b). Similarly, Fig. 5.3 (c) defines the maximum vertical overlapping margin o_{ij}^V. o_{ij}^H and o_{ij}^V vary for different i and j.

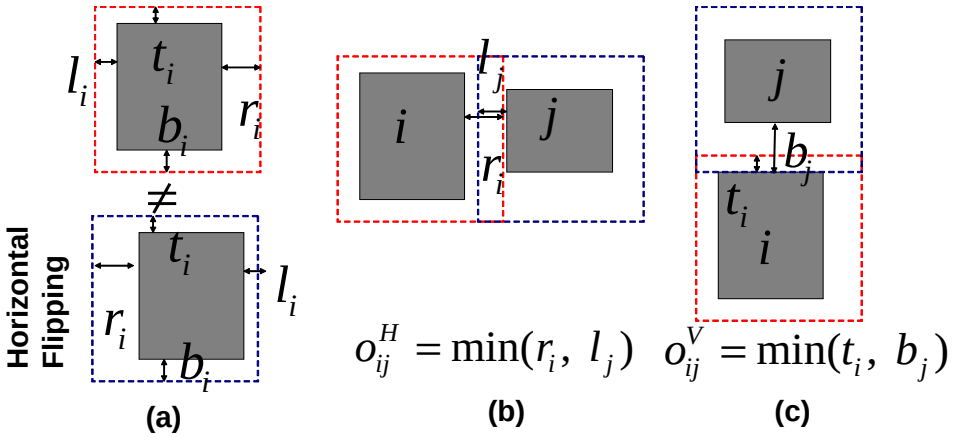


Figure 5.3: The dimensional variable of character candidates

Moreover, since the manufacturing time of EBL is dominantly determined by electronic beam shooting, in our work, we make use of total number of shots as the measurement of projection time. Suppose each candidate c_i is referred R_i times in the chip. For each of its appearance, the candidate c_i will be projected by either CP or VSB method, with a number of shots n_i^{CP}

and n_i^{VSB}. The total processing time (number of shots) of the entire circuit is computed by following equation.

$$\sum_{c_i \in C^{CP}} R_i n_i^{CP} + \sum_{c_i \in (C^C \setminus C^{CP})} R_i n_i^{VSB} \tag{5.1}$$

C^C is the set of all the character candidates. C^{CP} is the union of selected candidates processed by CP method, which is a subset of C^C.

In our work, for simplification purpose, we only design and optimize the stencil for single design. The general case of multiple chips can be easily extended, where the characters would be reused by different designs. Based on above description, our optimization problem can be stated as below:

Problem Formulation: Given a design and its set of character candidate C^C, select a subset C^{CP} out of C^C as characters, and place them on the stencil S. The objective is to minimize the total projection time (number of shots) of this design expressed by Equation (1), while the placement of C^{CP} is bounded by the outline of S. The width and height of stencil is W and H, respectively, and all the candidates have unique width w and height h. The maximum horizontal and vertical overlapping margins between adjacent characters are given by o_{ij}^H and o_{ij}^V, respectively.

In this chapter, we will first investigate on the special case of one dimensional stencil design in Section 5.3, where the amount of blanking spaces differs only in either horizontal or vertical direction. Then, in Section 5.4, the

algorithm, for generalized two dimensional problem, will be developed.

5.3 One Dimensional Stencil Design

Normally, each template implements one standard cell. That is to say, the enclosed circuit patterns of all the characters have the same height, and their layouts near top and bottom boundary edges are mostly regular power rails. As a result, illustrated by Fig. 5.4 (a), the required blanking spaces on the top t and bottom b are nearly identical for these candidates.

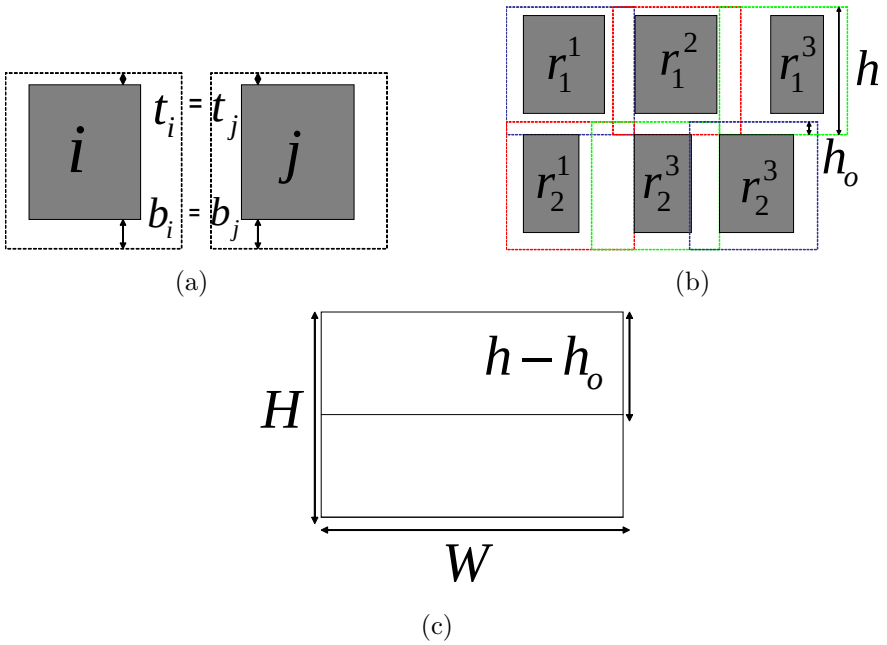


Figure 5.4: One-dimensional Stencil Design.

Therefore, in such case, characters are usually be placed on the stencil in a row-based manner, shown by Fig. 5.4 (b). All rows have a unique

height h. The overlapped blanking margin h_o between adjacent rows are also the same, which is $\min(t_i, b_i)$. In consequence, as Fig. 5.4 (c) shows, the overlapping-aware stencil design becomes a one-dimensional problem. The number of character rows can be pre determined as $\lfloor H/(h - h_o) \rfloor$). The candidates would be packed into these rows with maximum width W.

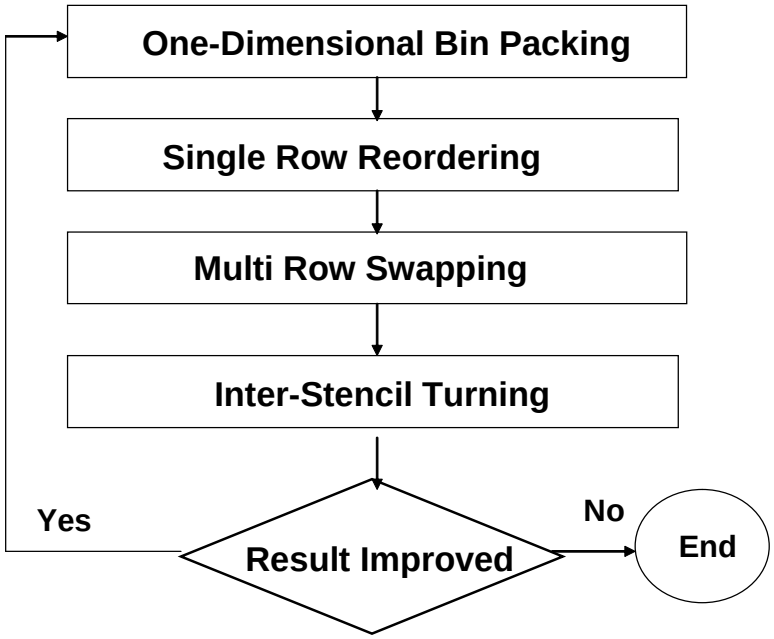


Figure 5.5: The overview of one dimensional stencil design with overlapped characters.

The overview of our four-phase iterative refinement algorithm for this special one-dimension problem is given in Fig. 5.5, and the details will be discussed in following subsections.

5.3.1 Greedy One Dimensional Bin Packing

To construct a reasonable good starting point, we adopt a descending best-fit bin packing algorithm to push the character candidates into stencil, until there is no enough capacity.

Note that the overall projection time (number of shots) of Objective (5.1) can also be represented as

$$\sum_{c_i \in C^C} R_i n_i^{VSB} - \sum_{c_i \in C^{CP}} R_i (n_i^{VSB} - n_i^{CP}) \tag{5.2}$$

where the first part is independent with stencil design. To reduce the processing time, during greedy bin-packing, $\sum_{c_i \in C^{CP}} R_i (n_i^{VSB} - n_i^{CP})$ should be made as large as possible.

Therefore, as preprocessing, we first assign each candidate c_i a profit value p_i, $R_i(n_i^{VSB} - n_i^{CP})$. The bigger p_i is, the larger amount of projection efforts can be reduced by printing c_i using CP than VSB method. For getting good greedy optimization result, the c_i with larger profit should be given higher priority to be placed on the stencil. Guided by this heuristic, in the second step, the character candidates, which have not been on the stencil yet, will be sorted decreasingly based on their profits and packed in a sequential manner.

Next, these sorted candidates will be pushed into stencil by a best-fit packing strategy. When c_i is to be packed, the row, which has the least amount of capacities left *after* accommodating c_i, will be picked. The possible shared space between adjacent objects must also be taken into account, when we are

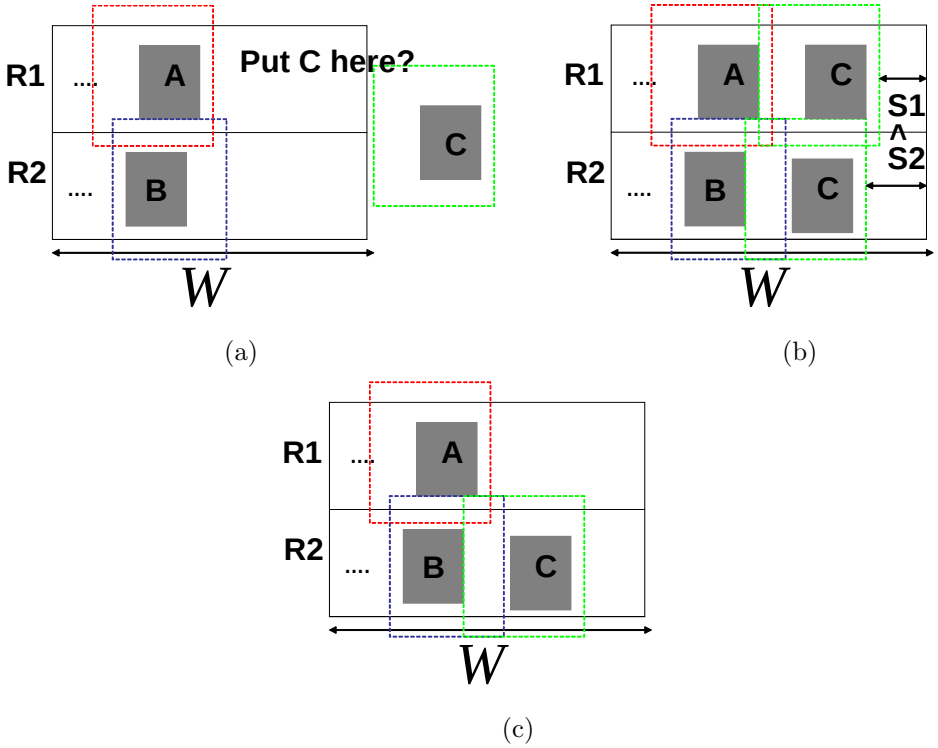


Figure 5.6: This figure illustrates the procedure of best-fit bin packing with overlapping awareness.

computing the remaining room in each row. As Fig. 5.7 (a) illustrates, suppose only two rows are available and candidate C is to be packed next. It appears that row R1 has more capacity left. However, as Fig. 5.7 (b) illustrates, when we try out C in both rows, it is R2 which has larger remaining room. As a result, candidate C is packed into R2, shown by Fig. 5.7 (c).

5.3.2 Single Row Reordering

After greedy bin packing, there is no room left to accommodate more candidates. However, as motivated by Fig. 5.2, we can adjust the relative locations of already-placed characters in each row to shrink its occupied width and increase remaining capacity. This allows pushing in more candidates, which further reduces the overall projection time. Therefore, in this phase, our goal is to minimize the total width of its characters in each row for maximizing remaining capacity.

Suppose row r contains a set of c_0^r .... c_n^r characters from left to right, its total occupied width can be computed as $\sum_{i=0}^{n} w - \sum_{i=0}^{n-1} o_{i,i+1}^H$. It is not difficult to see that $\sum_{i=0}^{n} w$ is a constant as long as the number of characters is not changed. Therefore, to minimize the total occupied width, the overall overlapped blanking margin $\sum_{i=0}^{n-1} o_{i,i+1}^H$ should be maximized.

To compute optimal character permutation for maximum amount of shared blanking width, we formulate a *minimum cost Hamiltonian path* problem. First of all, a graph G is constructed as follows: Each c_i^r is represented by a vertex v_i^r. For each pair of v_i^r and v_j^r, we add two directed edges e_{ij} and e_{ji}. The associated costs are $o_{big}^H - o_{ij}^H$ and $o_{big}^H - o_{ji}^H$, respectively. o_{ij}^H / o_{ji}^H is the shared space when c_i is put left/right adjacent to c_j, and o_{big}^H is a constant value, bigger than any of o_{ij}^H. To maximize $\sum_{i=0}^{n-1} o_{i,i+1}^H$, it suffice to find a path visiting each node of G exactly once such that the total edge weighs $(\sum_{e \in Path}(o_{big}^H - o_{ij}^H))$ along this path is minimized. As Fig. 5.7 (a) illustrates, a graph for three character placement (A,B,C) is given. Suppose the mini-

mum cost Hamiltonian path is found as Fig. 5.7 (b), Fig. 5.7 (c) shows its corresponding character placement.

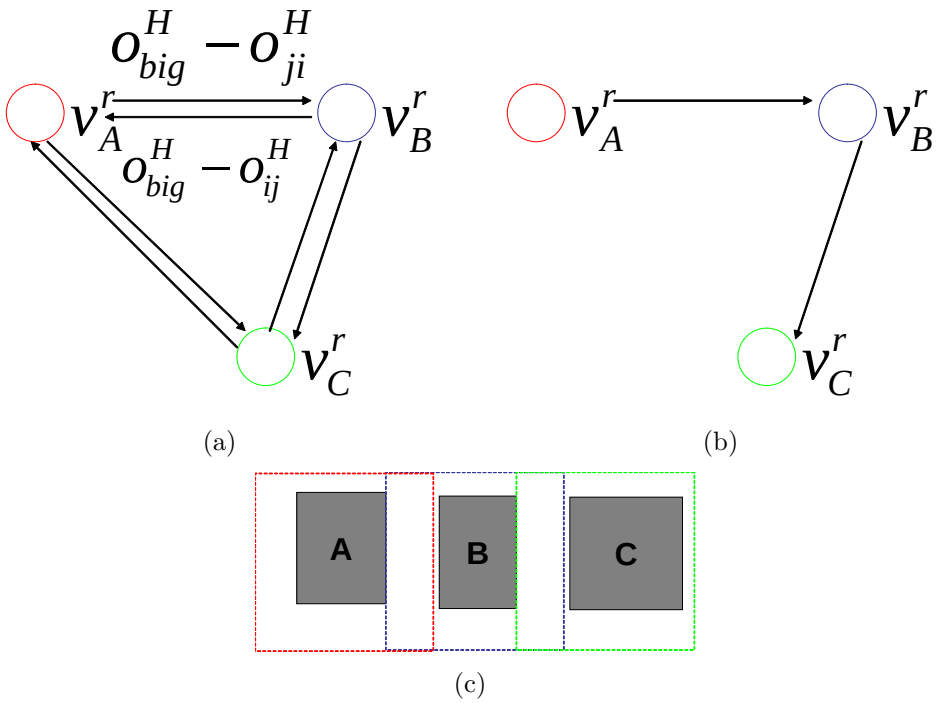
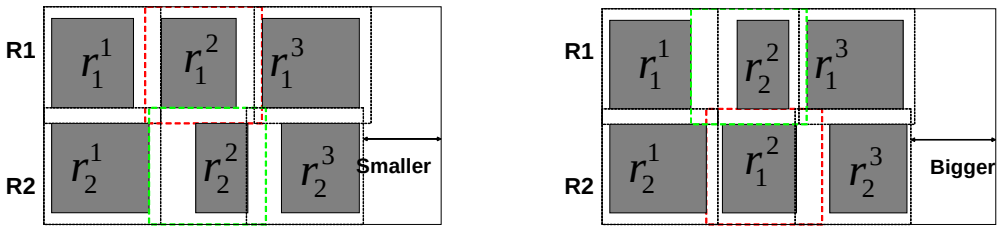


Figure 5.7: This figure shows how to optimize the occupied-width of each row as min-cost Hamiltonian path problem

Practically, since the problem of minimum cost Hamiltonian path is NP-hard, when a row has too many characters, it may be expensive to solve it in one time. In that case, our heuristic is to partition the row into multiple overlapped smaller segments, and solve each segment by Hamiltonian path based method.

5.3.3 Multiple Row Swapping

After single row reordering, the character permutation within each row has been extensively optimized. However, it is still possible to increase their remaining capacities, by swapping characters from different rows. As Fig 5.8 illustrates, by swapping r_1^2 and r_2^2, both characters find "better" neighbors with more overlapped blanking space. For row R1 and R2, their remaining rooms are both increased.



(a) (b)

Figure 5.8: This figure explains the motivation of multi-row swapping.

The algorithm is briefly explained as follows. We test every pair of characters from different rows. Only when the remaining capacities of both rows are increased after swapping, it is considered as a *reasonable* swap. This ensures, the modified placement is definitely better than original one. The reason is that, after swapping, if one row gains more room but another has less, it is possible that the following optimization is hurt by the row with shrunk capacity.

After all the *reasonable* swap pairs are found, they are sorted increas-

ingly by capacity gains, and performed one by one. When certain swap is done, the associated characters and their neighbors are locked. Any swap in the later trials is not allowed to move these locked characters as well as their neighbors. This honors previous optimization result.

5.3.4 Inter Stencil Tuning

The previous single and multiple row optimization are conducted based on the initial solution of bin-packing algorithm in Section 5.3.1. This may limit the optimization space. To get out of local optima, as the last step of ech iteration, we would like to exchange the placed characters with those which have not been selected.

Our approach is to randomly pick and exchange two character candidates, where one is from the stencil and another is not. The swapping will be accepted, only if the overall projection time, number of shots Objective (1), is reduced and the remaining capacity of any row is not shrunk.

5.4 Two Dimensional Stencil Design

In this section, we investigate on the general case of EBL stencil design with overlapped characters. The blanking spaces of templates are non-uniform along both horizontal and vertical directions. Due to NP-completeness of this problem, we adopt a simulated-annealing based heuristic approach to perform a robust iterative improvement.

5.4.1 Sequential Pair Representation

To represent the character placement solution, we make use of sequential pair (SP) proposed in [5].

Given a set of character candidates C^C, its SP consists in two permutations $\overline{X}\&\overline{Y}$ of these templates $(c_0, c_1...c_n)$, which specifies their geometry relationships as below.

$$(\overline{X} :< .., c_i..c_j.. >, \overline{Y} :< ...c_i...c_j... >) : c_i \text{ is left to } c_j \tag{5.3}$$

$$(\overline{X} :< .., c_j..c_i.. >, \overline{Y} :< ...c_i...c_j... >) : c_i \text{ is below } c_j \tag{5.4}$$

Based these constraints, we can map any SP into a solution of character placement as following procedure:

Procedure 1:

Step1: Compute a minimum area packing of C^C, following similar methods of [5, 6]. The detailed consideration will be described in Section 5.4.1.1 and 5.4.1.2.

Step2: Assuming the left-bottom coordinates of packing results and stencil are the same, the candidates, which are located completely within the outline of stencil, are considered as selected characters. □

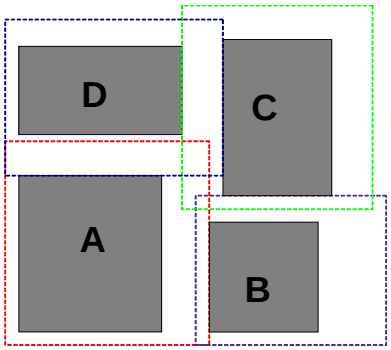
The first step of minimum area packing is the critical one in above transformation. Due to specific properties of our problem, its implementation actually differs from the conventional approaches of [5, 6], explained as follows.

5.4.1.1 Correct Packing Algorithm

The key step of packing solution evaluation from SP is to determine the physical coordinates of each block. This problem has been well investigated, when overlapping is not considered between adjacent blocks. The original algorithm is proposed in [5], and improved by [6] with new solution pruning technique. The work of [5] is extensible for our overlapping-enabled character placement problem. However, the key speed-up idea in [6] does not apply, although it is much faster.

The method of [5] is based on longest path algorithm, and starts from constraint graph construction. Given a SP, a H/V graph is built first to capture the horizontal/vertical relationship between different blocks. Assume there are totally C^C candidates, the H/V graph has $|C^C|$+2 vertexes, one v_i for each candidate c_i plus a source s and sink t. If c_j is (left adjacent to)/(below) c_k, a directed edge e_{jk} is added from v_j to v_k. The weight of e_{jk} is the minimum possible horizontal/vertical distance between the centers of c_j and c_k. Beside these, there is a zero-weight edge from source to every v_i, and a zero-weight edge from every v_i to sink. For the example of Figure 5.9 (a), Figure 5.9 (b) and (c) show the resulting H and V constraint graphes, respectively.

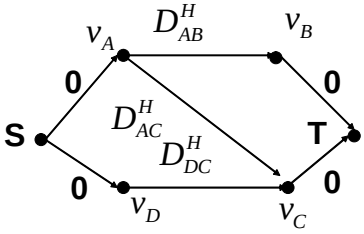
After that, the x/y coordinates of these candidates can be obtained by finding its weighted longest path algorithm from source. As easy to see, this methodology is also applicable for our problem, where overlapped space is allowed between adjacent vertexes. The only difference is that, when the weights of edge are assigned, the amount of shared blanking space must be



$$SP : \overline{X} = (D\ A\ C\ B)$$

$$\overline{Y} = (A\ B\ D\ C)$$

(a)

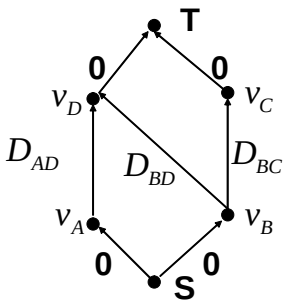


$$D^H_{AB} = \frac{1}{2} w_A + \frac{1}{2} w_B - o^H_{A,B}$$

$$D^H_{AC} = \frac{1}{2} w_A + \frac{1}{2} w_C - o^H_{A,C}$$

$$D^H_{DC} = \frac{1}{2} w_D + \frac{1}{2} w_C - o^H_{D,C}$$

(b) H graph



$$D^V_{BC} = \frac{1}{2} h_B + \frac{1}{2} h_C - o^V_{B,C}$$

$$D^V_{BD} = \frac{1}{2} h_B + \frac{1}{2} h_D - o^V_{B,D}$$

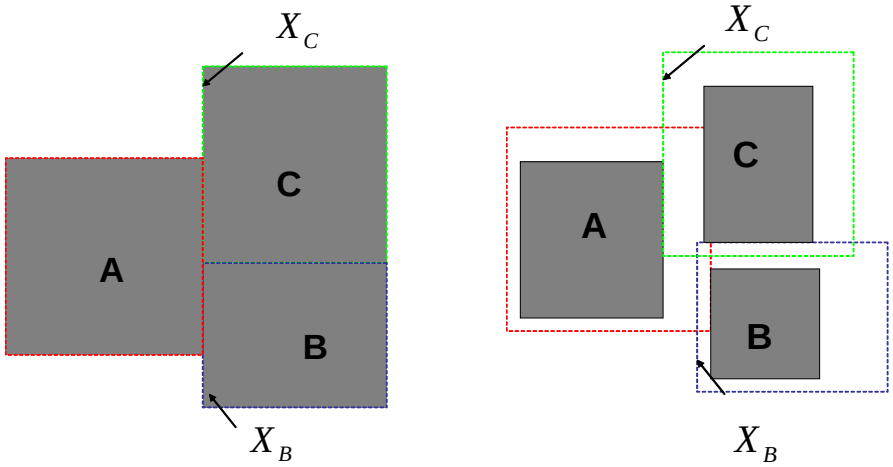
$$D^V_{AD} = \frac{1}{2} h_A + \frac{1}{2} h_D - o^V_{A,D}$$

(c) V graph

Figure 5.9: This figure explains packing evaluation of [5] based on sequential pair

considered, as highlighted by the red cycles in Figure 5.9 (b) and (c).

On the other side, the work of [6] does not explicitly build the constraints graphs but depends on the longest common subsequence computations. They evaluate the placement of character candidates much faster than [5], depending on the following property.



(a) non-overlapping (b) overlapping

Figure 5.10: This figure illustrates the key idea of of [6]

Property 5.4.1. Given two blocks B and C, if we put them (right adjacent)/up to a common component A, then the x/y coordinates of these two blocks should be same.

The correctness of this property can be easily seen for the conventional packing, as shown by Figure 5.10 (a), while overlapping is not considered.

However, it does not hold true, when the sharing of characters becomes possible. As Figure 5.10 (b) illustrates, due to different overlapping margins, the coordinates of B and C are not the same.

5.4.1.2 Fast Packing Evaluation

After evaluating packing solution, in the step2 of Procedure 1, the candidates outside the outline of stencil will not be taken as characters. This implies, the detailed locations of these candidates are not important, and do not have to be computed in the step1. Great speedup can be achieved by making use of this property.

In detail, in the implementation of SP-based minimum area packing, we stop placement evaluation as soon as the contour of already-packed character candidates is completely outside the outline of stencil by at least a margin of o_{max}, given that o_{max} is the maximum value of o_{ij}^{H} and o_{ij}^{V}.

This strategy will not effect the solution of character placement. For any of unpacked candidates by the stopping time, it can not be totally fit into the stencil no matter how to push it around the boundaries of already-packed character clusters.

5.4.2 Simulated Annealing

During simulated annealing, we continuously make small modification on sequential pair, and evaluate the resulting stencil design. The new SP/solution will be for sure adopted if reducing the total time of Objective (5.1). While it

is actually a worse character placement, this non-improving result is accepted with probability decreasing over time.

In this subsection, we present two effective SP perturbation methods for better local search towards higher throughput: timing-driven swapping and slack-based insertion.

5.4.2.1 Timing-driven Swapping

The first type of perturbation we perform is timing-driven swapping. The basic idea is to try reducing overall projection time by swapping the positions of two candidates in the $\overline{X}\&\overline{Y}$ SP. This is equivalent to exchange their relative locations in the packing solution.

Fig. 5.11 illustrates a motivational example, which has five blocks A-E to be packed. The required number of shots, to project any of these candidates once, are assumed as 1 and 10 for CP (n_i^{CP})and VSB (n_i^{VSB}) methods respectively. The digit in the parentheses denotes how many times R_i of each component will be used and printed in the design.

Fig. 5.11 (a) gives a SP representation and its corresponding stencil design, based on the Procedure 1 in Section 5.4.1. Following the definition of Objective 5.1, the total processing time (number of shots) are $3+2+1+2+10\times 2=28$, since A-D are selected as characters while E is not. If swapping the locations of C and E in SP as Fig. 5.11 (b), we would end up with a better stencil design with less amount processing time. It only takes a number of 19 shots, which is computed as $3+2+10+2+2=19$, in this case.

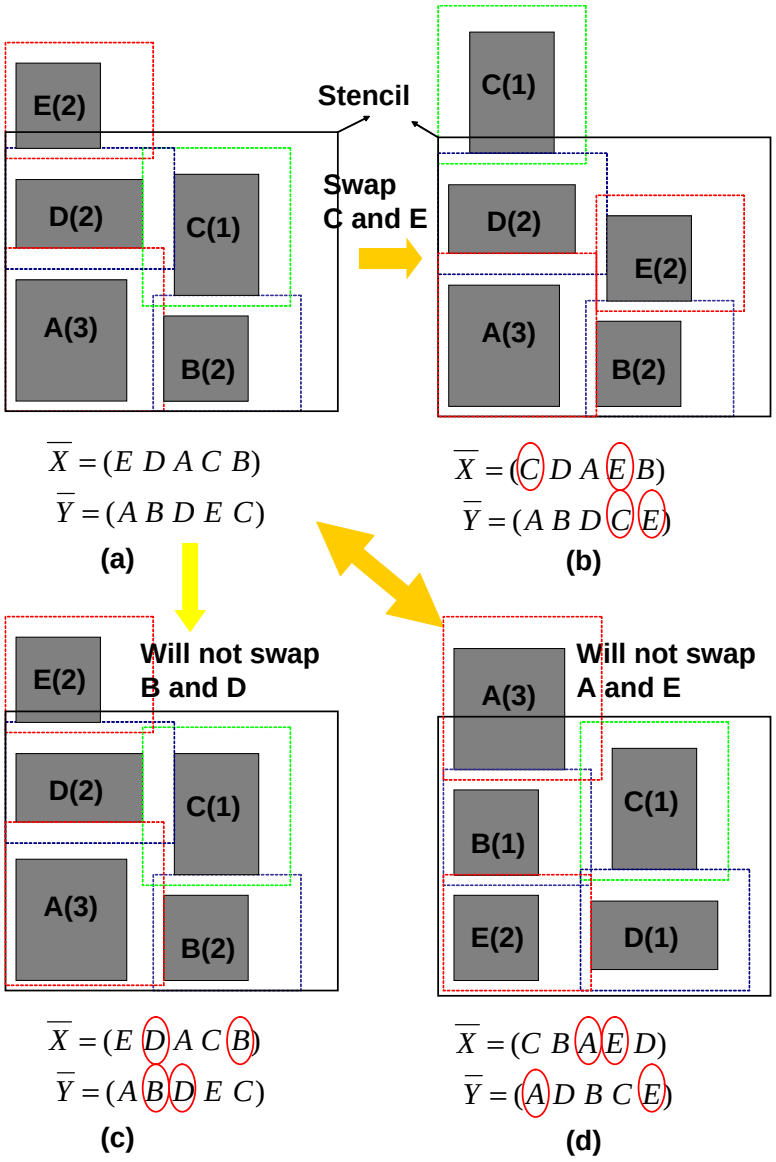


Figure 5.11: The figure illustrates timing-driven swapping.

In the detailed implementation, we enforce two heuristic swapping constraints, to enable efficient and effective shot number reduction.

First of all, given a SP, out of the pair of elements to be changed, we require that one candidate c_s, should have been selected as characters by its corresponding stencil packing, while the other one, c_o is not. For the example of Fig. 5.11 (a), we only allow the exchange of the positions between E and any of A-D. The swapping among any two of A-D is not enabled. The reason is that if the two candidates to be swapped are both in or out of stencil already, most likely the new SP generates a stencil solution with same set of selected characters and just different geometrical ordering. As an example, if we swap candidate B and D which are both already in the stencil, like from Fig. 5.11 (a) to Fig. 5.11 (c), the resulting packing result also selects A-D as characters, still requiring 28 shots totally.

Secondly, after randomly picking in-stencil candidate c_s and out-of-stencil one c_o for swapping, to decide whether this swapping would be tried on. The profit p_o/p_s is defined as same as $R_i(n_i^{VSB} - n_i^{CP})$ in Section 5.3.1, which reflects the reduction of the shoot number by printing this candidate by CP rather than VSB. If we swap the locations of c_s and c_o, it is highly like that c_s will be pushed out of stencil but the c_o would be selected as character in turn. Assuming all the other candidates stay either in or outside the stencil, as the state before the swapping, the total shot reduction by this exchange can be approximated as p_o-p_s. Therefore, if the difference p_o-p_s is smaller than zero, it is in high possibility that the swapping under consideration will not lead to

better packing result. For the example of Fig. 5.11 (a), suppose c_s and c_o are A and E, respectively, it turns out p_o-p_s is -9. In this case, and the corresponding stencil design indeed becomes worse, taking 35 shots as Fig. 5.11 (d) shows.

5.4.2.2 Slack-based insertion

Given a SP and its corresponding character solution, our purpose of slack-based insertion is to add-in a new candidate, which currently is not serving as character, into the stencil. To ensure robust throughput improvement, we would like to find a good strategy to insert such extra candidate, so that all the previously already-placed characters are still kept on the stencil in most trials. This equals to increase the number of usable templates. In this subsection, we make use of the concept of slack, applied in [79], to search such a good insertion location.

Given a character c_s on the stencil, its x/y slack is defined as the allowed movement range of x/y coordinates of c_s, under the constraint that none of all the other already-placed characters would be pushed outside the stencil after such move. Fig. 5.12 (a)-(b) illustrate a simple example, with four characters A-D. Their leftmost and rightmost packing solutions are shown by Fig. 5.12 (a) and (b), respectively. Based on this two extreme cases, the x slack of C, for example, can be computed as $X_c^{right} - X_c^{left}$.

Once slacks are known, we randomly pick a *base* character c_b, which has large slacks in both x and y directions, and insert a new candidate c_{new} before it. The reason is that the location of such *base* can be moved in relatively big

amount to make space for additional character. In terms of SP operation, this can be done by simply changing the position of c_{new} right before c_b in $\overline{X}$ and $\overline{Y}$ permutations. As illustrated by Fig. 5.12 (c), suppose the c_b and c_{new} are candidate C and E, respectively. The resulting new SP is obtained by insert E right in front of the position of C, as shown by Fig. 5.12 (d) .

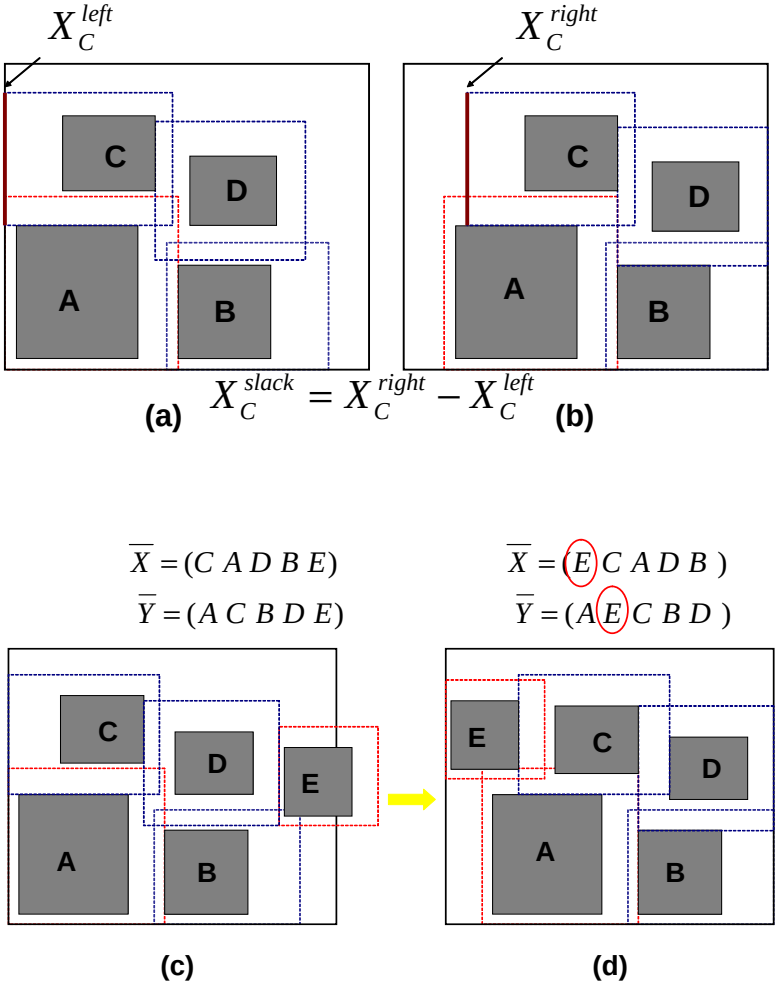


Figure 5.12: The simple example of slack-based insertion.

5.5 Experimental Results

We implement our algorithm in C++ and test on Intel Core 3.0GHz Linux machine with 32G RAM. LKH [80] is chosen as the solver for min-cost Hamilton path. Moreover, Parquet [79] is adopted as our simulated annealing framework.

To test the efficiency of proposed methods, we randomly generate eight benchmarks. The size of stencil is set as 100um x 100um, and a total number of 1000 character candidates with unique size are generated. The sharable blanking area within each candidate is randomly decided. For the special case of one dimensional problem, the blanking space along vertical direction is set as a constant value. Moreover, for each candidate c_i, we randomly assign a triple of value (R_i, n_i^{VSB}, n_i^{CP}) as its referred time in chip, and respective number of shots by VSB and CP. n_i^{VSB} is made 5-10x larger than n_i^{CP}.

The detailed statistical data for individual testcase is shown in Table 5.1. The first column denotes the name of benchmarks, which "1D-x" and "2D-x" are applied for one and two dimensional problem, respectively. "csize" is the size of each character candidate, formatted by "um x um". The units of all the other columns are "$1e^4um^2$". "total area" shows the total area of all the character candidates, and "total blanks" is the summation of their sharable blanking space. "optimal area" is computed as "total area" minus "total blanks", typically larger than the area of given stencil. This matches the fact that even under best possible case of stencil design, where all the blanking area are indeed shared by adjacent characters, the entire set of the

candidates can not be fully pushed into the stencil.

Table 5.1: Statistics on testcases.

ckts	csize	total area	total blanks	optimal area
1D-1	3.8x3.8	1.444	0.416	1.028
1D-2	4.0x4.0	1.6	0.479	1.121
1D-3	4.2x4.2	1.764	0.514	1.25
1D-4	4.4x4.4	1.936	0.569	1.367
2D-1	3.8x3.8	1.444	0.414	1.03
2D-2	4.0x4.0	1.6	0.529	1.071
2D-3	4.2x4.2	1.764	0.662	1.102
2D-4	4.4x4.4	1.936	0.774	1.162

For comparative reason, we implement two different stencil design approaches. The first one **NO-OVERLAP** is based on the work of [78], where no overlapped characters is allowed. A little difference is that, in its implementation, only one stencil with unique character size is considered. Moreover, for our problem, their algorithm is somewhat degenerated into a method of selecting the most profitable candidates, which profit is judged by $R_i(n_i^{CP} - n_i^{VSB})$. In the second comparative approach **GREEDY**, possible sharing is taken into account, but a greedy methodology is applied to chose character candidates. In 1D problem, only the first phase of heuristic Descending Best-Fit (DBF) packing in Section 5.3.1 is performed. For two-dimensional problem, 2D DBF packing is conducted.

5.5.1 One-dimensional Stencil Design

Table 5.2 lists the comparison of stencil design in one-dimensional case. "#shot" shows the total processing time (number of shots) of the circuit by

using corresponding stencil design methodologies, which is computed by the equation of Objective 5.1. "#char" is the number of characters that fits into stencils, and "#CPU" tells the runtime of these stencil optimization methods, in terms of seconds.

As we can see, compared to **NO-OVERLAP**, we are able to averagely put 42% more characters on the stencil, and reduce the total projection time (number of shots) by 51%. With respect to **GREEDY** algorithm, our approach still achieves averagely 14% more projection time reduction, by allowing 7% more characters placed. The CPU time of our approach is relatively large but its absolute value is only around 20s. These results show the effectiveness and efficiency of our proposed four-phase iterative refinement algorithm.

For this special one-dimension problem, **GREEDY** looks also quite useful. The reason is that the vertical blanking spaces of these candidates are uniform in this case, and have been fully shared during the stencil design.

Table 5.2: Result Comparison for 1D problem

	NO-OVERLAP			GREEDY			our approach		
ckts	#shot	#char	$CPU(s)$	#shot	#char	CPU	#shot	#char	CPU
1D-1	28654	676	1.2	13528	901	2.2	10083	951	22.3
1D-2	41727	625	1.1	17929	836	2.1	14921	880	21.8
1D-3	38460	529	0.9	25155	727	1.9	22503	768	20.6
1D-4	41260	484	0.8	29462	665	1.8	26756	702	20.1
total	150101	2314	4	86074	3129	8	74263	3301	84.8
ratio	2.0	1	0.05	1.16	1.35	0.10	1	1.42	1

5.5.2 Two-dimensional Stencil Design

Table 5.3 lists the comparison of stencil design in general two-dimensional case. The meaning of labels are the same as Table 5.2. Compared to **NO-OVERLAP** and **GREEDY** methods, in average, our proposed SP-based algorithm places 28% and 24% more characters on stencil, which reduces the projection time (number of shots) by 31% and 25%, respectively. The **GREEDY** algorithm does not work that well in this 2D problem, because the blanking area varies in both horizontal and vertical directions and the native first-bin-best-fit packing very easily get stuck in local optima.

Due to two-dimensional optimization, the runtime of our approach is much longer than 1D problem, comparatively. It takes a few hundred seconds, but is still satisfactory. The design of stencil is only a one-time job before projecting large volume of chips by EBL. Several minutes preprocessing time is relatively very tiny in the whole manufacturing procedure.

Table 5.3: Result Comparison for 2D problem

	NO-OVERLAP			GREEDY			our approach		
ckts	#shot	#char	CPU	#shot	#char	CPU	#shot	#char	CPU
2D-1	23319	676	1.3	26832	625	2.3	16877	803	466
2D-2	29368	576	1	25977	642	2.6	20141	750	447
2D-3	32399	526	0.9	30411	558	2.5	23850	688	424
2D-4	35410	474	0.8	31930	531	2.7	25278	660	416
total	120496	2252	4	115150	2356	10.1	86146	2901	1755
ratio	1.40	1	0.002	1.33	1.05	0.006	1	1.30	1

5.6 Summary

In this chapter, we have developed two algorithms for overlapping aware stencil design in electronic beam lithography. The experimental results show 51% reduction on the total projection time, compared to the conventional design when the characters are not overlapped.

Chapter 6

Conclusion

This dissertation investigates on physical design automation problems for double patterning and electronic beam lithography, which are summarized as follows:

- We develop a high quality post-routing layout decomposer for double patterning lithography. Our key feature is *simultaneous* optimization of conflict and stitch minimization, powered by practical grid model and integer linear programming. This delivers much better global solution than dealing with conflict and stitch in separate phases as previous works do. To improve the scalability, three speed-up techniques are also proposed.

- We next study double patterning friendly detailed routing algorithm with redundant via consideration in the first time. If not carefully planned and optimized, the additional metal, the double via introduces, could result in native conflicts or stitches. To encounter this problem, during detailed routing, double patterning related cost are enforced in maze searching for avoiding the generation of DPL-unfriendly redundant via candidates.

- We further make use of wire spreading technique to enhance the decomposability of double patterning mask assignment. A set of wire spreading candidates are pre identified to help eliminating conflicts or reducing stitches, and then modeled into a decomposition graph. An integer linear programming formulation is proposed next to perform coloring and layout modification together. Efficient graph reduction techniques are developed as well for reducing the runtime.

- We develop a stencil planning and optimization framework for improving the throughput of electronic beam lithography. The key idea is to increase the number of characters available in the stencil, by allowing overlapping the blanking space between adjacent characters. A Hamilton path-based flow has been proposed for specific one-dimensional packing problem, and a simulated annealing framework is developed for two-dimensional fixed-outline floorplanning.

Some of the further working directions could be:

- There are two major types of double patterning lithography: Lith-Etch-Lith-Etch (LELE) [81, 82] and Self-Aligned (SA) [83, 84] technologies. All the three DPL-related works in this dissertation focus on LELE. Future research could be investigated on self-aligned double patterning layout decomposition problem. Different from LELE, in SA double patterning, there are totally three masks, not two. Moreover, no stitches can be introduced due to its process constraints, putting less control on

overlay control than LELE [85–91]. It is also a good direction to investigate earlier stage placement and standard cell design [92] to produce DPL-friendly layout for SA double patterning.

- The framework of my electronic beam lithography stencil optimization can be extended for more generalized cases. Multiple-stencil multiple-design co-optimization would be further investigated. Moreover, the characters in the stencil do not have to be regular rectangles.

- Physical design related optimization are also needed to be studied for other emerging nanolithography systems, such as extreme ultra-violet [93, 94], nanoimprint lithography [95–101], and triple patterning techniques [102].

Bibliography

[1] Tsann-Bim Chiou, Robert Socha, Hong Chen, Luoqi Chen, Stephen Hsu, Peter Nikolsky, Anton van Oosten, and Alek C. Chen. Development of layout split algorithms and printability evaluation for double patterning technology. In *Proc. of SPIE*, volume 6924, 2008.

[2] M. Drapeau, V. Wiaux, E .Hendrickx, S. Verhaegen, and T. Machida. Double patterning design split implementation and validation for the 32nm node. In *Proc. of SPIE*, volume 6521, 2007.

[3] C. Christiansen, B. Li, J. Gill, R. Filippi, and M. Angyal. Via-depletion Electromigration in Copper Interconnects. In *IEEE Transactions on Device and Materials Reliability*, June 2006.

[4] Minsik Cho, Yongchan Ban, and David Z. Pan. Double patterning technology friendly detailed routing. In *Proc. Int. Conf. on Computer Aided Design*, October 2008.

[5] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. VLSI Module Placement Based on Rectangle-Packing by theSequence-Pair. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Decemember 1996.

[6] Xiaoping Tang, Ruiqi Tian, and Martin Wong. Fast Evaluation of Sequence Pair in Block placement by Longest Common Subsequence Computation. In *Proc. Design, Automation and Test in Eurpoe*, March 2000.

[7] G. E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38:114–117, April 1965.

[8] S. Wolf and R. N. Tauber. *Silicon Processing for the VLSI Era, Vol. 1: Process Technology.* Lattice Press, 1999.

[9] C. A. Mack. *Fundamental principles of optical lithography : the science of microfabrication,* Wiley, Janunary 2007.

[10] C. A. Mack. IEEE Spectrum: Seeing double. In *IEEE Spectrum*, November 2008.

[11] Burn J. Lin. The k3 coefficient in nonparaxial lambda/NA scaling equations for resolution, depth of focus, and immersion lithography. In *Journal of Microlithography, Microfabrication, and Microsystems*, April 2002.

[12] Timothy A. Brunner. Why optical lithography will live forever. In *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, November 2003.

[13] Alfred K. Wong. Microlithography: Trends, challenges, solutions, and their impact on design. In *IEEE Micro.*, March 2003.

[14] Peng Yu and David Z. Pan. TIP-OPC: A New Topological Invariant Paradigm for Pixel Based Optical Proximity Correction. In *Proc. Int. Conf. on Computer Aided Design*, November 2007.

[15] Peng Yu and David Z. Pan. A Novel Intensity Based OPC Algorithm with Speedup in Lithography Simulation. In *Proc. Int. Conf. on Computer Aided Design*, November 2007.

[16] Chris A. Mack. *Field Guide to Optical Lithography,* SPIE Press, 200*6*.

[17] D. Gil, T. Bailey, D. Corliss, M. J. Brodsky, P. Lawson, M. Rutten, Z. Chen, N. Lustig, and T. Nigussie. First microprocessors with immersion lithography. In *Proc. of SPIE*, volume 5274, 2004.

[18] R. H. French, H. Sewell, M. K. Yang, S. Peng, D. McCafferty, W. Qiu, R. C. Wheland, M. F. Lemon, L. Markoya, and M. K. Crawford. Imaging of 32-nm 1:1 lines and spaces using 193-nm immersion interference lithography with second-generation immersion fluids to achieve a numerical aperture of 1.5 and a k[sub 1] of 0.25. In *Journal of Microlithography, Microfabrication, and Microsystems*, July 2005.

[19] L. W. Liebman. Layout impact of resolution enhancement techniques: impediment or opportunity? In *Proc. Int. Symp. on Physical Design*, pages 110–117, 2003.

[20] F. M. Schellenberg. *Resolution Enhancement Technology: The past, the present and extension for the future.* Mentor Graphics White Paper,

2004 SPIE Microlithography Plenary.

[21] R. Socha, X. Shi, and D. LeHoty. Simultaneous source mask optimization. In *Proc. of SPIE*, volume 5853, 2005.

[22] P. Yu, S. X. Shi, and D. Z. Pan. True Process Variation Aware Optical Proximity Correction with Variational Lithography Modeling and Model Calibration. In *Journal of Microlithography, Microfabrication, and Microsystems*, September 2007.

[23] A. J. Strojwas, T. Jhaveri, V. Rovner, and L. Pileggi. Creating an affordable 22nm node using design lithography co-optimization. In *Proc. Design Automation Conf.*, July 2009.

[24] S. Lee, M. Chandhok, J. Roberts, and B. Rice. Characterization of flare on intels euv met. In *Proc. of SPIE*, volume 5751, 2005.

[25] F. Schellenberg, J. Word, and O. Toublan. Layout Compensation for EUV Flare. In *Proc. of SPIE*, volume 5751, 2005.

[26] J. Moon, C. Kim, B. Nam, B. Nam, Y. Hyun, S. Kim, C. Lim, Y. Kim, M. Kim, Y. Choi, C. Kim, and D. Yim. Evaluation of shadowing and flare effect for EUV tool. In *Proc. of SPIE*, volume 7271, 2009.

[27] C. Koh, L. Ren, J. Georger, F. Goodwin, S. Wurm, B. Pierson, J. Park, T. Wallow, T. Younkin, and P. Naulleau. Assessment of EUV resist readiness for 32nm hp manufacturing, and extendibility study of EUV ADT using state-of-the-art resist. In *Proc. of SPIE*, volume 7271, 2009.

[28] M. Maenhoudt, J. Versluijs, H. Struyf, J. Van Olmen, and M. Van Hove. Double patterning scheme for sub-0.25 k1 single damascene structures at NA=0.75, λ=193nm. In *Proc. of SPIE*, volume 5754, 2005.

[29] Mircea Dusa, John Quaedackers, Olaf F. A. Larsen, Jeroen Meessen, van der Heijden Eddy, Gerald Dicker, Onno Wismans, Paul de Haas, Koen van Ingen Schenau, Jo Finders, Bert Vleeming, Geert Storms, Patrick Jaenen, Shaunee Cheng, and Mireille Maenhoudt. Pitch doubling through dual-patterning lithography challenges in integration and litho budgets. In *Proc. of SPIE*, volume 6520, 2007.

[30] Seo-Min Kim, Sun-Young Koo, Jae-Seung Choi, Young-Sun Hwang, Jung-Woo Park, Eung-Kil Kang, Chang-Moon Lim, Seung-Chan Moon, and Jin-Woong Kim. Issues and challenges of double patterning lithography in dram. In *Proc. of SPIE*, volume 6520, 2007.

[31] W. H. Arnold. Guest editorial: Special section on double-patterning lithography. In *Journal of Micro/Nanolithography, MEMS and MOEMS*, March 2009.

[32] A. J. Hazelton, S. Wakamoto, S. Hirukawa, M. McCallum, N. Magome, J. Ishikawa, C. Lapeyre, I. Guilmeau, S. Barnola, and S. Gaugiran. Double-patterning requirements for optical lithography and prospects for optical extension without double patterning. In *Journal of Micro/Nanolithography, MEMS and MOEMS*, Janunary 2009.

[33] B. J. Lin. Making double patterning cost single. In *Journal of Micro/Nanolithography, MEMS and MOEMS*, March 2009.

[34] SPIE panel: DP is only litho solution for 22nm volume production. 2009.

[35] Kun Yuan, Jae-Seok Yang, and David Z. Pan. Double patterning layout decomposition for simultaneous conflict and stitch minimization. In *Proc. Int. Symp. on Physical Design*, March 2009.

[36] Kun Yuan, Katrina Lu, and David Z. Pan. Double patterning lithography friendly detailed routing with redundant via consideration. In *Proc. Design Automation Conf.*, July 2009.

[37] Kun Yuan and David Z. Pan. WISDOM: Wire Spreading Enhanced Decomposition of Masks in Double Patterning Lithography. In *Proc. Int. Conf. on Computer Aided Design*, November 2010.

[38] Christian K. Kalus. Towards systematic CD process control for e-beam lithography. In *Proc. of SPIE*, volume 5504, 2004.

[39] B. Cord, J. Yang, D. Joy, J. Klingfus, and K. Berggren. Limiting factors in sub-10-nm scanning electron beam lithography. In *Int. Conf. on Electron, Ion and Photon Beam Technology and Nanofabrication (EIPBN),* 2009.

[40] M. Wieland, G. Boer, G. Berge, R. Jager, T. Peut, J. Peijster, E. Slot, S. Steenbrink, T. Teepen, A. Veen, and B. Kampherbeek. MAPPER:

high-throughput maskless lithography. In *Proc. of SPIE*, volume 7378, 2009.

[41] Aki Fujimur. Beyond Light: The Growing Importance of E-Beam. In *Proc. Int. Conf. on Computer Aided Design*, November 2009.

[42] Aki Fujimur. Design for E-Beam: Getting the Best Wafers Without the Exploding Mask Costs. In *Proc. Int. Symp. on Quality Electronic Design*, March 2010.

[43] Akira Fujimura, Takashi Mitsuhashi, Kenji Yoshida, Shohei Matsushita, Larry Lam Chau, Tam Dinh Thanh Nguyen, and Donald MacMillen. Stencil Design and Method for Improving Character Density for Cell Projection Charged Particle Beam Lithography. Jan. 2010.

[44] Hans Co Pfeiffer. New Prospects for Electron Beams as Tools for Semiconductor Lithography. In *Proc. of SPIE*, volume 7378, 2009.

[45] G. Bailey, A. Tritchkov, J. Park, L. Hong, V. Wiaux, E. Hendrickx, S. Verhaegen, P. Xie, and J. Versluijs. Double pattern EDA solutions for 32nm HP and beyond. In *Proc. of SPIE*, volume 6521, 2007.

[46] Y. Inazuki, N. Toyama, T. Nagai, T. Sutou, Y. Morikawa, H. Mohri, N. Hayashi, M. Drapeau, K. Lucas, and C. Cork. Decomposition difficulty analysis for double patterning and the impact on photomask manufacturability. In *Proc. of SPIE*, volume 6925, 2008.

[47] C. Bencher, Y. Chen, H. Dai, W. Montgomery, and L. Hul. 22nm Half-Pitch Patterning by CVD Spacer Self Alignment Double Patterning (SADP). In *Proc. of SPIE*, volume 6924, 2008.

[48] Gianfranco Capetti, Pietro Cant, Elisa Galassini, Alessandro Vaglio Pret, Alessandro Vaccaro Catia Turco, Pierluigi Rigolli, Fabrizio D'Angelo, and Gina Cotti. Sub-k1 = 0.25 lithography with double patterning technique for 45-nm technology node flash memory devices at = 193nm. In *Proc. of SPIE*, volume 6520, 2007.

[49] A. Vanleenhove and D. Steenwinckel. A litho-only approach to double patterning. In *Proc. of SPIE*, volume 6521, 2007.

[50] V. Wiaux, S. Verhaegen, S. Cheng, F. Iwamoto, P. Jaenen, M. Maenhoudt, T. Matsuda, S. Postnikov, and G. Vandenberghe. Split and design guidelines for double patterning. In *Proc. of SPIE*, volume 6924, 2008.

[51] W. Arnold. Towards 3nm overlay and critical dimension uniformity: an integrated error budget for double patterning lithography. In *Proc. of SPIE*, volume 6924, 2008.

[52] J. Rubinstein and A. Neureuther. Post-decomposition assessment of double patterning layout. In *Proc. of SPIE*, volume 6924, 2008.

[53] Andrew B. Kahng, Chul-Hong Park, Xu Xu, and Hailong Yao. Layout decomposition for double patterning lithography. In *Proc. Int. Conf.*

on Computer Aided Design, November 2008.

[54] http://www.gnu.org/software/glpk/glpk.html/.

[55] http://glaros.dtc.umn.edu/gkhome/views/metis.

[56] Kuang-Yao Lee, Cheng-Kok Koh, Ting-Chi Wang, and Kai-Yuan Chao. Optimal Post-Routing Redundant Via Insertion. In *Proc. Int. Symp. on Physical Design*, April 2008.

[57] Huang-Yu Chen, Mei-Fang Chiang, Yao-Wen Chang, Lumdo Chen, and Brian Han. Novel Full-Chip Gridless Routing Considering Double-Via Insertion. In *Proc. Design Automation Conf.*, July 2006.

[58] Hailong Yao, Yici Cai, Qiang Zhou, and Xianlong Hong. Improved Multilevel Routing with Redundant Via Placement for Yield Improvement. May 2005.

[59] K. McCullen. Redundant Via Insertion in Restricted Topology Layout. In *Proc. Int. Symp. on Quality Electronic Design*, March 2007.

[60] Lun-Chun Wei, Hung-Ming Chen, Li-Da Huang, and Sarah Songjie Xu. Efficient and Optimal Post-Layout Double-Cut Via Insertion by Network Relaxation and Min-Cost Maximum Flow. May 2008.

[61] Cheok-Kei Lei, Po-Yi Chiang, and Yu-Min Lee. Post-Routing Redundant Via Insertion with Wire Spreading Capability. In *Proc. Asia and South Pacific Design Automation Conf.*, Jan 2009.

[62] Kuang-Yao Lee, Shing-Tung Lin, and Ting-Chi Wang. Redundant Via Insertion with Wire Bending. In *Proc. Int. Symp. on Physical Design*, March 2009.

[63] Chih-Ta Lin, Yen-Hung Lin, Guan-Chan Su, and Yih-Lang Li. Dead Via Minimization by Simultaneous Routing and Redundant Via Insertion. In *Proc. Asia and South Pacific Design Automation Conf.*, January 2010.

[64] *Taiwan Semiconductor Manufacturing Company (TSMC). Reference Flow 5.0 and Reference Flow 6.0.*

[65] G. Xu, L. Huang, D. Z. Pan, and D. F. Wong. Redundant-Via Enhanced Maze Routing for Yield Improvement. In *Proc. Asia and South Pacific Design Automation Conf.*, January 2005.

[66] Yue Xu and Chris Chu. GREMA: Graph Reduction based Efficient Mask Assignment for double patterning lithography. In *Proc. Int. Conf. on Computer Aided Design*, November 2009.

[67] Jae-Seok Yang, Katria Lu, Minsik Cho, Kun Yuan, and David Z. Pan. A New Graph-theoretic, Multi-objective Layout Decomposition Framework for Double Patterning Lithography. In *Proc. Asia and South Pacific Design Automation Conf.*, January 2010.

[68] Andrew B. Kahng, Chul-Hong Park, Xu Xu, and Hailong Yao. Revisiting the layout decomposition problem for double patterning lithography.

In *Proc. of SPIE*, October 2008.

[69] Yue Xu and Chris Chu. A Matching Based Decomposer for Double Patterning Lithography. In *Proc. Int. Symp. on Physical Design*, March 2010.

[70] Xin Gao and Luca Macchiarulo. Enhancing Double-Patterning Detailed Routing With Lazy Coloring and Within-Path Conflict Avoidance. In *Proc. Design, Automation and Test in Eurpoe*, March 2010.

[71] Chin-Hsiung Hsu, Yao-Wen Chang, and Sani R. Nassif. Simultanous layout migration and decomposition for double patterning technology. In *Proc. Int. Conf. on Computer Aided Design*, November 2009.

[72] R. Tarjan. Depth-first search and linear graph algorithms. In *SIAM J. Comput.*, vol. 2 1972.

[73] http://www.si2.org/?page=69.

[74] Takeshi Fujino, Yoshihiko Kajiya, and Masaya Yoshikawa. Character-build standard-cell layout technique for high-throughput character-projection EB lithography. In *Proc. of SPIE*, volume 5853, 2005.

[75] M. Sugihara, T. Takata, K. Nakamura, R. Inanami, H. Hayashi, K. Kishimoto, T. Hasebe, Y. Kawano, Y. Matsunaga, K. Murakami, and K. Okumura. Technology mapping technique for throughput enhancement of character projection equipment. In *Proc. of SPIE*, volume 6151, 2006.

[76] Makoto Sugihara, Kenta Nakamura, Yusuke Matsunaga, and Kazuaki Murakami. CP mask optimization for enhancing the throughput of MCC systems. In *Proc. of SPIE*, volume 6349, 2006.

[77] Yusuke Matsunaga Makoto Sugihara and Kazuaki Murakami. Technology mapping technique for enhancing throughput of multi-column-cell systems. In *Proc. of SPIE*, volume 6517, 2007.

[78] Makoto Sugihara. Optimal character-size exploration for increasing throughput of MCC lithographic systems. In *Proc. of SPIE*, volume 7271, 2009.

[79] Saurabh H. Adya and Igor L. Markov. Fixed-outline Floorplanning : Enabling Hierarchical Design. In *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Decemember 2003.

[80] http://www.akira.ruc.dk/ keld/research/LKH.

[81] Tomoyuki Ando, Masaru Takeshita, Ryoich Takasu, Yasuhiro Yoshii, Jun Iwashita, Shogo Matsumaru, Sho Abe, and Takeshi Iwai. Pattern Freezing Process Free Litho-Litho-Etch Double Patterning. In *Proc. of SPIE*, volume 7140, 2008.

[82] Hisanori Sugimachi, Hitoshi Kosugi, Tsuyoshi Shibata, Junichi Kitano, Koichi Fujiwara, Michihiro Mita, Akimasa Soyano, Shiro Kusumoto, Motoyuki Shima, and Yoshikazu Yamaguchi. CD Uniformity improvement

for Double-Patterning Lithography (Litho-Litho-Etch) Using Freezing Process. In *Proc. of SPIE*, volume 7273, 2009.

[83] Weicheng Shiu, Hung Jen Liu, Jan Shiun Wu, Tsu-Li Tseng, Chun Te Liao, Chien Mao Liao, Jerry Liu, and Troy Wang. Advanced self-aligned double patterning development for sub-30-nm DRAM manufacturing. In *Proc. of SPIE*, volume 7274, 2009.

[84] Y.-S. Chang, M.-F. Tsai, C-C Lin, and J-C Lai. Pattern decomposition and process integration of self-aligned double patterning for 30nm node nand flash process and beyond. In *Proc. of SPIE*, volume 7274, 2009.

[85] W.-K. Ma, J.-H. Kang, C.-M. Lim, H.-S. Kim, S.-C. Moon, S. Lalbahadoersing, and S.-C Oh. Alignment system and process optimization for improvement of double patterning overlay. In *Proc. of SPIE*, volume 6922, Feb 2008.

[86] J.-S. Yang and D. Z. Pan. Overlay Aware Interconnect and Timing Variation Modeling for Double Patterning Technology. In *Proc. Int. Conf. on Computer Aided Design*, November 2008.

[87] Ilan Englard, Rich Piech, Claudio Masia, Noam Hillel, Liraz Gershtein, Dana Sofer, Ram Peltinov, and Ofer Adan. Accurate in-resolution level overlay metrology for multi patterning lithography techniques. In *Proc. of SPIE*, volume 6922, Feb 2008.

[88] Dongsub Choi, Chulseung Lee, Changjin Bang, Daehee Cho, Myunggoon Gil, Pavel Izilson, Seunghoon Yoon, and Dohwa Lee. Optimization of high order control including overlay, alignment, and sampling. In *Proc. of SPIE*, volume 6922, Feb 2008.

[89] D. Laidler, P. Leray, K. D'have, and S. Cheng. Sources of Overlay Error in Double Patterning Integration Schemes. In *Proc. of SPIE*, volume 6922, Feb 2008.

[90] K. Jeong and A.B. Kahng. Timing analysis and optimization implications of bimodel cd distribution in double patterning lithography. In *Proc. Asia and South Pacific Design Automation Conf.*, January 2009.

[91] E. Y. Chin and A. R. Neureuther. Variability aware interconnect timing models for double patterning. In *Proc. of SPIE*, volume 7275, 2009.

[92] Christopher Cork, Kevin Lucas, John Hapli, Herve Raffard, and Levi Barnes. Large-scale double-patterning compliant layouts for dp engine and design rule development. In *Proc. of SPIE*, volume 7275, 2009.

[93] H. Mizuno, G. McIntyre, C. Koay, M. Burkhardt, B. Fontaine, and Obert Wood. Flare evaluation of asml alpha demo tool. In *Proc. SPIE*, volume 7271, 2009.

[94] P. Naulleau and S. George. Implications of image plane line-edge roughness requirements on extreme ultraviolet mask specifications. In *Proc. SPIE*, volume 7379, 2009.

[95] I. Yoneda, S. Mikami, T. Ota, T. Koshiba, M. Ito, T. Nakasugi, and T. Higashiki. Study of nanoimprint lithography for applications toward 22nm node cmos devices. In *Proc. of SPIE*, volume 6921, 2008.

[96] M. Li, L. Chen, and S. Chou. Direct three-dimensional patterning using nanoimprint lithography. In *Applied Physics Letters*, volume 78, 2001.

[97] Golden Kumar, Hong Tang, and Jan Schroers. Nanomoulding with amorphous metals. *Nature*, 457, 2009.

[98] C. Charpin-Nicolle, M. Irmscher, M. Pritschow, B. Vratzov, H. van Vossen, J. Chiaroni, J. Massin, and P. Gubbini. Integration issues in step and repeat UV nanoimprint lithography. In *Proc. of SPIE*, volume 6921, 2008.

[99] S. Landis, N. Chaix, C. Gourgon, C. Perret, and T. Leveder. Stamp design effect on 100 nm feature size for 8 inch NanoImprint lithography. *Nanotechnology*, 17:2701–2709, May 2006.

[100] *Nanoimprint lithography*, http://en.wikipedia.org/wiki/Nanoimprint-lithography.

[101] C. GREGORY and T. ALBERT. Programmable imprint lithography template. *US patent 7128559*, 2006.

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

Vita

Kun Yuan was born in Ji'an, JiangXi on 22 July 1983, the son of Guoqing Yuan and Zhaoying Lang. He received the B.S. degree in electronic engineering and information science from the University of Science and Technology of China, Hefei, China, in 2004. He is currently working toward the Ph.D. degree in electrical and computer engineering at the University of Texas, Austin. He was with TeraRoute, Austin, TX, during the summer of 2007 as a software engineering intern, and with NVIDIA, Santa Clara, CA, during the summer of 2009 as a hardware engineering intern.

His current research interests include physical design automation for manufacturability and numerical optimization. He has published about 4 journal papers and 9 conference papers. Mr. Yuan received the Microelectronic and Computer Development Fellowship for 2006-2008, the International Symposium on Physical Design Routing Contest Award in 2007, and the Best Paper Award at the Asian and South Pacific Design Automation Conference in 2010.

Permanent address: bigfish.yuan@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.