LAYOUT DECOMPOSITION FOR TRIPLE PATTERNING LITHOGRAPHY

BY

HAITONG TIAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Professor Martin D.F. Wong, Chair
Professor Deming Chen
Professor Rob A. Rutenbar
Professor Wen-mei Hwu

ABSTRACT

Nowadays the semiconductor industry is continuing to advance the limits of physics as the feature size of the chip keeps shrinking. Products of the 22 nm technology node are already available on the market, and there are many ongoing research studies for the 14/10 nm technology nodes and beyond. Due to the physical limitations, the traditional 193 nm immersion lithography is facing huge challenges in fabricating such tiny features. Several types of next-generation lithography techniques have been discussed for years, such as *extreme ultra-violet* (EUV) lithography, *E-beam direct write*, and *block copolymer directed self-assembly* (DSA). However, the source power for EUV is still an unresolved issue. The low throughput of E-beam makes it impractical for massive productions. DSA is still under calibration in research labs and is not ready for massive industrial deployment.

Traditionally features are fabricated under single litho exposure. As feature size becomes smaller and smaller, single exposure is no longer adequate in satisfying the quality requirements. *Double patterning lithography* (DPL) utilizes two litho exposures to manufacture features on the same layer. Features are assigned to two masks, with each mask going through a separate litho exposure. With one more mask, the effective pitch is doubled, thus greatly enhancing the printing resolution. Therefore, DPL has been widely recognized as a feasible lithography solution in the sub-22 nm technology node. However, as the technology continues to scale down to 14/10 nm and beyond, DPL begins to show its limitations as it introduces a high number of stitches, which increases the manufacturing cost and potentially leads to functional errors of the circuits. *Triple pattering lithography* (TPL) uses three masks to print the features on the same layer, which further enhances the printing resolution. It is a natural extension for DPL with three masks available, and it is one of the most promising solutions for the 14/10 nm technology node and beyond.

In this thesis, TPL decomposition for standard-cell-based designs is extensively studied. We proposed a polynomial time triple patterning decomposition algorithm which guarantees finding a TPL decomposition if one exists. For complex designs with stitch candidates, our algorithm is able to find a solution with the optimal number of stitches. For standard-cell-based designs, there are additional coloring constraints where the same type of cell should be fabricated following the same pattern. We proposed an algorithm that is guaranteed to find a solution when one exists. The framework of the algorithm is also extended to pattern-based TPL decompositions, where the cost of a decomposition can be minimized given a library of different patterns. The polynomial time TPL algorithm is further optimized in terms of runtime and memory while keeping the solution quality unaffected. We also studied the TPL aware detailed placement problem, where our approach is guaranteed to find a legal detailed placement satisfying TPL coloring constraints as well as minimizing the *half-perimeter wire length* (HPWL).

Finally, we studied the problem of performance variations due to mask misalignment in *multiple patterning decompositions* (MPL). For advanced technology nodes, process variations (mainly mask misalignment) have significant influences on the quality of fabricated circuits, and often lead to unexpected power/timing degenerations. Mask misalignment would complicate the way of simulating timing closure if engineers do not understand the underlying effects of mask misalignment, which only exists in multiple patterning decompositions. We mathematically proved the worst-case scenarios of coupling capacitance incurred by mask misalignment in MPL decompositions. A graph model is proposed which is guaranteed to compute the tight upper bound on the worst-case coupling capacitance of any MPL decompositions for a given layout.

To my parents, for their love and support.

ACKNOWLEDGMENTS

Firstly I would like to express my special thanks to my adviser Prof. Martin D.F. Wong. You have been a tremendous mentor for me. I would like to thank you for giving me a lot of insightful advice on my research and helping me to grow as a research scientist. This dissertation would not be possible without your advice and wisdom.

I am also very thankful to all my doctoral committee, Prof. Deming Chen, Prof. Wen-mei Hwu and Prof. Rob Rutenbar. Their constructive comments and suggestions have proven to be extremely useful for this thesis.

I am extremely grateful for all my labmates in UIUC, who have always been supportive in both my research and my life. I want to thank Dr. Hongbo Zhang for helping with my research topics and sharing much career advice. I want to thank Dr. Qiang Ma for discussing exciting research ideas, and teaching me to drive when we were in UIUC. I want to thank Dr. Yuelin Du for giving insightful comments for several of my research topics, and kindly accommodating me when my temporary housing expired while I was interning in the Bay Area. I want to thank Dr. Zigang Xiao for helping with my research topics. We have had the same adviser ever since we were masters in Hong Kong, and you have always been of tremendous help for both my research and my life endeavors. I also want to thank my labmates Prof. Fan Zhang, Dr. Pei-Ci Wu, Dr. Ting Yu, Ms. Leslie Hwang, Mr. Daifeng Guo, Mr. Tsung-Wei Huang, Mr. Chun-Xun Lin, Ms. Tin-Yin Lai and Mr. Iou-Jen Liu. You have made my PhD life more colorful and enjoyable.

I am extremely lucky for meeting lots of friends in UIUC. My roommate Dr. Mingcheng Chen has been extremely helpful during my ups and downs throughout my PhD life. I also want to thank my friends Mr. Jian Guan, Mr. Xiufu Wang, Mr. Yi Song, Dr. Jialu Liu, Mr. Xiang Ren, Mr. Zhuotao Liu, Mr. Zhenhuan Gao, Mr. Yi Liang, Mr. Zelei Sun, Ms. Shiya Liu, Dr. Qingxi Li, Mr. Zhenqi Huang, Dr. Dong Ye, Ms. Ying Chen, Ms. Mengjia Yan,

Ms. Wenting Hou, Ms. Xueman Mou, Mr. Shuai Tang and many others. I am proud to be a member of Chinese Students and Scholars Association (CSSA) when I was at UIUC. I am honored to work with all the folks in CSSA including Dr. Jiansong Zhang, Ms. Yuwei Chen, Ms. Yingqi Zhou, Ms. Yitang Guo, Mr. Lizi Zhang, Mr. Jing Jiang, Mr. Donghai Gai, Mr. Jin Xing, Ms. Jiahui Yu, Mr. Yuxiang Zhu, Mr. Wanlin Kong, Ms. Sujin Shi, Ms. Shiyan Zhang, Mr. Ti Xu, Ms. Ziqi Tang, Mr. Junfeng Guan, Ms. Xuran Peng, Ms. Zhenni Wang, Mr. Xuan Lv, Ms. Xuan Liu, Ms. Jinglin Zhong, Ms. Miaoyan Li, Ms. Yuan Liao, Mr. Cheng Wan and many others.

Finally, I give my deepest gratitude to my parents and my two sisters, Jing Tian and Cui Tian, who have always been supportive throughout my whole life. Words cannot express my love and gratitude for them.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AG Adjacency Graph

AU Atomic Unit

BCP Boundary Conflicted Polygon

BCG Boundary Conflicted Graph

BP Boundary Polygon

CD Critical Dimension

CG Constraint Graph

CPA Constrained Pattern Assignment

DSA Directed Self-Assembly

DOF Depth of Focus

DPL Double Patterning Lithography

EUV Extreme Ultra-Violet

HPWL Half-Perimeter Wire Length

IC Integrated Circuit

ICL Influenced Cutting Line

ILP Integer Linear Programming

LUT Look-Up Table

MPL Multiple Patterning Lithography

SADP Self-ALigned Double Patterning

SAT Boolean Satisfiability Problem

SDP Semi-Definite Programming

SG Solution Graph

SPC Solutions Per-Cell

STD Standard Deviation

TPL Triple Patterning Lithography

VSB Variable Shaped Beam

CHAPTER 1

A POLYNOMIAL TIME TRIPLE PATTERNING ALGORITHM FOR CELL-BASED ROW-STRUCTURE LAYOUT

1.1 Introduction

As technology advances, the feature size of chips continues to scale down. However, advancements in lithography technology have been slow and lagged behind. Due to the limitation of current 193 nm ArF immersion lithography, advancing the IC industry towards the 14/10 nm technology node has become a challenge. Although different types of next-generation lithography techniques have been discussed for years, such as *extreme ultra-violet* (EUV) [1, 2, 3, 4, 5, 6] lithography, E-beam direct write [7, 8, 9, 10, 11, 12] and nano-imprint techniques [7, 13], the most promising printing technique that will be used in the 14/10 nm technology node is still the 193 nm immersion lithography with multiple patterning techniques.

The key idea of multiple pattering lithography is to use several exposure processes for a single layer. Typically, the patterning techniques can be classified as: double patterning lithography (DPL, also known as litho-etch-litho-etch technique), triple patterning lithography (TPL, also known as litho-etch-litho-etch-litho-etch technique) and self-aligned double patterning (SADP). Due to the difficulties of bridging the mask rules and design rules in SADP [14, 15], DPL is now considered the key enabling technique for the 20 nm technology node. In DPL, patterns on one layer would be assigned to two different masks to double the printing pitch. In DPL, the color assignment is usually done by applying a minimum spacing rule, and any features that are closer than d_{min} (the minimum spacing) must be assigned different colors. Figure 1.1(b) shows an example of DPL decomposition. Because every two of them conflict with each other, a stitching is needed and feature a has to be further sliced into two parts, a_1 and a_2, to resolve coloring conflicts. Although it is always preferred to minimize the stitch number dur-

ing the DPL decomposition process, stitches in DPL are usually inevitable, especially in the circumstances of a high density layout. Those stitches potentially cause yield lost and increase manufacturing cost [16, 17, 18].

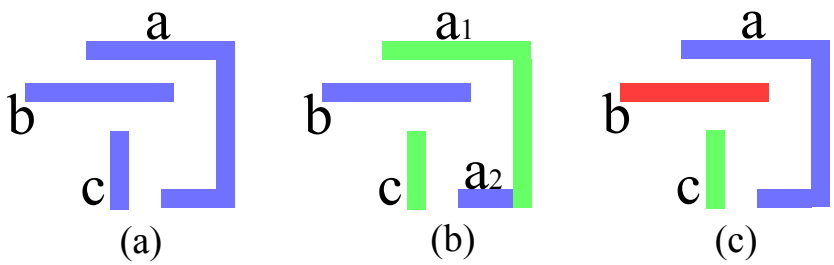


Figure 1.1: (a) A simple layout. (b) Patterning solution using double patterning lithography. (DPL) (c) Patterning solution using triple patterning lithography (TPL). Polygons with different colors mean that they appear in different masks.

DPL has been extensively studied in the literature. An ILP-based algorithm is proposed by Kahng et al. [17] to minimize the number of stitches. Xu and Chu [19] presented a graph reduction algorithm to minimize the number of stitches. A min-cut algorithm is proposed by Yang et al. [20] to minimize stitches, balance density and compensate overlay simultaneously. Xu and Chu [21] proved that the *conflict graph* used to model DPL is planar and introduced a matching based decomposer to simultaneously minimize the number of stitches and conflicts. Some DPL algorithms also consider layout modification to resolve the native conflicts [22, 23, 24]. Self-aligned double patterning is another choice for the future technology node. However, due to the decomposition difficulties and the big gaps between design rules and mask rules, it still requires further research before being massively adopted in the IC industry [14, 15].

Compared to DPL, TPL uses three masks for pattern assignment. Therefore, we can have more flexibility with color assignment and fewer conflicts among features. For the same layout in Fig. 1.1 (a), a stitch-free decomposition can be easily achieved using TPL as shown in Fig. 1.1 (c). Using different colors representing different masks, the TPL layout decomposition problem can be formulated as a 3-coloring problem, which is NP-complete. Yu et al. [25] made the first contribution in TPL decomposition with an ILP-based approach, and further proposed a semidefinite programming approximation algorithm to deal with dense layouts. However, the ILP-based algorithm is

expensive while the modified semidefinite programming is losing the optimality. A graph-based heuristic is proposed in [26], but it cannot guarantee to find a solution without resorting to an exponential algorithm. The TPL problem is also studied in [27]. However, it also failed to guarantee to find a solution if one exists. Moreover, more stitches are introduced compared with the results in [25].

In this chapter, we propose a polynomial time algorithm to find triple patterning decompositions of a standard-cell-based layout. Our contributions can be summarized as follows:

- We propose a polynomial time algorithm to solve the standard-cell-based row-structure TPL layout decomposition problem, and our algorithm has the capability to find all stitch-free decompositions.
- Color balancing is considered to achieve a balanced layout decomposition.
- We further improve our algorithm by first preprocessing each standard cell and then decomposing the whole layout on cell level. Experimental results show that this improved algorithm reduces the runtime by 34.5% on average without sacrificing the optimality.
- To deal with more complex designs, we further extend our approach to accommodate stitches. Our extension is very efficient, and guarantees to find an optimal solution using the minimum number of stitches.
- Our approach is highly scalable, and can be easily migrated to parallel implementations to further reduce the runtime.

The rest of the chapter is organized as follows: some preliminaries of the TPL problem are discussed in Section 1.2. Our basic algorithm will be presented in Section 1.3. The extended algorithm with stitches is discussed in Section 1.4. Section 1.5 shows the experimental results, followed by a conclusion in Section 1.6.

1.2 Preliminaries

Preliminaries of standard-cell-based layout decomposition are introduced here, including introductions to the standard-cell-based layout and the problem

definition.

1.2.1 Standard-Cell-Based Row-Structure Layout

In standard-cell-based designs, designers are given a library of pre-designed standard cells. All standard cells in the library have the same height, with power and ground tracks going from the far left to the far right. A layout consists of multiple rows, and in each row, the cells are aligned with power and ground connecting each other. The same type of cells may appear multiple times within a standard cell row.

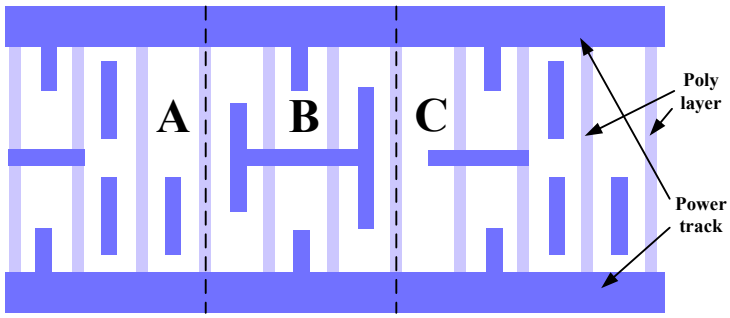


Figure 1.2: Layout of part of a standard cell row. Three cells, A, B, and C lie in the standard cell row. Only the poly layer and metal 1 layer are shown here for simplicity.

In the 14/10 nm technology node, TPL is only need for the densest layer with finest features – most likely including gate, low-level interconnect layers. For the gate and low-level interconnect layers except metal 1 (M1) which have preferred/required directions defined, solving triple patterning problem could be trivial by modified track-coloring assignment. The most difficult part of TPL decomposition comes from the M1 layer. For the M1 layer, the most commonly seen properties are as follows:

- Power and ground tracks connect all cells in the M1 layer in a row from left to right. A limited number of tracks is available between the power and ground tracks.
- Power and ground tracks are usually several times thicker than the finest features in the M1 layer, which can perfectly isolate the influences between different rows.

- Wires have no preferred directions.
- Most wires are defined locally inside the cells, with few connecting different cells in the same row.

A sample standard-cell-based circuit layout is shown in Fig. 1.2. There are three cells in the layout. Power track, which refers to the power and ground connections, is on the M1 layer. Higher metal layers are not shown here for simplicity.

1.2.2 Color Balancing

Among all legal decompositions, the ones where the features are evenly distributed on the three masks are more favorable. These well-balanced decompositions fully take advantage of each mask, and maximally benefit from the manufacturing process. This issue can be easily addressed in our framework as our algorithm has the capability to find all legal solutions of a layout. As we will explain in Section 1.3.6, our algorithm is able to find a balanced layout decomposition by scanning our solutions only once.

For a layout, there could be many legal decompositions. Although all solutions satisfy the minimum distance constraint, people in practice want to find a relatively balanced solution, in which none of the three masks dominates other ones. This concept can be efficiently incorporated into our framework as our algorithm is able to find all legal solutions of a layout.

1.2.3 Problem Definition

Given an M1 layer layout and minimum colorable distance d_{min}, our objective is to find a legal triple patterning decomposition for the M1 layer layout while balancing the area utilization in the three masks.

1.3 A Polynomial Time Algorithm

In the following ,we use different colors to denote different masks. Polygons with the same color will appear in the same mask. In this section, we will introduce our polynomial time triple patterning algorithm. Based on coloring

of standard cell rows, we also proposed a hierarchical approach to further accelerate our algorithm.

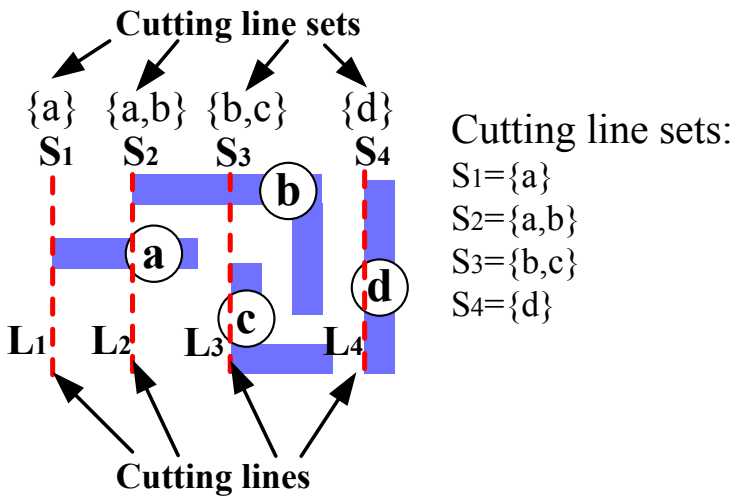


Figure 1.3: Cutting lines and cutting line sets.

1.3.1 Basic Terminologies

Some terminologies used in the algorithm are first introduced here.

Definition 1 (cutting line): A *cutting line* is defined as a vertical line going from the top of the standard cell row to the bottom of it.

Definition 2 (cutting line set): A *cutting line set* is defined as the set of polygons which intersect with the same cutting line.

Let us associate each polygon in one row with a cutting line with the same x coordinate of its left boundary, and eliminate the redundant ones. We then obtain a set L of n cutting lines with $L = \{L_1, L_2, ..., L_n\}$. Note that n is at most the number of polygons in this row. Let us assume the cutting lines in L are sorted in nondecreasing order with respect to their x coordinates, i.e. $x(L_i) \leq x(L_j)$ if $i \leq j$. Each cutting line L_i is associated with a cutting line set S_i, which consists of all polygons intersecting with cutting line L_i.

Consider the example shown in Fig. 1.3, there are four cutting lines L_1, L_2, L_3 and L_4, which are shown in red dashed lines. Their corresponding cutting line sets are $S_1 = \{a\}$, $S_2 = \{a, b\}$, $S_3 = \{b, c\}$, and $S_4 = \{d\}$ respectively.

To capture all coloring conflicts among the polygons and all legal solutions of a layout, two graphs, constraint graph and solution graph, are used in our

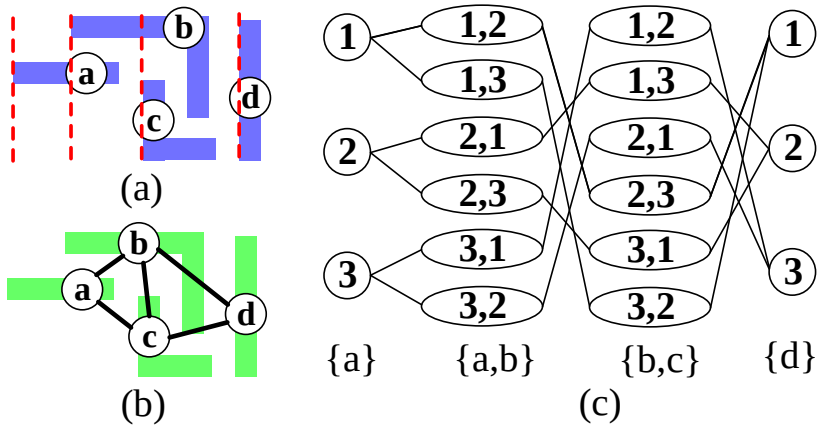


Figure 1.4: (a) Input layout. (b) Constraint graph. (c) Solution graph (different numbers here denote different colors).

algorithm.

Definition 3 (constraint graph): The constraint graph is defined as an undirected graph where the nodes represent polygons in a given layout, and where an edge means that the distance of the two polygons it connects are within the distance threshold d_{min} (the minimum spacing rule).

Figure 1.4(a) shows a simple layout with four polygons, and the corresponding constraint graph is shown in Fig. 1.4(b). If two nodes are connected in the constraint graph, they cannot be assigned the same color in a legal layout decomposition.

Definition 4 (solution graph): The solution graph is a directed graph where each node records a legal coloring solution of a cutting set, and where an edge exists between two nodes which belong to adjacent cutting lines if the coloring solutions of the two nodes are compatible to each other.

For each cutting line set S_i, all the coloring solutions can be generated by simply enumerating all possible coloring assignments. For each of the coloring solution of S_i, a node is created in the solution graph. Denote the set of nodes generated from the coloring solutions of S_i as N_i, and $N_i = \{N_i^1, N_i^2, ..., N_i^q\}$, where $q \leq 3^t$, and t is the maximum number of tracks in this row. For any node N_i^j and N_{i+1}^k, an edge is added to connect the two nodes if the two coloring solutions do not conflict with each other.

A simple example is shown in Fig. 1.4. There are four polygons in the layout. The constraint graph is shown in Fig. 1.4(b). There are four cutting

lines in the layout, which are shown in red dotted lines. For the first cutting line, which is the leftmost one, the cutting line set includes only polygon a. It has three coloring solutions: 1, 2, and 3, which are denoted by three nodes in the solution graph. For the second cutting line, the cutting line set includes polygons a and b. Similarly, its coloring solutions are denoted as six nodes. Edges are added if two nodes are compatible with each other. The same thing is done for the third and fourth cutting lines. Figure 1.4(c) shows the solution graph of the layout in Fig. 1.4(a).

1.3.2 Polygon Dummy Extension

In the constraint graph, a polygon may conflict with several other polygons. It is necessary to consider all conflicting polygons together to ensure a valid decomposition. However, those polygons are usually distributed in different cutting lines.

For the example shown in Fig. 1.5(a), there is only one polygon intersecting with each cutting line. The corresponding solution graph is shown in Fig. 1.5(c). A path from the leftmost to the rightmost of the solution graph corresponds to a patterning solution. For example, path (1,2,1,2) means that polygons a and c can be colored using color "1", while b and d can be colored using color "2". This solution is illegal since polygons a and c cannot be assigned the same color, which can be clearly seen from the constraint graph shown in Fig. 1.5(b). This is because conflicts between non-adjacent cutting lines are neglected, which leads to color assignment violations.

Based on the constraint graph, we propose a *polygon dummy extension* method to capture the conflicts of the polygons between multiple cutting lines. For each polygon in the layout, we locate its conflicting polygon that has the largest left x coordinate. Denote its left x as x_0. Then, the right boundary of the current polygon is virtually extended to $x_0 - \delta$, where δ is a very small value and is used to ensure that the new right boundary does not intersect with the cutting line $x = x_0$. After extending the right boundaries of the polygons, it is guaranteed that for any polygon in a cutting line set S_i, all its conflicting polygons (with smaller x coordinates) appear in the previous cutting line set S_{i-1}. Based on the modified layout, we go through each cutting line L_i and find the corresponding cutting line set S_i. The solutions

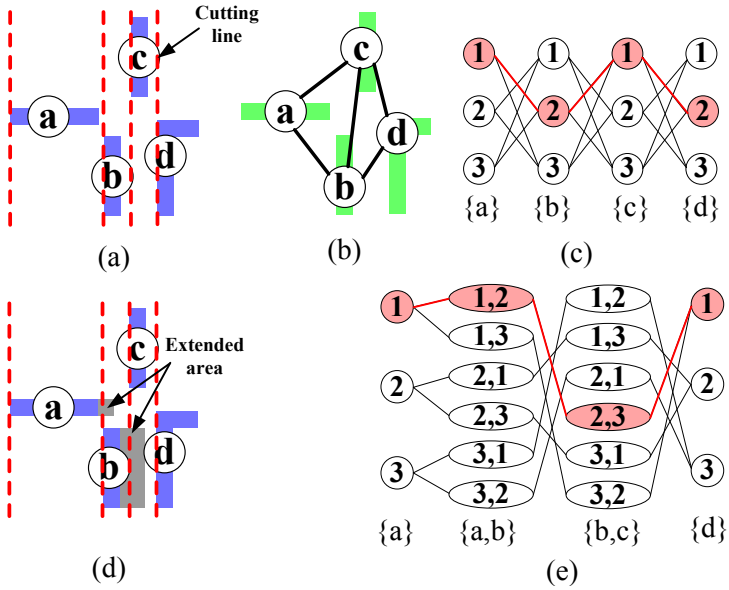


Figure 1.5: (a) Input layout. (b) Constraint graph. (c) Solution graph without polygon dummy extension. (d) Input layout after polygon dummy extension. (e) Solution graph with polygon dummy extension.

of S_i can be computed. For each solution of S_i, which is represented by a node in the solution graph, its compatible nodes in the solution graph are identified and an edge is added between the two nodes. Repeating the above steps, we can build a solution graph for a given layout.

For polygon a in Fig. 1.5, its conflicting polygons are polygons b and c. Denote the x coordinates of polygon b and c as x_b and x_c respectively. Since the left boundary of polygon c has a larger x coordinate, the right boundary of polygon a is virtually extended to $x_c - \delta$, where δ is chosen to be small enough such that polygon a intersects with the second cutting line, but not the third one. Figure 1.5(c) is the layout after polygon dummy extension, and Fig. 1.5(d) is the corresponding solution graph. We can see that based on the modified layout, every path in Fig. 1.5(d) is guaranteed to be a valid solution.

Theorem 1. *There is a valid triple patterning decomposition if and only if there is a path going from the leftmost of the solution graph to the rightmost of it.*

Proof. We prove the theorem by mathematical induction. The base case is that for the first cutting line L_1, all paths reaching nodes in N_1 in the solution

Algorithm 1: Coloring of a Standard Cell Row

```
begin
    Initialize solution graph G to be empty;
    P ← all polygons in a standard cell row;
    X ← x coordinates of the left boundaries of all polygons in P;
    Sort X in increasing order;
    w ← size of X;
    for i ← 1 to w do
        Set cutting line x = X_i;
        Find all polygons intersecting with x = X_i;
        Compute solutions for these polygons;
        Add the solutions into the solution graph G;
    end
    Find a path from the leftmost side to the rightmost side of G;
end
```

graph are legal, since these paths contain only one node, which must be legal. Now assume for cutting line L_i, all paths reaching nodes in N_i in the solution graph are legal. Consider the next cutting line L_{i+1}, the set of solution nodes are N_{i+1}. For the set of solution nodes in N_i and N_{i+1}, edges are only added when two nodes are compatible with each other. Using polygon dummy extension, it is guaranteed that for any polygon in cutting line set S_{i+1}, all its conflicting polygons appear in the previous cutting line set S_i. This means that the solutions nodes in N_{i+1} are only affected by the nodes in N_i. Since all paths reaching nodes in N_i are already legal, adding one more legal edge to those paths guarantees that the new paths are legal.

Similarly, the reverse case can also be proved by mathematical induction. Assume triple patterning solutions exist for a given layout. For any known solution, the coloring of all the polygons are known. The base case is that for the first cutting lines L_1, we can always find a node $N_1^{k^1}$ from the node sets N_1, in which all polygons in the cutting line sets S_1 are assigned the same color as they are in the legal TPL solution. Now consider the cutting lines L_i and L_{i+1}. Denote the compatible node we find in N_i and N_{i+1} as $N_i^{k^i}$ and $N_{i+1}^{k^{i+1}}$ respectively. Since the solutions of $N_i^{k^i}$ and $N_{i+1}^{k^{i+1}}$ are contained in the legal TPL solution, by definition, there must be an edge connecting the two nodes in the solution graph. All connecting nodes form a path in the solution graph. The proof is complete. □

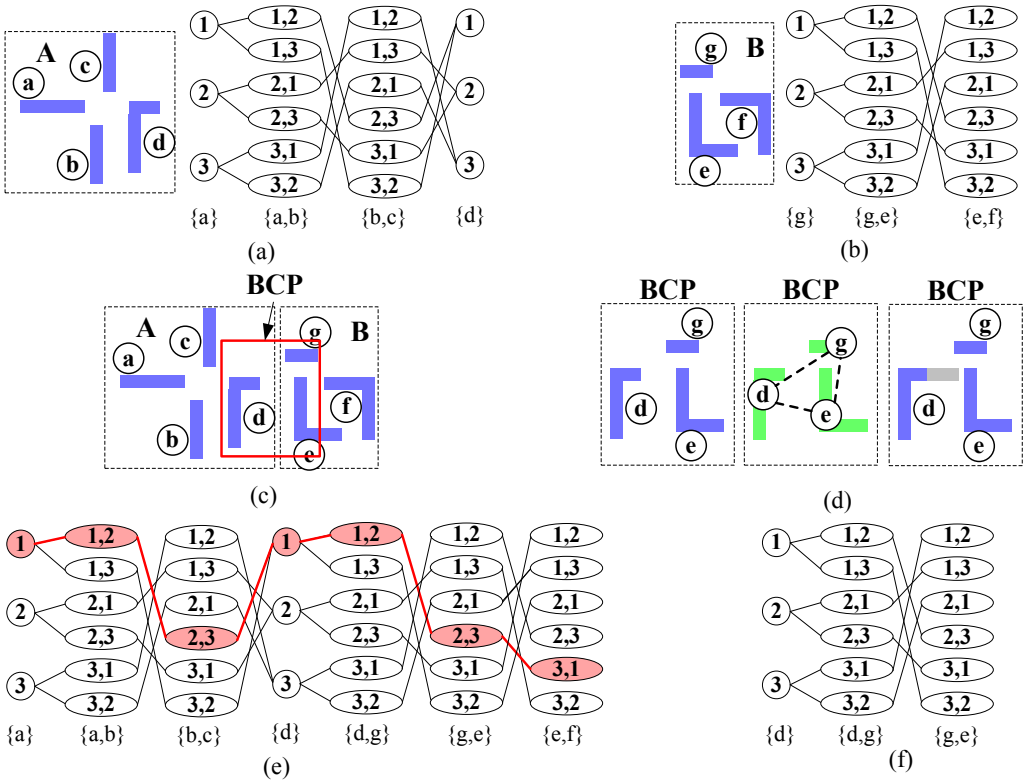


Figure 1.6: Illustration of BCP without connections. "SG" denotes "solution graph". (a) Three polygons, d, e, and g, appear in the BCP. (b) Solution graph of cells A, B, and the BCP. (c) Final solution graph and a sample solution path, which is shown in red color.

1.3.3 Power and Ground Connections

For standard-cell-based designs, each cell has its power and ground connections on the top and bottom that goes from the far left to the far right. Since the power and ground connections appear in all cutting lines, we can pre-color them before processing other polygons. They can either be assigned the same color, or different colors. There is no need to try all combinations, since we can generate other solutions from existing ones by rotating colors. For example, if we already get a solution graph G, we can easily get another solution graph G' by changing color 1 to color 2, color 2 to color 3 and color 3 to color 1. In the algorithm, both ways of pre-coloring by assigning the same and different colors to the power track are tried, and for each way of pre-coloring, the algorithm is able to find all possible coloring solutions. Therefore, the pre-coloring step does not affect the optimality of

our approach. Our row-based algorithm is shown in Algorithm 1.

Solution graphs of adjacent rows can be combined together based on the power and ground connections. If two rows share the same power connections, we require the coloring of the power connections in the two solution graphs to be the same. The same principle applies for ground connections.

1.3.4 Algorithm Complexities

Assume that there are n polygons in a standard cell row, and there are at most t horizontal tracks available for routing per standard cell row. Note that t can be regarded as constant under a particular manufacturing technology. Thus, each cutting line intersects at most t polygons. Due to the dummy extension of polygons, we need to enumerate the solutions of at most $2t$ polygons per cutting line. The number of solutions is thus upper bounded by 3^{2t}. Since the cutting lines are based on the left boundaries of the polygons, there are at most n cutting lines. For the solution nodes of two successive cutting lines, 3^{4t} operations are needed to connect the compatible nodes. The overall time complexity of our approach is $O((3^{4t}+3^{2t})n)$. Note that $3^{4t}+3^{2t}$ is constant here. Therefore, the overall complexity is $O(n)$. This shows that the standard-cell-based TPL problem is polynomial time solvable.

Note that this is a very pessimistic upper bound. In practice, there are seldom $2t$ polygons intersecting with a cutting line. Even there are $2t$ polygons, the number of solutions are far less than 3^{2t}, as many solutions can be pruned away based on the constraint graph. Moreover, different rows can be solved independently. For each row, our algorithm is guaranteed to find all possible solutions. Therefore, the parallel implementation does not affect the optimality of our algorithm, and the solution graph will be the same as that without parallel implementation.

1.3.5 Hierarchical Speedup Approach

For standard-cell-based circuit designs, millions of elements in a chip are typically composed of hundreds of basic cells in the standard cell library. If the solution graphs of all basic cells are precomputed, they can be reused in the higher hierarchy to color a given layout. In practice, the number

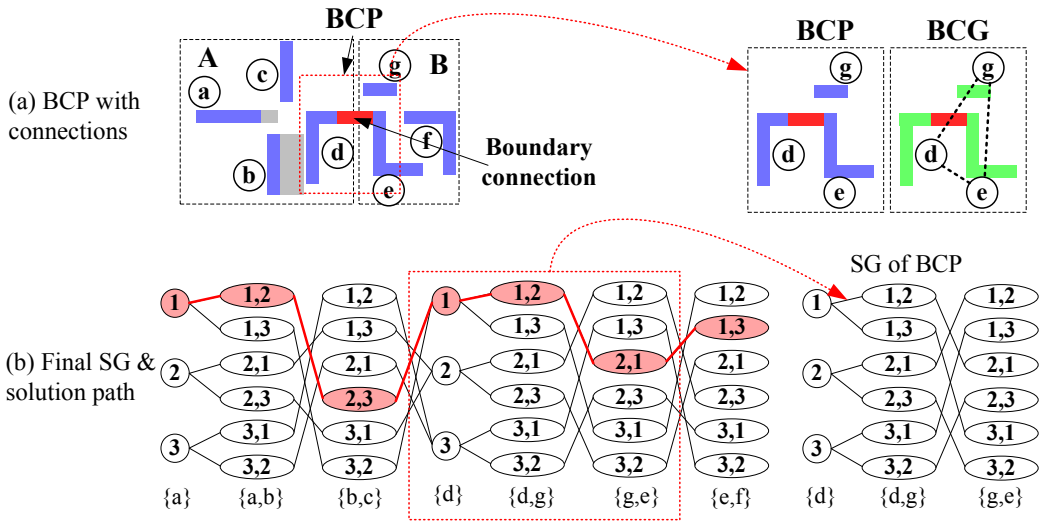


Figure 1.7: Illustration of BCP with connections. "SG" denotes "solution graph". (a) Three polygons, d, e, and g, appear in the BCP. (b) Final solution graph and a sample solution path, which is shown in red color.

of elements in a chip usually overwhelms the number of basic cells in the standard cell library. Thus, cell based hierarchical approach is expected to greatly accelerate the runtime compared with coloring a given layout as one large graph.

For each cell in the cell library, the constraint graph and solution graph are constructed. These two graphs can be constructed the same way as that for the standard cell rows. To connect different cells in a standard cell row, connections between cells are considered.

Boundary Polygons between Adjacent Cells

Adjacent cells in a standard cell row may introduce additional coloring conflicts. If polygons of adjacent cells are within the distance d_{min}, they have to be assigned different colors. To capture such constraints, we introduce additional conflicting edges recording all coloring conflicts between these polygons. Define *boundary conflicting polygons* (BCP) as the set of polygons within a distance of d_{min} from the boundaries of the adjacent cells. For the BCPs of two adjacent cells, all conflicting edges are identified. Based on the constraint graphs of the two cells and these conflicting edges, a *boundary constraint graph* (BCG) is constructed, which represents all conflicting rela-

tions between the BCPs. Polygon dummy extension is performed based on BCG. After that, we go over the left boundaries of polygons in BCP, and compute a solution graph for BCP. By combining the solution graphs of the two standard cells and the solution graph of BCPs, we can build a larger solution graph, which contains all possible legal patterning solutions for the adjacent cells.

A simple example with two cells is illustrated in Fig. 1.6. For cell A and cell B, their BCPs are first identified. Based on the BCPs and their constraint graph, polygon dummy extension is performed. Then, the solution graph for BCPs can be computed. By combining the solution graphs for cells A, B, and the BCPs, the final solution graph can be computed as shown in Fig. 1.6(c).

Connections between Cells

Additional coloring conflicts also include connections between different cells. If there are connections between two polygons for adjacent cells, they should be assigned the same color. Note that a BCP also contains all connecting polygons in adjacent cells. When enumerating solutions for BCPs, connected polygons are assigned the same color. For a connection that goes across several cells, we can group the cells it covers and treat them as one large cell. The row-based method can be used directly to compute the constraint graph and solution graph of the large cell. Then, the cell is treated as a regular cell in the algorithm.

An example of boundary connection between adjacent cells is shown in Fig. 1.7 to illustrate the above ideas. For the two connecting polygons d and e, they are assigned the same color in the solution graph. The flow of our hierarchical approach is shown in Algorithm 2.

1.3.6 Color Balancing

For a good TPL layout decomposition, the area utilization of the three masks should be balanced so that none of them dominate the other ones. These well-balanced decompositions fully take advantage of each mask, and maximally benefit from the manufacturing process. The area of polygons on each mask is used as the metric to evaluate the quality of a patterning solution. After

Algorithm 2: Hierarchical Speedup Approach

```
1  begin
2      C_lib ← all standard cells in the library;
3      C_row ← all standard cells in a row;
4      foreach Cell C_i in C_lib do
5          Build constraint graph G_i;
6          Build solution graph S_i;
7      end
8      m ← number of long connections in C_row ;
9      for j ← 1 to m do
10         C_new ← all the cells the jth connection covers;
11         Build its constraint graph G_new;
12         Build its solution graph S_new;
13         C_lib ←C_new;
14     end
15     w ← size of C_row ;
16     for j ← 1 to w do
17         Build partial solution graph G for the first jth cells in C_row;
18     end
19     Find a path from the leftmost side to the rightmost side of G;
20 end
```

obtaining the solution graph, a balanced patterning solution is chosen as follows.

Three variables are used here, each representing the total area of polygons with the same color. The solution graph is scanned from the leftmost cutting line to the rightmost cutting line. In each step, the color with the largest polygon areas is assigned the lowest priority, while the color with the smallest polygon areas has the highest priority. New polygons will be assigned the color that is legal and has the highest priority. Note that new polygons can only be assigned the color that is compatible with the color assignments of previous polygons.

1.4 TPL Incorporating Stitches

Since stitches potentially introduce many undesirable effects and will increase the manufacturing cost, it is always preferable to design a circuit layout which is 3-colorable. However, in practice, there may be complex cell layouts which are impossible to decompose into three masks without introducing stitches. For those layouts, we first find a set of legal stitch positions and decompose the original polygons into a set of smaller stitch polygons. Then, a modified

solution graph is constructed based on our previous optimal TPL algorithm. Lastly, a shortest path algorithm is invoked to get an optimal solution with the minimum number of stitches.

1.4.1 Stitch Position Identification

The same method is adopted as what is used in [25] to identify all stitch candidate positions. For a given layout, the layout graph simplification technique [25] is first performed to find the polygons that potentially require stitches. Then, node projection is invoked to find all projected segments on those polygons. Based on the projection results, all legal stitch positions are computed. Note that a stitch position is legal if it does not intersect with any projected segments. If the vertical stitch is illegal, the horizontal one will be tried. The original polygons are decomposed into a set of new polygons by the stitches. The constraint graph can be constructed based on those decomposed polygons. Besides the constraint edges, there also exists stitch edges in the constraint graph. There is a stitch edge connecting two nodes if a stitch candidate exists between the two corresponding polygons.

A simple example is shown in Fig. 1.8. Obviously, the layout shown in Fig. 1.8(a) is not 3-colorable since the constraint graph of the four nodes forms a complete graph, which is shown in Fig. 1.8(b). The node projection result is shown in Fig. 1.8(c). Two legal stitch positions are computed based on the node projection results, which are shown in Fig. 1.8(d). We can see that after adding the two stitches, legal triple patterning decompositions can be achieved.

1.4.2 Coloring a Standard Cell Row

After finding all stitch positions, coloring of the new layout is similar to the TPL algorithm without stitches. The solution graph here is different from that without stitches. Weights are assigned to edges in the solution graph. If two nodes corresponding to cutting line set S_i and S_{i+1} requires c stitches,[1] the weight of the edge will be assigned as c. Similarly, a hierarchical approach can also be adopted to further reduce the runtime.

[1] c is an integer, which reflects how many stitches are needed from one coloring solution to another.

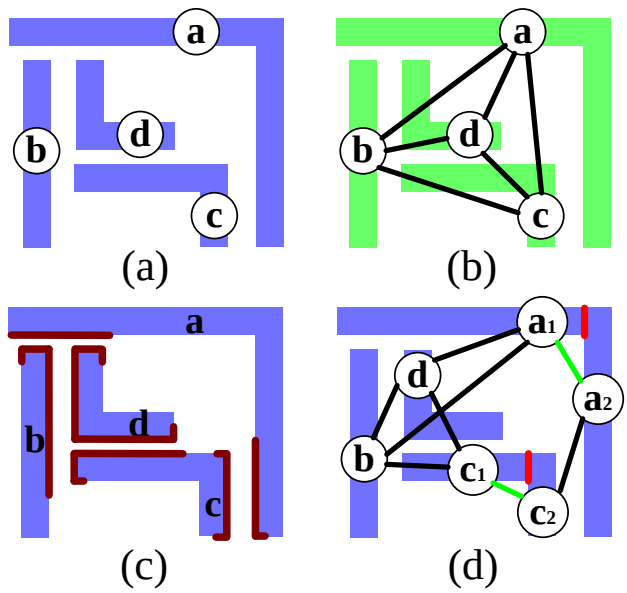


Figure 1.8: (a) Input layout. (b) Constraint graph of the input layout in (a). (c) The node projection. Projection edges are shown in bold brown lines. (d) Constraint graph after stitch decomposition. Polygon a is decomposed into polygons a_1 and a_2. Polygon c is decomposed into polygons c_1 and c_2. The stitch positions are shown using bold red lines. Stitch edges are shown in bold green lines.

1.4.3 Finding an Optimal Decomposition

Once we construct the solution graph, finding an optimal solution is quite straightforward. A shortest path algorithm can be employed to get an optimal decomposition with the minimum number of stitches. Note that the way we construct the solution graph is intrinsically beneficial for the shortest path formulation. If we go through the solution graph from left to right based on the cutting lines, all the nodes we visited are already in topological order. Unlike the ILP formulation which is very slow and the semidefinite programming formulation which loses its optimality in [25], the shortest path formulation is very fast and guarantees finding an optimal solution.

Similar to the TPL algorithm without stitches, the TPL algorithm with stitches also runs in polynomial time. The time complexity is $O(n+s)$, where n is the number of polygons and s is the number of stitch candidates in a give layout.

1.5 Experimental Results

The algorithm is implemented in C++ and run on a Linux server with 4GB RAM and a 2.8 GHZ CPU. NanGate FreePDK45 Generic Open Cell Library [28] is used to generate all benchmarks. We randomly select the standard cells in the cell library, and align them adjacently in different rows of a chip. The size of the standard cells are proportionally scaled down to reflect a 14 nm technology node. Connections between adjacent cells are randomly generated between their boundary constraint polygons. d_{min} is set to be 82 nm. Wires on the M1 layer are used for all experiments, as more wires including power tracks are on layer 1 and they also have more complex shapes compared with other layers.

1.5.1 Results of the Basic TPL Algorithm

Five benchmarks, $C1$ to $C5$, are generated with increasing number of polygons. The detailed results of our approach are shown in Table 1.1. For the largest benchmark with over 26 million polygons, the runtime is within an hour.

Table 1.1: Triple Patterning Decomposition Results

Test Cases	n	Balanced Area Ratio	Random Area Ratio	T (s)
C1	106690	1 : 1 : 1	1 : 0.27 : 0.23	10
C2	674841	1 : 1 : 1	1 : 0.26 : 0.24	66
C3	2695803	1 : 1 : 1	1 : 0.25 : 0.24	264
C4	10782073	1 : 1 : 1	1 : 0.25 : 0.24	1062
C5	26949406	1 : 1 : 1	1 : 0.26 : 0.24	2655

Note 1: "n" denotes the number of polygons in the benchmark.
Note 2: "T (s)" here is the results of our hierarchical algorithm.
Note 3: The area of the power track is subtracted.

The third column shows the results using our color balancing technique, while the results of a random color selection approach is shown in column four. For the random color selection approach, we go through the solution graph once and randomly assign the polygons a valid color. With the coloring balancing technique, we can achieve a much balanced decomposition compared with the result of a random color selection approach. Runtime is

Table 1.2: Runtime Comparisons

Test Cases	n	Tracks	T1 (s)	T2 (s)	Improve (%)
C1	106690	143	10	16	35.6
C2	674841	358	66	101	34.2
C3	2695803	715	264	403	34.4
C4	10782073	1429	1062	1610	34.0
C5	26949406	715	2655	4028	34.1
Ave.	8241763	672	812	1232	34.5

Note: Column of "T1" shows the runtime of our hierarchical algorithm, while column "T2" shows the results of our row-based algorithm.

shown in the last column in Table 1.1. We can see that the runtime is linearly correlated to the number of polygons in the benchmark, which further verifies that the algorithm is a polynomial time algorithm.

We also compare the runtime between our basic approach and our hierarchal approach, and the results are shown in Table 1.2. Our proposed hierarchal cell based algorithm can further improve the runtime by 34.5% on average without affecting the optimality of our algorithm. This clearly verifies the effectiveness of the hierarchical cell based algorithm.

1.5.2 TPL Algorithm with Stitches

Five benchmarks, C_6 to C_{10}, are also generated using more complex standard cells. The results shown in Table 1.3 are based on the hierarchical implementation. The number of polygons, the number of tracks, the number of stitch candidate, the number of final stitches, and the runtime are shown in column 2, 3, 4, 5, and 6 respectively. For the largest benchmark with over 17 million polygons, the runtime is within three hours. Note that with our shortest path formulation, we guarantee that the number of stitches computed is minimum.

1.5.3 Comparisons with Previous Works

We also compared our results with the previous works in [25] and [27] using the ISCAS-85 & 89 benchmarks provided by the authors of [25]. The same settings are used as those used in [27]. The detailed results are shown in Table 1.4.

Table 1.3: Triple Decomposition Results with Stitches

Test Cases	n	Tracks	Stitch Candidates	Stitches	T (s)
C6	179201	143	78102	3420	80
C7	904292	322	394349	17146	388
C8	4449681	715	1940587	83916	1900
C9	10031115	1072	4382524	188854	4277
C10	17813611	1429	7778321	334642	7613

Table 1.4: Comparisons with Previous Works

	SDP Based [25]			Algorithm in [27]			Ours		
Test Cases	C	S		C	S		C	S	
C432	3	1	✗	0	6	✓	–	–	✗
C499	0	0	✓	0	0	✓	0	0	✓
C880	1	6	✗	1	15	✗	0	7	✓
C1355	1	6	✗	1	7	✗	0	3	✓
C1908	0	1	✓	1	0	✗	0	1	✓
C2670	2	4	✗	2	14	✗	0	6	✓
C3540	5	6	✗	2	15	✗	–	–	✗
C5315	7	7	✗	3	11	✗	–	–	✗
C6288	82	131	✗	19	341	✗	–	–	✗
C7552	12	15	✗	3	46	✗	–	–	✗
S1488	1	1	✗	0	4	✓	0	2	✓
S38417	44	55	✗	20	122	✗	–	–	✗
S35932	93	18	✗	46	103	✗	–	–	✗
S38548	63	122	✗	36	280	✗	–	–	✗
S15850	73	91	✗	36	201	✗	–	–	✗

Note: "C" means the number of conflicts. If $C \neq 0$, no legal solutions are found (marked with ✗). "S" denotes the number of stitches.

Note that if the number of conflicts (shown in column named "C") is not zero, it means that the algorithm fails to find a legal decomposition. For all solved benchmarks in [25], we are able to find legal decompositions with optimal number of stitches. Our algorithm further solved four more benchmarks which the SDP-based algorithm cannot handle. For the algorithm in [27], they use a different stitch identification method, thus solving benchmark C432. However, the stitch identification method in [27] can be easily

incorporated into our framework to compute the optimal solutions. Moreover, we can solve four benchmarks where their approach fails. This clearly verifies the effectiveness of our approach.

1.6 Conclusions

In this chapter, we proposed a polynomial time algorithm to solve the standard-cell-based TPL problem. Our approach is highly scalable and can be implemented in parallel. Color balancing is considered to achieve a valid and balanced solution. Our approach has the capability to find all stitch-free decompositions for a standard-cell-based layout. To further reduce the runtime, we propose a hierarchical approach, which can reduce the runtime by 34.5% on average without sacrificing the optimality of the algorithm. To cope with more complex designs, we extended our approach to allow stitches. Our approach guarantees finding a solution with the minimum number of stitches. Our approach is expected to bring convenience to industry on the TPL problem and relieve the manufacturing bottlenecks on 14/10 nm technologies.

CHAPTER 2

CONSTRAINED PATTERN ASSIGNMENT FOR STANDARD-CELL-BASED TRIPLE PATTERNING LITHOGRAPHY

2.1 Introduction

As the technology continues to advance into 14/10 nm technology node, people are facing more and more challenging process requirements to print these small features. Double patterning technology (DPL) [17, 22, 29, 30, 31, 32, 33] is already reaching its limit at 20 nm technology node [34]. Beyond 20 nm technology node, next-generation lithography such as *extreme ultra-violet* (EUV) lithography and *E-beam*, or multiple patterning techniques have to be utilized to conquer these manufacturing difficulties. EUV [35, 36, 37, 38] has drawn plenty academic and industry attention as a viable candidate for the 14/10 nm technology node. However, the source power for EUV is still an unresolved issue, which delays its usage as a practical industry solution. DSA [39, 40, 41, 42, 43, 44, 45, 46] is still under calibration in research labs and is not ready to be deployed in industry as a feasible lithography technique. The low throughput of the E-beam [8, 9, 47] makes it unpractical for massive productions. TPL is a natural extension for double patterning lithography, which uses three masks to accommodate all the features in a layout. With one more mask than DPL, TPL provides more flexibilities for pattern assignment and is able to resolve most of the coloring conflicts. It serves as one of the most promising techniques for future lithography solutions.

For standard-cell-based designs, the designers are not only interested in achieving legal TPL decompositions, but also concerned with the quality of a TPL decomposition. There are many practical coloring constraints for TPL decompositions, among which the following two aspects are of great importance. Firstly, the same type of standard cells are preferred to be assigned the same color. This will best guarantee that the same type of cells

eventually have similar physical and electrical characteristics. Secondly, it is preferred to balance the usage of different colors during TPL decompositions. The solutions with better color balancing are more welcomed in small regions as well as in a full chip range. In this chapter, the color balancing scheme defined in a small region is called local color balancing; the color balancing scheme defined in a full chip range is called global color balancing. In practice, local color balancing is usually more important than globally balancing different masks, since the printability is more influenced by adjacent features. The well-balanced masks both locally and globally can be better utilized, and maximally benefits the manufacturing process.

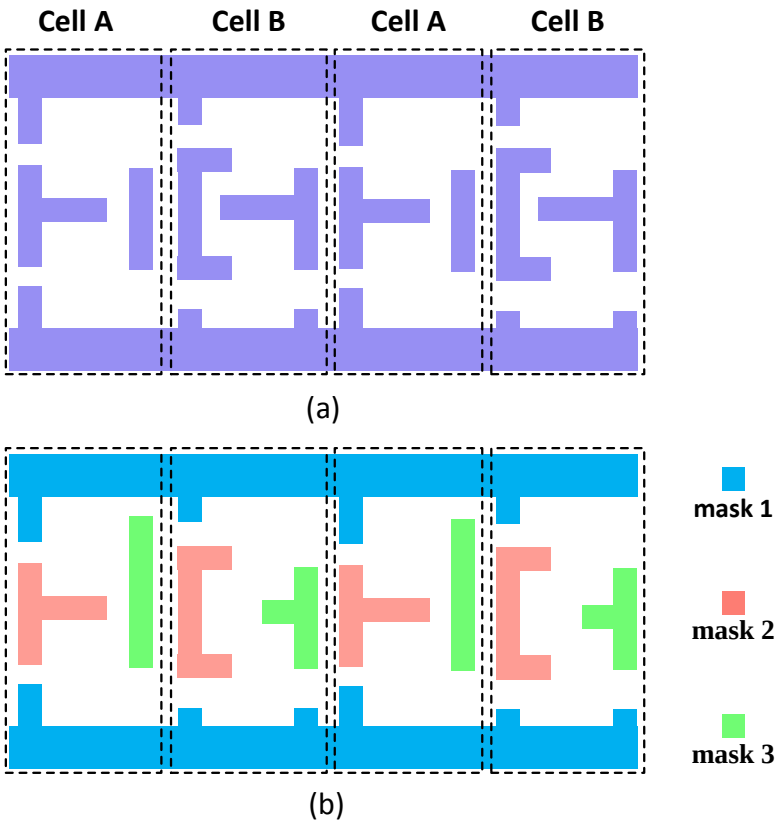


Figure 2.1: Illustration of the constrained pattern assignment problem. (a) Input layout. (b) TPL decomposition for the input layout. Note that the same type of standard cells are colored in the same way in this TPL decomposition. Different colors denote different masks.

For most of the triple patterning works, there is a minimum coloring distance d_{min}. Features within the distance d_{min} have to be assigned to different masks to resolve the coloring conflicts. With three masks, we can triple the

effective pitch distance and effectively improve the resolutions for printing. Many research efforts have been devoted to TPL [25, 26, 27, 48, 49, 50, 51, 52, 53, 54]. Bei Yu et al. showed that the general TPL decomposition problem is NP-hard, and further proposed an ILP-based algorithm to compute legal TPL solutions [25]. A semidefinite programming technique is also proposed to reduce the runtime. However, the ILP formulation is slow and the semidefinite formulation sacrifices the optimality. Moreover, the approach is not handling the above two coloring constraints. It has no control of assigning the same patterns for the same type of standard cells. Color balancing is also neglected in their formulation. Therefore, their algorithm cannot be directly used in the constrained pattern assignment problem. A graph-based heuristic is proposed in [27], which fails to capture the coloring requirements to assign the same pattern for the same type of cells. Moreover, color balancing is neglected in the approach, which could lead to very unbalanced TPL decompositions. Recently, Tian et al. [48] proposed a polynomial time triple patterning algorithm for standard-cell-based designs. A simple color balancing scheme is also proposed to achieve globally balanced decompositions. However, the proposed algorithm has no control of assigning the same patterns for the same type of cells. Moreover, the greedy method in [48] for global color balancing does not necessarily leads to a locally balanced decomposition.

We illustrate the idea of constrained pattern assignment problem using a simple example in Fig. 2.1. There are four cells in the layout, two cells of type A and two cells of type B. Based on the requirement from the constrained pattern assignment problem, a solution is shown in Fig. 2.1 (b). We can see that in the decomposition, the same type of cell is colored exactly the same way. Assigning the same pattern for the same type of cells gives the cell more predictable and consistent performance, and is more favorable in practice.

One straightforward approach to ensure identical pattern assignment for the same type of cells is to fix the colors of all the standard cells before placement and routing. However, it is not practical due to the adjacency and local interconnects of different standard cells in a layout. They possibly introduce additional coloring conflicts, thus rendering the fixed TPL decomposition approach ineffective. In this chapter, we proposed a novel hybrid approach to compute a constrained pattern decomposition for standard-cell-based designs. The main contributions of this chapter can be summarized as

follows.

- We proposed a novel hybrid approach to efficiently compute a constrained pattern decomposition for standard-cell-based designs. The approach guarantees finding a solution if one exists.
- When no solution exists for the constrained pattern assignment problem, we proposed another hybrid approach by solving a partial Max-SAT problem, which guarantees finding a legal decomposition if one exists, and tries to assign the same coloring solutions for as many cells as possible.
- To find a more balanced decomposition, a sliding window scheme is used to effectively compute locally balanced decompositions.

The rest of the chapter is organized as follows. Some preliminaries are introduced in Section 2.2. The constrained pattern assignment problem is formally defined in Section 2.3. The first step of our hybrid algorithm is discussed in Section 2.4, followed by second step which is a path-finding scheme based on sliding windows in Section 2.5. Experimental results are shown in Section 2.6. Finally, we conclude the chapter in Section 2.7

2.2 Preliminaries

In the following sections, we will briefly introduce the standard-cell-based row structure designs, the previous TPL algorithm, and the coloring constraints in the constrained pattern assignment problem.

2.2.1 Standard-Cell-Based Designs

In this chapter, we are focusing on the standard-cell-based row structure layout, which is also used in [48]. All the standard cells in the cell library have the same height, with power and ground rails going from the leftmost of the cell to the rightmost of it. Typical layout consists of multiple standard cell rows, with each row exactly the same height as the standard cell. The same type of cell may corresponds to many instances in a layout.

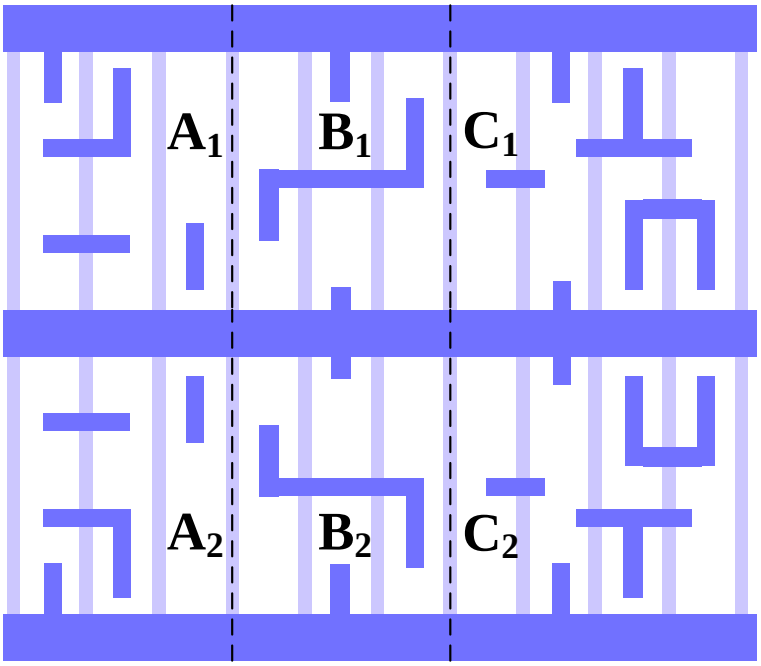


Figure 2.2: Example of standard-cell-based row structure layout. All the cells have exactly the same height. The same type of cell appears multiple times in the layout.

TPL is needed for the most dense layer in the 14/10 nm technology node, which is M1 in practice. For upper metal layers, preferred routing directions are given, where all the wires are either horizontal or vertical. In this chapter, we are focusing on TPL decompositions with coloring constraints for the M1 layer.

A simple example of standard-cell-based layout is shown in Fig. 2.2, where we have six instances. The six instances are composed from three types of cells in the cell library.

2.2.2 Previous TPL Algorithm

A TPL algorithm for standard-cell-based designs is proposed in [48]. Given a layout, its *constraint graph* (CG) is first computed. In the constraint graph, every polygon is represented as a vertex and an edge connects two vertexes if their distance is less than d_{min}. A *solution graph* (SG) is also defined in [48], in which every legal TPL solution corresponds to a path in SG and every path in SG corresponds to a legal TPL solution.

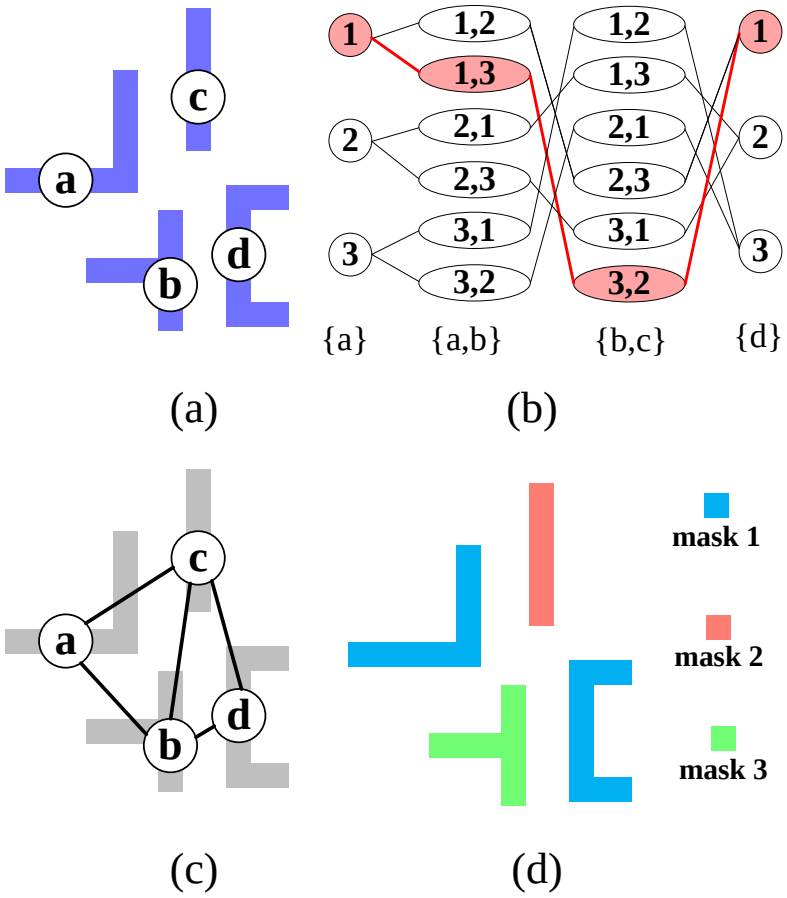


Figure 2.3: A simple example of the previous TPL algorithm in [48]. (a) The input layout with four features. (b) Solution graph of this layout. Different numbers here denote different masks. The path highlighted in red is a legal TPL solution. (c) Constraint graph of the input layout. (d) Final solution corresponding to the highlighted path, with different colors representing different masks.

For a standard cell row, a set of cutting lines are first derived based on the left boundaries of the features within the row. Every cutting line intersects with several features, whose TPL decompositions are enumerated and added into the solution graph. By sequentially processing all the cutting lines, a complete solution graph is constructed. A greedy color balancing approach is also proposed, which achieves good results for global color balancing. Interested readers please refer to [48] for a more detailed description of the algorithm.

Figure 2.3 gives a simple example to illustrate the previous TPL algorithm. Given a standard-cell-based layout, a solution graph is computed

which incorporates all legal coloring solutions. However, their approach has no control of assigning the same type of patterns for the same type of cells. The proposed greedy color balancing method is also too simple to achieve locally balanced decompositions.

2.3 Problem Definition

In this section, we will introduce the coloring constraints for TPL decompositions, and formally define the constrained pattern assignment problem.

2.3.1 Coloring Constraints

For standard-cell-based designs, millions of the elements on a chip are composed of hundreds or thousands of cells in the cell library. The same type of cells are preferred to be colored in the same way to achieve similar physical and electrical characteristics. However, no existing algorithms are able to handle this coloring constraint.

Properly balancing the usage of the three masks is another important coloring constraint for TPL decompositions. While there could be many legal TPL decompositions for a layout, the ones with balanced features both locally and globally are always more favorable. These well-balanced decompositions fully take advantage of each mask, and maximally benefits the manufacturing process.

For color balancing, locally balancing the features are more important than globally balancing the features within different masks. In practice, the printing quality of a feature are more affected by the features nearby rather than the features far away. Local color balancing best captures the local environment that affects printability, and therefore guarantees more favorable and meaningful decompositions.

2.3.2 Constrained Pattern Assignment

Constrained Pattern Assignment Problem: Given a standard-cell-based row structure layout, our objective is to find a legal TPL decomposi-

tion in which the same type of standard cells has exactly the same coloring solution, and features in different masks are locally balanced with each other.

2.4 A Hybrid Approach

Our algorithm can be divided into two steps. Firstly, fixing the cell boundaries and computing a solution graph for each standard cell. Secondly, utilizing the sliding window approach to compute a locally balanced decomposition. These two steps can be solved sequentially, and the algorithms are discussed as follows.

To compute the solution graph for each type of cell in the given layout, all the constraints within the layout have to be properly captured. The first step of our hybrid approach is to solve a small SAT problem to fix the cell boundaries, followed by computing a solution graph for each type of cell in the library. The details are discussed as follows.

2.4.1 Variable Notations

Given a feature, three binary variables are used to represent its mask assignment. For example, if we have a feature x_i, three variables, x_{i1}, x_{i2}, x_{i3}, are used to denote its coloring solutions. If x_i is assigned to mask 1, we have $x_{i1} = 1$, $x_{i2} = 0$ and $x_{i3} = 0$ respectively. The same principle applies when x_i is assigned to mask 2 or mask 3. Note that at any time, exactly one of the three variables is true.

2.4.2 Boundary Polygons

We first define some terminologies used in our algorithm. We reuse the definitions of *constraint graph* (CG) and *solution graph* (SG) in [48] for consistency. Besides that, we have one more technical term defined as follows.

Boundary Polygon: It is defined as a polygon within a standard cell that conflicts or connects with another polygon in any other standard cell in a given layout.

Figure 2.4 shows a simple example of boundary polygons. There are two adjacent cells, A and B, in the layout. As polygon x_1 connects to x_3, x_1

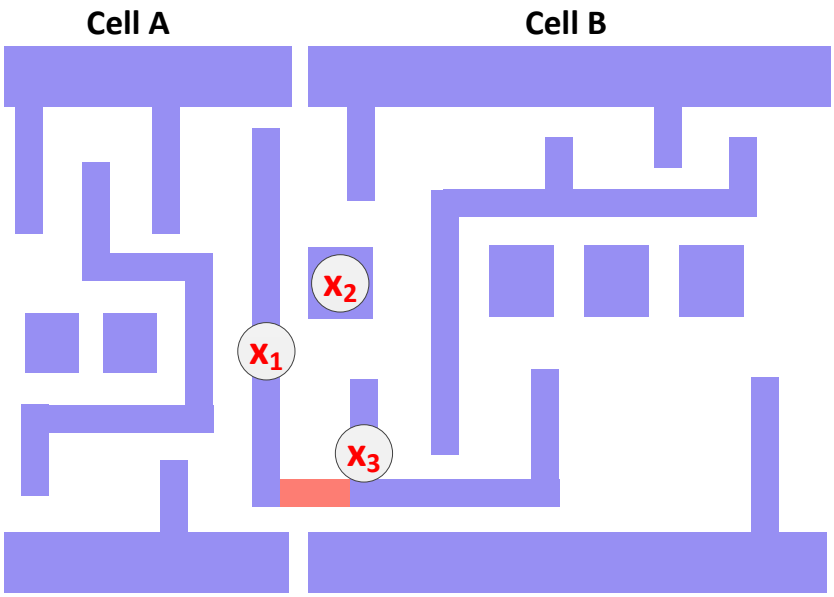


Figure 2.4: Boundary polygons in two adjacent cells, A and B, in a layout. For cell A, the boundary polygon is x_1. For cell B, the boundary polygons are x_2 and x_3 respectively. Note that the distance between polygon x_1 and x_2 is within d_{min}. The red polygon denotes the interconnect between the two polygons.

becomes a boundary polygon in cell A and x_3 becomes a boundary polygon in cell B respectively. Polygon x_2 is also a boundary polygon in cell B since x_2 conflicts with x_1 in the given layout. We can see that the boundary polygons for a standard cell are layout-dependent. For example, x_3 is a boundary polygon in cell B in the layout shown in Fig. 2.4. However, it is possible that x_3 does not correspond to a boundary polygon in cell B in another layout. A simple case is when there is no local interconnect between x_1 and x_3, x_3 will not be a boundary polygon in cell B.

For each adjacent cell boundary, we compute its constraint graph and get its local connection information. Based on the local connection information and the constraint graph, the boundary polygons for each standard cell can be computed. Therefore, after traversing the whole layout, all boundary polygons in the layout can be identified.

2.4.3 Capturing Boundary Constraints

After computing the boundary polygons for all standard cells in the cell library, we are ready to formulate the constraints among these polygons using SAT. Three types of boundary constraints are captured here.

- Boundary conflict: Due to the adjacency of different cells, polygons within different cells may conflict with each other. Let us use the polygons x_1 and x_2 shown in Fig 2.4 as an example. For each polygon, we have three binary variables representing the three masks. x_{11}, x_{12} and x_{13} denote the three masks for the polygon x_1. Similarly, x_{21}, x_{22} and x_{23} denote the three masks for the polygon x_2. If x_{11} is true, which means that x_1 is assigned to mask 1, x_{21} cannot be true since x_1 and x_2 conflict with each other. Similarly, if x_{12} is true, x_{22} should be false. If x_{13} is true, x_{23} needs to be false. The above constraints can be formulated as follows:

  $$(\neg x_{11} \vee \neg x_{21}) \wedge (\neg x_{12} \vee \neg x_{22}) \wedge (\neg x_{13} \vee \neg x_{23}) \tag{2.1}$$

  For any of two boundary polygons with distance less than d_{min}, we can formulate their constraints using SAT clauses similar to the above equation.

- Boundary connection: Boundary connections between two adjacent cells also impose constraints for constrained TPL decompositions. Again, the example in Fig. 2.4 can be used to illustrate the idea of how to formulate boundary connections based on SAT. For x_3, we have three variables x_{31}, x_{32} and x_{33} to denote its pattern assignment. As x_1 connects with x_3, they have to be assigned to the same mask. This means that if x_{11} is true, x_{31} has to be true. If x_{12} is true, x_{32} has to be true. Similarly, if x_{13} is true, x_{33} has to be true. The constraints are formulated into the following clauses using SAT:

  $$(\neg x_{11} \vee x_{31}) \wedge (\neg x_{12} \vee x_{32}) \wedge (\neg x_{13} \vee x_{33}) \tag{2.2}$$

  Similar with boundary conflicts, all boundary connections can be formulated into SAT clauses based on the above principle.

- Native constraint: This is a quite straightforward constraint. For each

polygon, we have three variables representing its coloring solutions. At any time, exactly one of the three variables has to be true. We will use Fig. 2.4 again to illustrate how to formulate the constraint. For x_1, if x_{11} is true, then both x_{12} and x_{13} have to be false. If x_{12} is true, both x_{11} and x_{13} have to be false. Similarly, if x_{13} is true, both x_{11} and x_{12} are set to be false. It is well known that if a statement is true, its contrapositive is also true (vice versa). It means that the two constraints, $x_{11} \rightarrow \neg x_{12}$ and $x_{12} \rightarrow \neg x_{11}$, are equivalent. Therefore, the above six constraints can be reduced into three clauses as follows:

$$(\neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{11} \vee \neg x_{13}) \wedge (\neg x_{12} \vee \neg x_{13}) \tag{2.3}$$

Similarly, the constraints for x_2 and x_3 can be written as:

$$(\neg x_{21} \vee \neg x_{22}) \wedge (\neg x_{21} \vee \neg x_{23}) \wedge (\neg x_{22} \vee \neg x_{23}) \tag{2.4}$$

$$(\neg x_{31} \vee \neg x_{32}) \wedge (\neg x_{31} \vee \neg x_{33}) \wedge (\neg x_{32} \vee \neg x_{33}) \tag{2.5}$$

The above constraints are not enough to ensure a valid solution. A trivial solution would be setting all variables to be 0. We need one more clause to ensure that for each polygon, at least one of its three binary variables is true. For the example in Fig. 2.4, we can formulate the constraints as:

$$\begin{aligned}(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \\ \wedge (x_{31} \vee x_{32} \vee x_{33})\end{aligned} \tag{2.6}$$

2.4.4 Capturing Cell Inner Constraints

All boundary constraints have been incorporated into our SAT formulation based on the discussion in Section 2.4.3. However, the above formulation does not guarantee that the solution computed will eventually lead to a valid solution. Now look at the example shown in Fig. 2.5. For cell E, there are two boundary polygons, x_1 and x_4 respectively. For cell F, there are two boundary polygons, x_2 and x_3 respectively. Based on the previous SAT

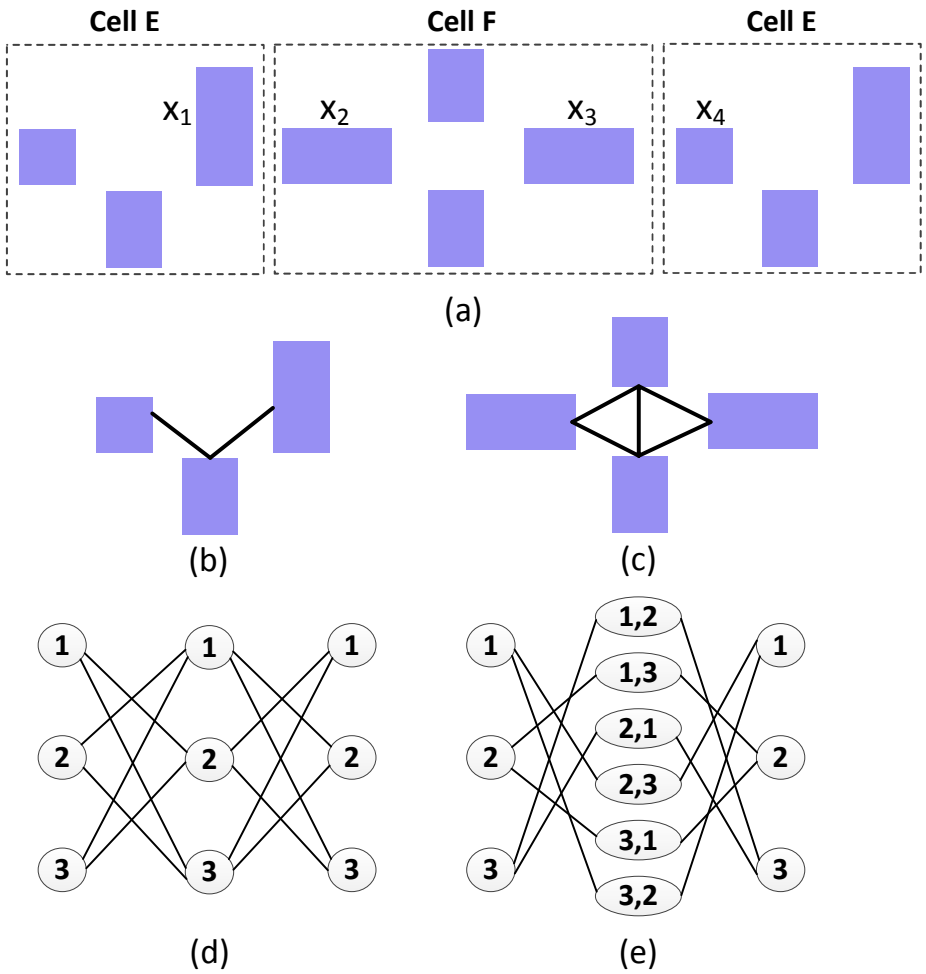


Figure 2.5: (a) Input layout. There are two cells, E and F, in the layout. (b) Constraint graph of cell E. (c) Constraint graph of cell F. Polygons conflicting with each other are connected with solid lines. (d) Solution graph of cell E. (e) Solution graph of cell F.

formulation, one possible solution would be x_1 is assigned to mask 1, x_2 is assigned to mask 2, x_3 is assigned to mask 3, and x_4 is assigned to mask 2 respectively. If x_2 is on mask 2 and x_3 is on mask 3, one can easily verify that there is no path connecting $x_2 = 2$ and $x_3 = 3$ in the solution graph of cell F shown in Fig. 2.5 (e). The problem for the above SAT formulation is that cell inner constraints are neglected. To capture this kind of constraints, we proposed the following technique.

For any cell C_i in the cell library, its constraint graph and solution graph are first computed. After we identify its boundary polygons, all possible coloring combinations for these polygons can be enumerated. Based on the

solution graph of cell C_i, one can easily verify whether a particular combination is feasible. For any combination that is illegal, one clause is added into the SAT formulation to forbid it. For example, for cell F shown in Fig. 2.5, there are two boundary polygons, x_2 and x_3 respectively. Based on the solution graph shown in Fig. 2.5 (e), one can easily verify that $x_2 = 1$ and $x_3 = 2$ does not correspond to any path in the graph, which means that this is not a valid combination. Therefore, we can add a clause as $(\neg x_{21} \vee \neg x_{32})$. The same procedure is applied for all the cells in the standard cell library. After adding the cell inner constraints, any solution computed by a SAT solver is guaranteed to be legal.

2.4.5 Computing the Solution Graph

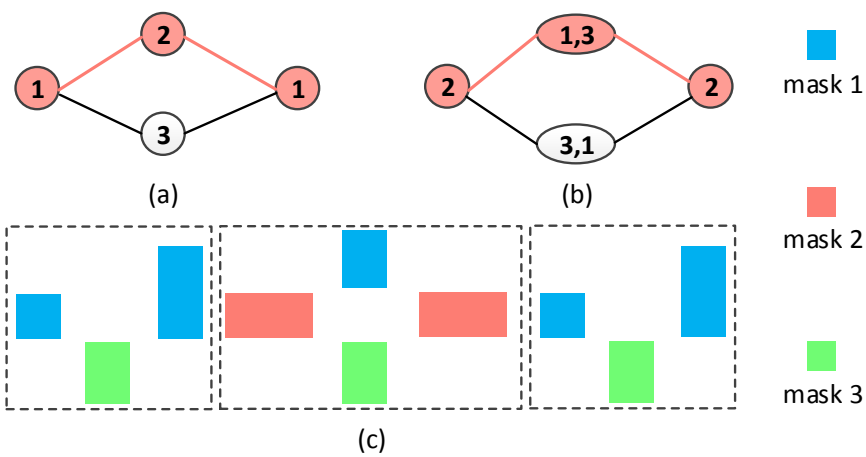


Figure 2.6: (a) Updated solution graph of cell E. (b) Updated solution graph of cell F. (c) Final coloring solution. The highlighted path is the coloring solution for cells E and F.

Based on the above formulations, solving the SAT problem will give us a solution for cell boundaries that guarantees to be legal. For any cell C_i in the cell library, the coloring assignments of all its boundary polygons are fixed after we solve the SAT formulation. After that, the algorithm in [48] is invoked to compute the updated solution graph of all cells in the cell library. For any cell C_i, its boundary polygons serve as the anchor polygons, whose coloring assignments have already been determined by the SAT solution.

One possible SAT solution for the example shown in Fig. 2.5 is $x_1 = 1$,

$x_2 = 2$, $x_3 = 2$, and $x_4 = 1$ respectively. The updated solution graphs for cell E and F are shown in Fig. 2.6 (a) and (b) respectively. After updating the solution graph for a cell in the library, we can traverse the graph and make any of the path to be its TPL decompositions. Note that any path in the solution corresponds to a legal TPL solution, which has been proven in [48]. A sample TPL solution for the layout in Fig. 2.5 is shown in Fig. 2.6 (c).

2.4.6 Power Tracks

For standard-cell-based designs, there are power tracks going from the left end of the cell to the right end of it. Power tracks of adjacent cells always connect with each other. Therefore, the power tracks always appear in a cell's boundary polygons.

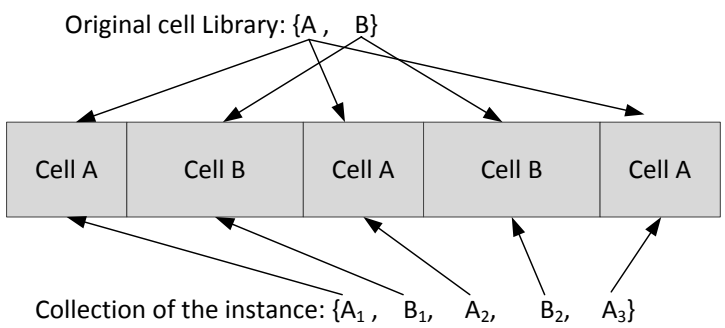


Figure 2.7: Example of creating a collection of instances.

Power tracks can be assigned to the same mask, or different masks. In practice, the power tracks are preferred to be on the same mask. In the experiments, we assume that the power tracks are on mask 1.

2.4.7 An Extended Partial Max SAT Approach

When no solution exists for the above SAT formulation, it means that not all the same type of cell can be colored the same way. By removing the constraint of enforcing the same color for the same type of cell, we can convert the constrained pattern assignment problem into a partial Max-SAT problem. In the partial Max-SAT problem, there are two type of clauses: hard clause and soft clause. The objective is to find a feasible assignment

that satisfies all the hard clauses together with the maximum number of soft ones.

A collection of all the instances is created based on the given layout. For each cell in the layout, we create an instance in the instance collections. For example, if cell A is repeated three times in the layout, three instances, A_1, A_2, and A_3 will be created in the instance collection. A_1, A_2, and A_3 are said to be of the same base type, since they are derived from the same type of cell in the cell library. After creating the instance collection, we eventually have the same number of instances in the collection as that in the given layout. We illustrate the idea using the example in Fig. 2.7. Originally there are two types of cells in the library. Based on the given layout, there will be five instances in the instance collection.

For each instance in the collection, its boundary polygons are identified and all the SAT clauses discussed above are added. These clauses are classified as hard clauses. Besides that, if two cells are of the same base type and the same boundary polygon appear in both cells, we add some soft clauses to make them on the same mask. For example, assume polygon x_1 is a boundary polygon in cell A_1 and polygon x_2 is a boundary polygon in cell A_2, and x_1 and x_2 correspond to the same polygon x in cell A, we prefer the polygons x_1 and x_2 to be on the same mask. If x_{11} is true, x_{21} is preferred to be true. If x_{12} is true, x_{22} is preferred to be true. Similarly, if x_{13} is true, x_{23} is preferred to be true. Therefore, we can add the following soft clauses into out partial Max-SAT formulation:

$$(\neg x_{11} \vee x_{21}) \wedge (\neg x_{12} \vee x_{22}) \wedge (\neg x_{13} \vee x_{23}) \tag{2.7}$$

Utilizing a partial Max-SAT solver, all the hard clauses and a maximum number of the soft clauses are satisfied. Based on the partial Max-SAT formulation, we can guarantee to compute a legal TPL decomposition if one exists, and tries to achieve the same coloring solution for the same type of cell for as many cells as possible.

2.4.8 Analysis of the Algorithm

The size of the SAT formulation is analyzed here to give some insights of the problem. Assume there are totally n boundary polygons. For each boundary

polygon, four clauses are added to represent the native constraints. There are $4n$ clauses for native constraints in total. For either boundary connection or boundary conflict, three clauses are added into the SAT formulation. If two boundary polygons does not conflict or connect with each other, no clauses are introduced. The worst case is that any two boundary polygons either conflict or connect with each other. In this case, there are at most $\frac{3n(n-1)}{2}$ clauses. Additional clauses are introduced by the cell inner constraints. Note that in each cell, the number of boundary polygons is very small. The number of clauses contributed by cell inner constraints is also limited.

In practice, the total number of boundary polygons are small, and conflicts between these boundary polygons are sparse. Local interconnects can also be enforced to be on other metal layers. The clauses contributed by boundary conflicts and boundary connections are far less than $3n^2$. The number of boundary polygons in a cell is also limited, which is usually far smaller than the number of features in the cell. Therefore, the size of the SAT problem is small and can be solved efficiently.

2.5 Approach for Local Color Balancing

In practice, designers are more interested in achieving a TPL decomposition where features on the three masks are both locally and globally balanced. Local color balancing is more important since the printability of a feature is mostly affected by the features nearby. Decomposition with locally balanced features everywhere usually means that the decomposition is roughly globally balanced. Local color balancing can be defined as follows.

Local Color Balancing: Given a user specified distance d and the bounding box of a feature, the bounding box is first extended towards all directions by d. Denote the area on the three masks within the bounding box B as a_1, a_2, and a_3 respectively. The objective of local color balancing is to $MIN(MAX(a_i - a_j))$ where $1 \leq i \leq 3$, $1 \leq j \leq 3$, and $i \neq j$.

None of the previous TPL works explicitly consider the issue of color balancing, except the work in [48]. They proposed a simple greedy heuristic targeting on globally balancing the area usage on the three masks. However, globally balancing the area usage on different mask does not necessarily leads to locally balanced decompositions. In the second step of our hybrid

approach, we propose a sliding window scheme which explicitly targets on locally balancing different masks. Only features within a certain distance range are considered when assigning masks for a feature. The sliding window scheme best captures the local environment that affects printability, and therefore generates more accurate and meaningful decompositions. This approach works as follows.

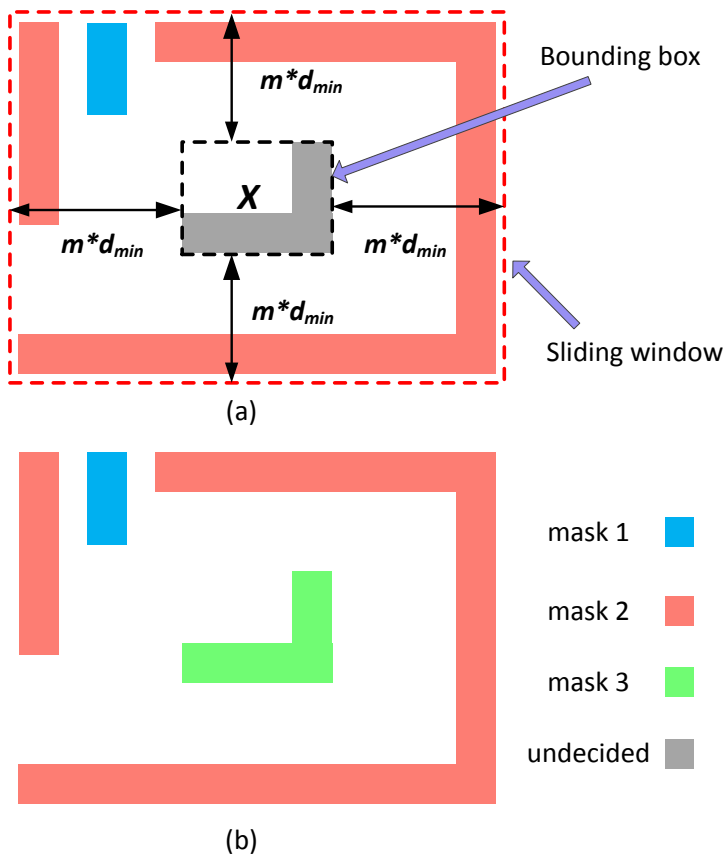


Figure 2.8: Illustration of the generation of a sliding window. Grey polygons mean that their colors have not been decided yet. (a) Bounding box of polygon X, shown in black dashed lines, and sliding window for X, shown in red dashed lines. (b) Coloring solution of polygon X.

The sliding window scheme is applied on all cells in the cell library. For any cell C_i, denote its jth polygon as P_{ij}. After computing the solution graph of cell C_i, we traverse the graph from its left boundary to its right boundary. For any polygon P_{ij} encountered, its bounding box is computed. After that, the bounding box is uniformly extended toward all directions by a distance of $m * d_{min}$, where m is a user specified parameter. The expanded bounding

box is defined as a sliding window for polygon P_{ij}, which is denoted as W_{ij}.

Three variables, a_1, a_2 and a_3, are associated with each sliding window. The variable a_1 represents the total area of the polygons that are assigned to mask 1 covered by the sliding window. Similarly, the variables a_2 and a_3 represent the total area of the polygons that are assigned to masks 2 and 3 and covered by the sliding window respectively. For a polygon partially covered by a sliding window, only the area covered is counted.

For each sliding window W_{ij}, we update the values of the above three variables. The mask with the smallest area is given the highest priority, whereas the mask with the largest area is assigned the lowest priority. The polygon P_{ij} is always assigned to a legal mask with the highest priority.

A sliding window example is shown in Fig. 2.8, where the color of the polygon X is to be decided. The bounding box of X is shown in Fig. 2.8 (a). After uniformly extending the bounding box, we get its sliding window shown in red dashed lines. Based on the values of a_1, a_2 and a_3 within the sliding window,[1] polygon X is assigned to mask 3, which is shown in Fig. 2.8 (b).

By enforcing a local sliding window, all nearby features that potentially affects the printability are captured. In each sliding window, the approach balances the utilizations of the three masks. As we traverse the solution graph from left to right, the sliding window is recomputed for every polygon in the layout, and the three variables are updated accordingly. The color that best balances the local area utilizations is assigned to the new polygon. In this way, the approach generates a locally balanced TPL decomposition.

2.6 Experimental Results

The algorithm is implemented in C++ and run on a Linux server with 8GB RAM and a 2.8 GHZ CPU. All benchmarks are generated using NanGate FreePDK45 Generic Open Cell Library [28], which is available online. The standard cells are randomly selected from the cell library, and are aligned adjacently in different rows of a chip. Local interconnects are assumed to be on higher mental layers. d_{min} is set to be 82 nm, and m is set to be 5. Wires on the M1 layer are used for all experiments. The Linux version of

[1]The variable a_3 equals to 0 for this particular example.

MiniSat-V1.14 is used in the experiments [55].

2.6.1 Constrained Pattern Assignments Results

We compare our hybrid approach with the previous work in [48], which also focuses on standard-cell-based designs. Five benchmarks are generated with increasing number of cells in the layout. The detailed results are shown in Table 2.1. The number of polygons and the number of boundary polygons in the benchmark are detailed in columns 2 and 3 respectively. Column 4 shows the average number of coloring solutions for a cell in the cell library generated by the algorithm in [48].

Cell A	Cell B	Cell A	Cell B	Cell A
Solution A_1	Solution B_1	Solution A_2	Solution B_2	Solution A_3

Figure 2.9: Calculating the average number of solutions per cell.

The average number of solutions per cell is computed as follows. Given a layout, we first run the algorithm in [48] to get its solution graph. After that, the color balancing heuristic in [48] is applied to get a TPL decomposition for the layout. For each type of cell in the layout, we count the number of distinct solutions based on the TPL decomposition computed. The average number of solutions per cell is obtained by adding the numbers together and dividing them by the number of different cells in the layout.

A simple example is shown in Fig. 2.9, where we have two types of cells in the layout. There are three distinct solutions for cell A, while there are two different solutions for cell B. The average number of solutions per cell is calculated as $\frac{3+2}{2} = 2.5$. This analysis indicates the necessity to use our method for the constrained pattern assignment problem. Note that the SAT algorithm guarantees that each type of cell has exactly the same decomposition, which means the average number of solutions per cell is 1. For the previous algorithm, there are usually multiple coloring solutions for a cell in the same layout. The larger the layout is, the more solutions there will be. This clearly shows the effectiveness of our approach.

The runtime of our hybrid approach stays almost unchanged for different

benchmarks, as there are limited number boundary polygons in the layout. Since the number of cells in the cell library is small, the number of boundary polygons is also limited. This enables us to utilize the SAT based algorithm in our hybrid approach. As shown in Table 2.1, the runtime of the SAT occupies a very small portion of the overall runtime.

Table 2.1: Comparisons with Previous Work in [48]

Test Cases	# P	# BP	Average SPC [48]	Runtime (s)	SAT Time (s)
test1	945	6	1.5	6.9	< 0.01
test2	3727	12	2.7	6.9	< 0.01
test3	7055	16	3.2	6.9	< 0.01
test4	14825	23	3.5	7.0	< 0.01
test5	31823	30	3.8	7.1	0.01

Note: SPC denotes number of solutions per cell.

2.6.2 Local Color Balancing

As the sliding window approach is seeing all the local information, more locally balanced decompositions can be achieved for a given layout. In the previous work [48], the authors propose a color balancing strategy to compute globally balanced decompositions. We compare our sliding window results with the results obtained by the previous algorithm in [48]. Our results are calculated by running the algorithm in [48] first, and then applying our sliding window scheme to compute a locally balanced solution.

Table 2.2: Local Color Balancing Results

Test Cases	# P	# BP	STD Ratio [48]/Ours	Runtime Ours (s)	Runtime [48] (s)
test1	945	6	1.21	8.1	6.5
test2	3727	12	1.19	12.4	6.8
test3	7055	16	1.21	18.0	7.3
test4	14825	23	1.21	31.7	8.3
test5	31823	30	1.21	62.1	10.1

Note: STD denotes standard deviation.

For any feature in the layout, its sliding window can be calculated. Based on the sliding window, the three variables, a_1, a_2, and a_3 are computed. These

three variables denote the area of the features on the three masks covered current sliding window respectively. The standard deviation of these three variables are also computed. For each feature in the layout, the standard deviation based on its sliding window is computed. For all the features, their standard deviations are accumulated. The ratio of the previous results over ours is showed in Table 2.2. We can see that the sliding window approach can achieve more balanced decompositions with less deviations compared with previous algorithm. The sliding window approach is slower compared with the previous approach. This is reasonable since more computational efforts are needed to obtain locally balanced decompositions. The runtime is acceptable in practice.

2.7 Conclusions

In this chapter, we propose a novel hybrid approach to solve the constrained pattern assignment problem for standard-cell-based TPL decompositions. Our algorithm efficiently solves this problem, and guarantees to find a solution if one exists. Our proposed sliding window approach also effectively computes locally balanced TPL decompositions, and gives superior locally balanced decompositions compared with the previous work in [48]. Experimental results show that the algorithm solves all the benchmarks in a very short runtime.

CHAPTER 3

TRIPLE PATTERNING AWARE DETAILED PLACEMENT WITH CONSTRAINED PATTERN ASSIGNMENT

3.1 Introduction

With the fast development of the semiconductor industry, products are already available using the 22 nm technology node, and the 14/10 nm technology node is also coming near. For such small features, traditional immersion lithography are facing great challenges, as the features are so small and close to each other that they cannot be well printed in one exposure. *Double pattering lithography* (DPL) is proposed to conquer the physical limitations, mostly diffractions, in the 22 nm technology node. However, they cannot be further extended to the 14/10 nm technology node. *extreme ultra-violet* (EUV) Lithography [35, 37] and *E-beam* [9] are also proposed to conquer the manufacturing difficulties and have drawn lots of research attentions recently. Problems still exist for these technologies, such as the demanding source power for EUV, and the low productivity for E-beam. These unresolved issues make them unpractical to be massively used in industry. *Triple pattering lithography* (TPL), which uses three masks to print the features in a layout, is a natural extension for DPL. Many of the research efforts have been devoted to TPL, and it is one of the most promising solutions for the 14/10 nm technology node.

Most of the existing works [25, 27, 48, 53, 56] focus on devising algorithms for TPL decompositions without modifying the layout, which are typically after placement and routing. There has been extensive research on the placement problem in the literature [57, 58, 59, 60, 61]. These works all focused on minimizing the HPWL/congestions of the final placement result without considering the manufacturing requirements, as double/triple patterning lithography are typically needed for the advanced technology node. In this chapter, we integrate the flow of detailed placement and TPL decompositions,

which simultaneously optimize the placement and decomposition processes.

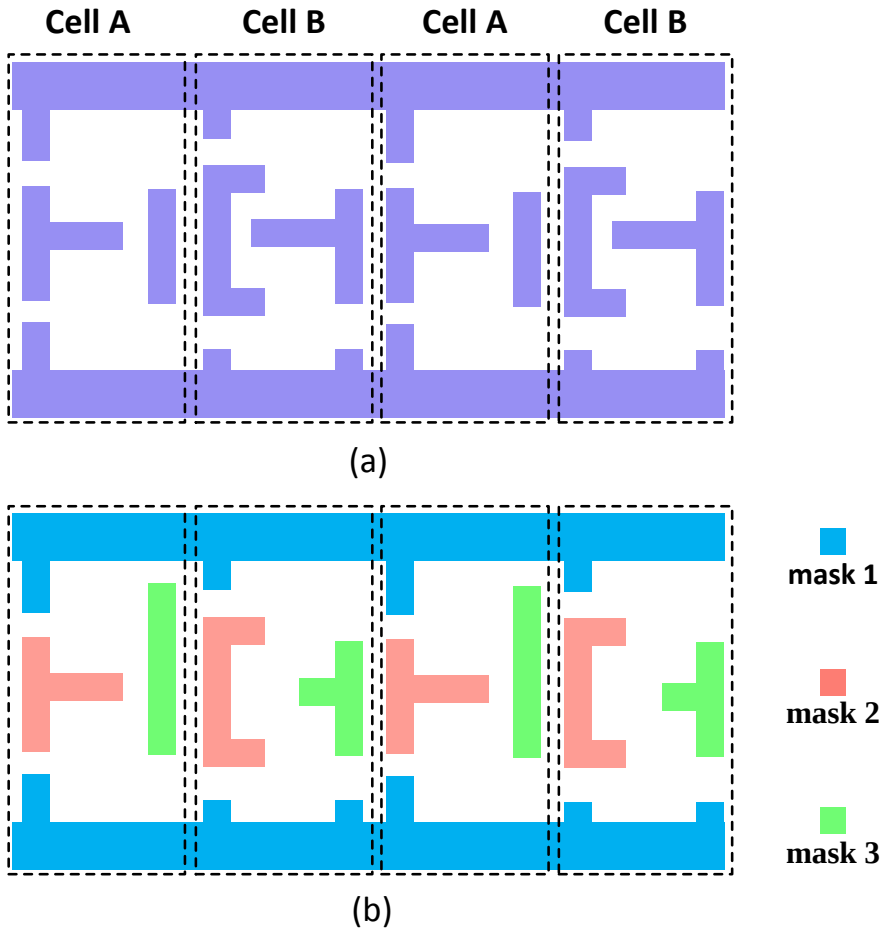


Figure 3.1: (a) Input layout. (b) TPL decomposition. The same type of cells are colored in the same way. Different colors denote different masks.

For the general TPL problem, legal decompositions have to be guaranteed. For standard-cell-based designs, there are more requirements besides achieving a legal TPL decomposition. One practical concern for designers is to assign the same patterns for the same type of standard cell. How to assign the same pattern for the same type of cells is called a constrained patterning assignment (CPA) problem. An example is shown in Fig. 3.1 where there are four cell instances composed from two types of cells. The TPL decomposition is shown in Fig. 3.1 (b), where the same type of cells are colored exactly in the same way. The additional coloring constraint is more robust for process variations, and gives the same type of cells similar physical and electrical characteristics, more predictable performance, and is

more favorable in practice.

As modifying the layout after the placement stage is extremely costly and inefficient, it is highly preferred to refine the layout during the detailed placement stage to make it CPA-friendly. In this chapter, we integrate the flow of detailed placement and TPL decomposition, and propose a hybrid approach to simultaneously optimize the placement and decomposition process. We formulate the problem into a weighted partial Max-SAT problem with a limited number of clauses, which guarantees finding a solution while minimizing the area overhead. An efficient graph model is also proposed to compute the exact locations of the cells with optimal HPWL. For each standard cell, our algorithm computes a CPA-friendly solution graph, which essentially explores all legal solution space for the cell. The contributions of this chapter can be summarized as follows:

- We propose an approach to effectively deal with the TPL aware detailed placement problem with CPA coloring constraints. Our algorithm is guaranteed to generate a legal detailed placement layout while minimizing the total area overhead.

- We propose an efficient graph model to compute the exact locations of the cells with optimal HPWL. The generated layout is guaranteed to be CPA-friendly.

- Instead of fixing the TPL decomposition after the integrated flow, a solution graph which explores all legal solution space is computed, giving the designers the freedom to choose desired TPL decompositions.

The rest of the chapter is organized as follows. Preliminaries of the CPA problem are introduced in Section 3.2. The CPA aware detailed placement problem is formally defined in Section 3.2.3. Our hybrid algorithm is discussed in Section 3.3 and 3.4. Experimental results are shown in Section 3.5 followed by conclusions in Section 3.6.

3.2 Preliminaries

A brief discussion about the characteristics of the cell-based row structure layout, the previous TPL algorithm in [48] and the CPA-friendly detailed placement problem are presented here.

3.2.1 Standard-Cell-Based Row Structure Layout

We assume that the layout is composed from limited types of standard cells from the cell library. All the cells are of exactly the same height, with power rails going from the leftmost of the cell to the rightmost of it. For different cell instances, they are placed in different standard cell rows. Within a standard cell row, the cells are aligned adjacently to each other, with all the cells sharing power and ground tracks. Features in different rows are isolated by the power tracks, which do not have coloring conflicts to each other. Similar assumptions have been made in previous papers [48, 62] as well.

3.2.2 Previous TPL Algorithm

The previous TPL algorithm in [48] is briefly introduced here, which is used to formulated some of the constraints in our problems. In their algorithm, a swiping line is utilized to scan the layout, where the solution graph is incrementally updated based on all the features that intersect with current swiping line. Some techniques are used to guarantee the legality of the solution graph. There is a nice property for the solution graph. Every path in the solution graph is guaranteed to be a legal TPL decomposition, and every legal TPL decomposition corresponds to a path in the solution graph.

3.2.3 CPA-Friendly Detailed Placement

For standard-cell-based designs, there are usually several hundreds types of cells while there could be millions of cell elements in a typical circuit nowadays. For the standard-cell-based designs, the designers are not only interested in computing legal TPL decompositions, but also concerned with the quality of a decomposition. One of the practical concern is to color the same type of standard cell in the same way to achieve similar physical and electrical characteristics, which is the nice property of a CPA-friendly layout.

To guarantee a feasible CPA solution for a layout, it is preferred to refine the layout during the placement stage to be CPA-friendly before doing TPL decompositions. A straightforward approach is to fix the colors of all cells in the library beforehand. Whenever two adjacent cells have coloring conflicts, they need to be placed further away from each other. As long as there is no

coloring conflicts between any two adjacent cells in the layout, there will be conflicts in the whole design space. It is obvious that a feasible CPA solution exists since there are no coloring conflicts between any two adjacent cells. However, as we show later in Section 3.5, the simple method suffers from high area overhead.

The CPA-friendly detailed placement problem is described as follows.

CPA-Friendly Detailed Placement: Given a legalized standard-cell-based detailed placement layout and a minimum conflicting distance d_{min}, our objective is to compute a CPA-friendly placement layout while minimizing the total area and HPWL overhead.

The CPA-friendly means that there are feasible TPL solutions where the same type of cells are colored exactly the same way in a layout. Refining the layout to be CPA-friendly can be effectively incorporated into the detailed placement stage after performing legalization, global and local swapping and flipping. By restricting all the cells to be shifted within the same row only, CPA-friendly layout can be achieved while minimizing the area overhead and pertaining the relative orders of all the elements in the layout.

3.3 CPA-Friendly Detailed Placement

In the following sections, we will introduce our weighted partial Max-SAT based algorithm, which guarantees to obtain a CPA-friendly placement layout while minimizing the area overhead. The size of the Max-SAT problem is also analyzed to give more insights into the problem.

3.3.1 Weighted Partial Max-SAT Variables

Denote $C = \{c_1, c_2, ..., c_n\}$ as the set of cells in the cell library. We reuse the definitions of *constraint graph* (CG) and *solution graph* (SG) in [48], and *boundary polygons* (BP) in [63] for consistency. In the following, we briefly review the three terminologies used here.

In the constraint graph, each node represents a polygon in the layout, with an edge connecting two vertices if the corresponding polygons are within the conflicting distance d_{min}. In the solution graph, the authors in [48] proved that every legal TPL solution corresponds to a path in the graph, and every

path is a legal TPL decomposition. The boundary polygon refers to the polygons within a standard cell that conflicts with another polygon in other standard cells in a given layout.

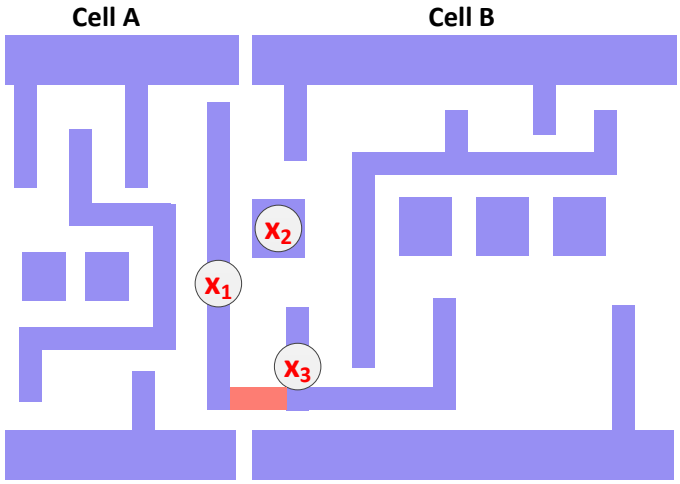


Figure 3.2: Layout with two cells c_i and c_j. There are three boundary polygons, x_1, x_2, and x_3, which are highlighted in blue color.

Given a legalized detailed placement result, all cells in different rows are sequentially parsed. Any polygon within one cell that conflict with other polygons in other cells is classified as a boundary polygon. The boundary polygons are represented as $X = \{x_1, x_2, ..., x_m\}$, where m is the total number of boundary polygons in a layout. An example is illustrated in Fig. 3.2, where we have two cells c_i and c_j in the layout. Because the distance of x_1 and x_2 is within d_{min}, x_1 and x_2 are all boundary polygons. Similarly, x_3 is also a boundary polygon as the distance of x_1 and x_3 is within d_{min}.

Given any boundary polygon x_i, we use three binary variables, x_{i1}, x_{i2}, x_{i3}, to represent its mask assignment. If x_i is assigned to mask 1, we have $x_{i1} = 1$, $x_{i2} = 0$ and $x_{i3} = 0$ respectively. When x_i is assigned to mask 2, we have $x_{i1} = 0$, $x_{i2} = 1$ and $x_{i3} = 0$ respectively. Similarly if x_i is on mask 3, we have $x_{i1} = 0$, $x_{i2} = 0$ and $x_{i3} = 1$ respectively. At any time, exactly one of the three variables is true.

3.3.2 Hard Clauses

The hard clauses denote those constraints that must be satisfied. After identifying all boundary polygons, the hard clauses are formulated as follows.

At any time, exactly one of the three variables for each polygon has to be true. Figure 3.2 is used to illustrate how to formulate the hard clauses. For x_1, if x_{11} is true which means that x_1 is assigned to mask 1, then both x_{12} and x_{13} have to be false. If x_{12} is true, both x_{11} and x_{13} have to be false. Similarly, if x_{13} is true, both x_{11} and x_{12} are set to be false. The same principle applies for x_2 and x_3. The hard clauses are formulated as follows:

$$(\neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{11} \vee \neg x_{13}) \wedge (\neg x_{12} \vee \neg x_{13}) \tag{3.1}$$

$$(\neg x_{21} \vee \neg x_{22}) \wedge (\neg x_{21} \vee \neg x_{23}) \wedge (\neg x_{22} \vee \neg x_{23}) \tag{3.2}$$

$$(\neg x_{31} \vee \neg x_{32}) \wedge (\neg x_{31} \vee \neg x_{33}) \wedge (\neg x_{32} \vee \neg x_{33}) \tag{3.3}$$

The following hard clause is also added to avoid the trivial solution that sets all variables to be 0. For the example in Fig. 3.2, the constraints are formulated as follows:

$$\begin{aligned}(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \\ \wedge (x_{31} \vee x_{32} \vee x_{33})\end{aligned} \tag{3.4}$$

Hard clauses reflecting the coloring constraints within a cell are also added. An example is shown in Fig. 3.3 (a) where there are two boundary polygons x_1 and x_2 respectively. Its solution graph is shown in Fig. 3.3 (b). One can easily verify that the coloring solution of $x_1 = 1, x_2 = 2$ is illegal. Therefore, a hard clause $\neg x_{11} \vee \neg x_{22}$ is added to prevent such an illegal assignment. For the example shown in Fig. 3.3, the hard clauses are formulated as follows:

$$\begin{aligned}(\neg x_{11} \vee \neg x_{22}) \wedge (\neg x_{11} \vee \neg x_{23}) \wedge (\neg x_{12} \vee \neg x_{21}) \\ \wedge (\neg x_{12} \vee \neg x_{23}) \wedge (\neg x_{13} \vee \neg x_{21}) \wedge (\neg x_{13} \vee \neg x_{22})\end{aligned} \tag{3.5}$$

Note that all the hard clauses are on the cell level without considering the inter-cell conflicts. If there is a solution satisfying all the hard clauses, it is guaranteed that the cell has a legal TPL decomposition.

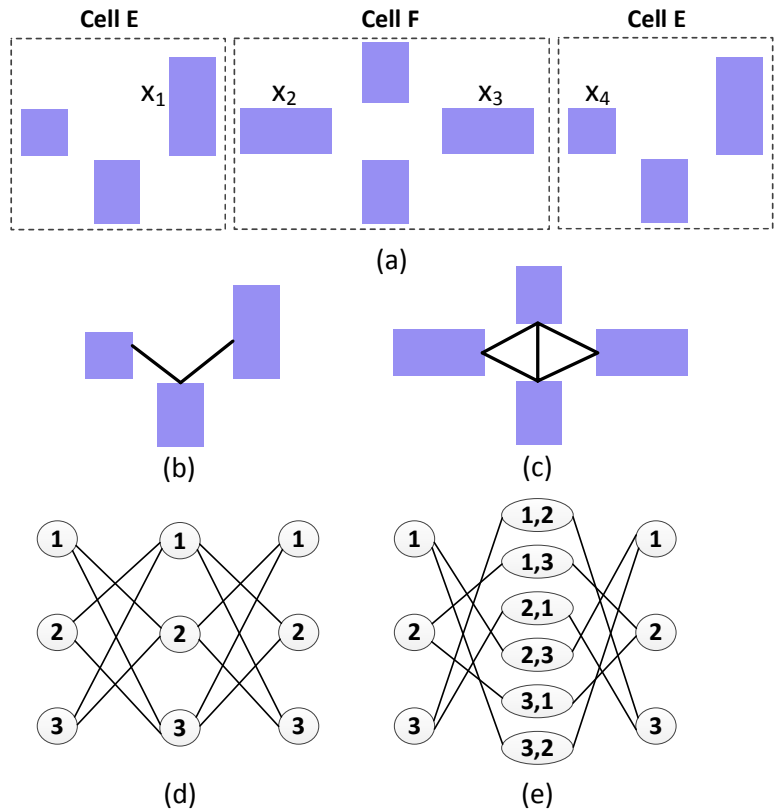


Figure 3.3: (a) Input layout with two boundary polygons x_1 and x_2. (b) Solution graph of the layout.

3.3.3 Soft Clauses

After setting up all the hard clauses, a solution computed is guaranteed to be legal at the cell level. At the layout level, the inter-cell constraints need to be properly captured to reflect the CPA coloring requirements.

Similar approaches can be used to formulate the soft clauses for two conflicting boundary polygons. Denote all cell instances in the layout as $S = \{S_1, S_2, ..., S_t\}$ where t is the total number of standard cell rows in the layout. Denote $S_i = \{s_1, s_2, ..., s_{u^i}\}$, where u^i is the total number of cell instances in row i and s_i is adjacent to s_{i+1}. For any two boundary polygons x_i and x_j, d_{ij}^{ab} denotes the minimum number of placement sites needed to make them conflict free to each other between cell instances s_a and s_{a+1} in row b. If x_i or x_j do not exist in the boundary between s_a and s_{a+1}, d_{ij}^{ab} is zero. Define w_{ij} as follows, which is the total number of placement sites needed to make

all x_i and x_j conflict free to each other.

$$w_{ij} = \sum_{r=1}^{t} \sum_{\lambda=1}^{u^r-1} d_{ij}^{\lambda r} \tag{3.6}$$

Use Fig. 3.2 as an example. For the two polygons x_1 and x_2, if x_{11} is true, x_{21} must be false. If x_{12} is true, x_{22} must be false. Similarly, x_{13} is true, x_{23} must be false. Weights are assigned to the clauses to reflect the area penalties when they are violated. For two boundary polygons x_1 and x_2, w_{12} placement sites are needed to make them conflict free, the soft clauses will have weight w_{12}. Similarly, the weight for clauses between x_1 and x_3 is w_{13}. The soft clauses in Fig. 3.2 can be expressed as follows:

$$w_{12}[(\neg x_{11} \vee \neg x_{21}) \wedge (\neg x_{12} \vee \neg x_{22}) \wedge (\neg x_{13} \vee \neg x_{23})] \tag{3.7}$$

$$w_{13}[(\neg x_{11} \vee \neg x_{31}) \wedge (\neg x_{12} \vee \neg x_{32}) \wedge (\neg x_{13} \vee \neg x_{33})] \tag{3.8}$$

Definition 1 (Atomic Unit)**.** *For any two conflicting polygons x_i and x_j, the Atomic Unit (AU) is defined as the following clauses:*

$$((\neg x_{i1} \vee \neg x_{j1}) \wedge (\neg x_{i2} \vee \neg x_{j2}) \wedge (\neg x_{i3} \vee \neg x_{j3}))$$

We can see that the AU refers to the soft clauses of two conflicting boundary polygons. AU is empty for non-conflicting boundary polygons. When all the hard clauses are satisfied, we have the following lemma.

Lemma 2. *For any AU, at least two of its three clauses are satisfied.*

Proof. When the hard clauses are satisfied, exactly one of the three variables, x_{i1}, x_{i2} and x_{i3}, is true. The same principle applies for x_{j1}, x_{j2} and x_{j3}. Without loss of generality, assume $x_{is} = 1$ and $x_{jt} = 1$ where $s = \{1, 2, 3\}$ and $t = \{1, 2, 3\}$. If $s \neq t$, one can easily verify that all three clauses in the AU are satisfied. If $s = t$, all clauses except $(\neg x_{is} \vee \neg x_{jt})$ are satisfied. This concludes our proof. □

Denote the atomic unit of two boundary polygons x_i and x_j as AU_{ij}. The soft clause for x_i and x_j can be denoted as $w_{ij}AU_{ij}$. Denote all the hard clauses as C_{hard}, and rewrite the soft clauses as $C_{soft} = \sum_i \sum_j w_{ij}AU_{ij}$, where $i = \{1, 2, 3, ..., m\}$ and $j = \{1, 2, 3, ..., m\}$, and m is the total number of boundary polygons. Our objective is to minimize F while satisfying all

clauses in C_{hard} where

$$F = 3\sum_i \sum_j w_{ij} - \sum_i \sum_j T_{ij} \tag{3.9}$$

$$T_{ij} = \begin{cases} 3w_{ij} & \text{If } x_i \text{ and } x_j \text{ are on different masks} \\ 2w_{ij} & \text{Otherwise} \end{cases} \tag{3.10}$$

Note that minimizing F is exactly the same with solving the weighted partial Max-SAT formulation. The formulation are composed of two parts: hard clauses which must be satisfied and soft clauses where the clauses with a maximum amount of total weight are satisfied. The problem remains of how can we properly map the value of the objective function to the area overhead for achieving a CPA-friendly layout.

3.3.4 Capturing Critical Polygons

The area overhead is not accurately captured by the above model. Figure 3.2 is used to illustrate the inaccuracies of the formulation. Assume the constraints between x_1 and x_2, x_1 and x_3 cannot be resolved, and one placement site is needed to resolve all the conflicts. Based on Lemma 2, the total weight of the violated constraints will be $w_{12} + w_{13}$. To remove the conflicts, we need to move all adjacent cells of c_i and c_j one placement site away, which in practice incurs an area overhead of w_{12} number of placement site. Thus, the weight of the clauses no long reflects the area overhead to resolve these conflicts.

Before introducing the approach to accurately capture the area overhead, we first define some terminologies used here.

Definition 2 (X-Freedom)**.** *The X-Freedom of two boundary polygons x_i and x_j is defined as the horizontal distance h_{ij} needed to move x_i and x_j further apart from each other, such that the distance of x_i and x_j equals to d_{min}.*

A simple example is shown in Fig. 3.4 to illustrate the concept of X-Freedom. Here d_{min} is assumed to be 5. For x_1 and x_2, the X-Freedom is 2, as it is the minimum horizontal distance to move x_1 and x_2 apart such that their distance equals to d_{min}. Similarly, the X-Freedom of x_1 and x_3 is about 1.58.

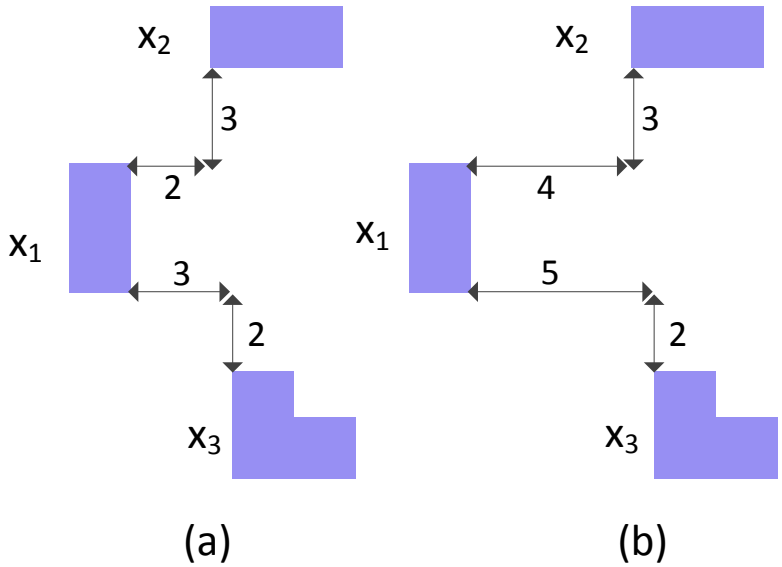


Figure 3.4: (a) Input layout with three boundary polygons. Distance of x_1 to x_2, and x_1 to x_3 are the same. (b) Critical polygons are x_1 and x_2. $h_{12} = 2$, $h_{13} = 1.58$.

Definition 3 (Critical Polygons)**.** *The critical polygons between two types of cells are defined as the pair of boundary polygons with the largest X-Freedom.*

The key observation is that if the critical polygons for cell c_i and c_j are conflict free, all boundary polygons within c_i and c_j are also conflict free. One can easily verify this based on the definitions of X-Freedom and critical polygons.

For any two adjacent cells c_i and c_j in the layout, the pair of critical polygons are computed. The constraints of the critical polygons are treated as soft clauses, while the remaining constraints of all other boundary polygons are added into the hard clauses. For the example shown in Fig. 3.2, the clauses in equation 3.8 will be added into the hard clauses. Finally, only clauses of the critical polygons are included in the soft clauses. Note that adding the constraints of other boundary polygons into the hard clauses imposes more restrictions on the problem. Therefore, the formulation tries to minimize the area overhead but may not necessarily lead to an optimal value.

Lemma 3. *The value of F equals to the area overhead to achieve a CPA-friendly placement result.*

Proof. From Lemma 2, for any violated AU_{ij}, it contributes w_{ij} to F. Otherwise, it contribute 0 to F. Without loss of generality, assume AU_{ij} is violated, where the two corresponding boundary polygons are x_i and x_j respectively. According to the definition of w_{ij}, the total area overhead to remove all conflicts between x_i and x_j is exactly w_{ij}. Therefore, for any violated AU_{ij}, the area overhead is exactly the same with the weight it contributes to the objective function. This concludes our proof. □

3.3.5 Excluding Native Conflicts

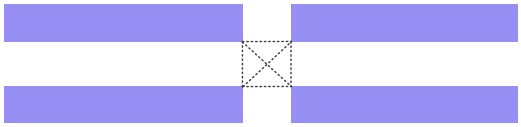


Figure 3.5: Layout with native conflict.

For non-critical boundary polygons, we need to be careful with the native conflicts when adding them into the hard clauses. Native conflicts means that there is no legal TPL decompositions for the features. As shown in Fig. 3.5 where there are native conflicts among the four polygons. If they are added into the hard clauses, the weighted partial Max-SAT solver will return no results even a solution exists. Therefore, a preprocessing step is incorporated to detect the native conflicts among non-critical boundary polygons. If there are native conflicts, they are not added into the hard clauses. Instead, we locate all such boundaries and insert necessary placement sites to remove these conflicts. Denote the area overhead incurred by the native conflicts as F_{native}, the final area overhead equals to $F + F_{native}$.

3.3.6 CPA-Friendly Solution Graph

After solving the Max-SAT formulation, the coloring solutions of all boundary polygons are known. The algorithm in [48] is applied to compute a solution graph for each type of cell in the cell library. The algorithm in [48] essentially explores all the solution space that satisfies the Max-SAT constraints which are solved in the SAT formulation. The solution graph incorporates all

Algorithm 3: CPA-Aware Layout Generation

```
1  begin
2      C ← All cells in the library;
3      Compute solution graph for all cells in C;
4      BP ← Boundary polygons in the layout;
5      CP ← Critical polygons in the layout;
6      Exclude naive conflicts;
7      C_hard ← All hard clauses for BP;
8      C_soft ← All soft clauses for CP;
9      Solve C_hard + C_soft;
10     Extract colors for BP;
11     Update solution graph for all cells in C;
12 end
```

legal solutions for the cell. Instead of fixing the TPL decomposition, we leave the designers the freedom to choose whichever decomposition that suits their particular needs. The overall flow of the algorithm is shown in Algorithm 3.

3.3.7 Analysis of the Algorithm

The size of the Max-SAT formulation is analyzed here to give some insights of the problem. Assume there are totally n boundary polygons. For each boundary polygon, four clauses are added to represent the native constraints. There are $4n$ clauses for native constraints in total. For boundary conflict, three clauses are added into the Max-SAT formulation. If two boundary polygons does not conflict with each other, no clauses are introduced. The worst case is that any two boundary polygons conflict with each other. In this case, there are at most $\frac{3n(n-1)}{2}$ clauses. Additional clauses are introduced by the cell inner constraints. Note that in each cell, the number of boundary polygons is very small. The number of clauses contributed by cell inner constraints is also limited.

In practice, the total number of boundary polygons are small, and conflicts between these boundary polygons are sparse. The clauses contributed by boundary conflicts and boundary connections are far less than $3n^2$. The number of boundary polygons in a cell is also limited, which is usually far smaller than the number of features in the cell. Therefore, the size of the Max-SAT problem is small and can be solved efficiently.

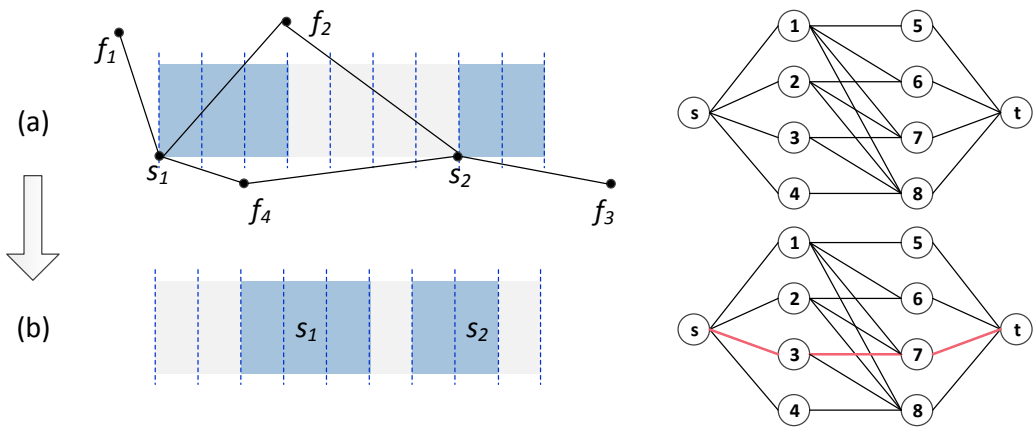


Figure 3.6: (a) Input layout with two cell instances s_1 and s_2, nine placement sites, and four nets $\{s_1 f_1\}$, $\{s_1 f_2 f_4\}$, $\{s_2 f_2 f_4\}$, and $\{s_2 f_3\}$. The graph model is shown on the right. Weight of the graph is not shown here for simplicity. (b) Final solution of the placement with optimal HPWL. The shortest path in the graph is highlighted in red.

3.4 CPA-Friendly Refinement with Optimal HPWL

For both global and detailed placement, HPWL has always been a key metric to evaluate the quality of a placement result. Minimizing HPWL is one of the primary objectives for many state-of-the-art placement algorithms [57, 58, 59, 61, 64, 65]. In this section, we focus on computing the locations of all the cells in a single row with optimal HPWL while satisfying the CPA coloring constraints. Similar problems have been addressed in some previous works [60, 62, 66, 67, 68]. Here we propose a graph model that correctly captures our CPA coloring constraints, and solve the single row cell ordering problem with optimal HPWL. The following discussions are for cell instances in a single row, where each cell instance is uniquely identified by its lower left coordinates. The same algorithm is applied for all rows in the layout repeatedly until the total HPWL improvement is less than a user-specified threshold.

For this problem, we have the following input:

- Standard cell library $C = \{c_1, c_2, ..., c_n\}$ with their length $L = \{l_1, l_2, ..., l_n\}$ where n is the number of cells in the library. The width of the cell is scaled as a multiple value of the width of a placement site.

- All cell instances in a row $S = \{s_1, s_2, ..., s_u\}$, where u is the number of

cell instances in the row.

- Net information of all cell instances in S.
- The solution of the previous partial weighted Max-SAT problem.

In the following, we will discuss the details of the proposed algorithm.

3.4.1 Capturing X-Scope of a Cell

Definition 4 (X-Scope)**.** *The X-Scope of a cell instance s_j is defined as an interval $[L_j, R_j]$, where L_j and R_j represent the leftmost location and rightmost location respectively that we can place s_j while satisfying CPA coloring constraints.*

A lookup table LUT is constructed with dimension $n \times n$ where n is the number of cells in the library. For any two cells c_i and c_j, there are four types of cell adjacency: left boundary of c_i to left boundary of c_j, left boundary of c_i to right boundary of c_j, right boundary of c_i to left boundary of c_j, and right boundary of c_i to right boundary of c_j. Denote them as ll, lr, rl, rr respectively. Each entry $LUT(i, j)$ includes four more entries $LUT(i, j, 0)$, $LUT(i, j, 1)$, $LUT(i, j, 2)$ and $LUT(i, j, 3)$, which stores the CPA-friendly distance needed for ll, lr, rl and rr boundary adjacency of cell c_i and c_j respectively.

Given any two types of cells c_i and c_j, assume ll boundary adjacency appears in the soft clauses and is satisfied in the SAT solution. Denote the width a placement site is w_p. Denote the minimum distance among all ll boundary adjacency for c_i and c_j in the layout as d^{ij0}_{min}. $LUT(i, j, 0)$ is set to be $\lceil \frac{d^{ij0}_{min}}{w_p} \rceil$. The same principle applies for lr, rl and rr boundary adjacency. For all adjacency that are violated or not included in the soft clauses, the values in LUT are set to be $\lceil \frac{d_{min}}{w_p} \rceil$, which means that all the cell instances need to be conflict free to each other.

Intuitively, if we pack all the cell instances before s_j as compact as possible, we get L_j. Similarly, if we pack all the cell instances after s_j as compact as possible, we get R_j. Denote L as the length of the cell row and t_j as the type of cell instance s_j in the cell library where $t_j \in \{1, 2, ..., n\}$. Denote the type of boundary adjacency of two instances s_j and s_{j+1} as $b_{j,j+1}$ where

$b_{j,j+1} \in \{0, 1, 2, 3\}$. The X-Scope can be computed as follows:

$$L_j = \sum_{m=1}^{j-1}(l_{t_m} + LUT(t_m, t_{m+1}, b_{m,m+1})) \tag{3.11}$$

$$R_j = L - l_{t_u} - \sum_{m=j}^{u-1}(l_{t_m} + LUT(t_m, t_{m+1}, b_{m,m+1})) \tag{3.12}$$

3.4.2 Constructing a Graph Model

After computing the X-Scope of all cell instances, a directed acyclic graph model $G = (V, E)$ can be constructed to compute the exact locations of all the cell instances with optimal HPWL. The graph can be divided into u columns, where the vertices in the jth column represent all possible locations where we can place s_j. The graph is constructed as follows.

For any cell instance s_j with X-Scope $[L_j, R_j]$, $R_j - L_j + 1$ number of vertices are created. Similarly, $R_{j+1} - L_{j+1} + 1$ number of vertices are created for s_{j+1}. Denote the ith vertex for cell instance s_j as v_i^j and its location as $L(v_i^j)$. There is an edge connecting two adjacent nodes v_i^j and v_k^{j+1} if the following condition holds:

$$L(v_i^j) + l_{t_j} + LUT(t_j, t_{j+1}, b_{j,j+1}) \leq L(v_k^{j+1}) \tag{3.13}$$

where the weight of the edge is assigned to be the HPWL increase when placing s_{j+1} at $L(v_k^{j+1})$.

There is a source vertex s which connects to all the vertices that belongs to s_1. The weight of the edge connecting s and v_i^1 is assigned to be the HPWL increase when we place s_1 at v_i^1. There is also a sink vertex t which connects to all the vertices in the last column, where the weight of the edges are all 0. After constructing the graph, a shortest path algorithm is applied to compute the exact locations of all instances with optimal HPWL. The overall flow of the refinement procedure with optimal HPWL is shown in Algorithm 4.

A simple example is shown in Fig. 3.6 to illustrate our approach. There are two cell instances s_1 and s_2 and nine placement sites in the layout. The s_1 is involved with two nets, $\{s_1 f_1\}$ and $\{s_1 f_2 f_4\}$ respectively. The s_2 is involved with two nets, $\{s_2 f_3\}$ and $\{s_2 f_2 f_4\}$ respectively. The CPA-friendly distance between s_1 and s_2 is one placement site. Based on the input layout, we can compute the X-Scope of s_1 as $[1, 4]$, and X-Scope of s_2 as $[5, 8]$. The graph is constructed based on the X-Scope information of all cell instances. Finally,

Algorithm 4: CPA-Aware Refinement with Optimal HPWL

```
1  begin
2      Read Max-SAT solutions;
3      Initialize graph G as empty;
4      S ← All cell instances in a row;
5      LUT ← Update spacing for all cell adjacencies;
6      w ← size of S;
7      for i ← 1 to w do
8          [L_i, R_i] ← X-Scope for instance s_i;
9      end
10     for i ← 1 to w do
11         G ← Add R_i − L_i + 1 vertices;
12     end
13     G ← Add source and sink vertices;
14     Find shortest path in G;
15     Update cell locations;
16 end
```

a shortest path algorithm is utilized to compute the solution with optimal HPWL, which is shown in Fig. 3.6 (b).

3.4.3 Honoring Cell Displacement

In practice, minimizing the cell displacement is also one of the important objectives during the detailed placement stage. To achieve this, we can simply enforce a sliding window on each cell instance in the layout. The length of the window is set to be L_w, with its center sitting at the location of the cell instance. When constructing the graph model, the range that we can place a cell instance is set to be the overlapping part between its X-Scope and the sliding window.

3.5 Experimental Results

The algorithm is implemented in C++ and run on a Linux server with 4GB RAM and a 2.53 GHZ CPU. All benchmarks are generated using NanGate FreePDK45 Generic Open Cell Library [28], which is available online. The standard cells are randomly selected from the cell library, and are aligned adjacently in different rows of a chip. d_{min} is set to be 82 nm. Wires on the M1 layer are used for all experiments. Nets for all cells are randomly

Table 3.1: Experimental Results

Test	# P	# BP	# Clause	# Nets	SAT Runtime (s)	Area Overhead		
						Area* (nm^2)	Area (nm^2)	Improve (%)
C_1	6155	32	303	674	0.07	42160	3400	91.9
C_2	24356	39	484	2880	0.25	122740	27200	77.8
C_3	54033	39	529	6268	0.56	343400	86700	74.8
C_4	96078	39	541	11147	0.94	473620	120700	74.5
C_5	149951	39	541	17415	1.45	805913	178500	77.9
Ave.	66115	38	479.6	7677	0.65	357567	83300	79.4

Note: The column named "Area*" shows the area overhead of the approach **Fix-Color**, while the column named "Area" shows the results of our Max-SAT algorithm.

Table 3.2: Experimental Results

Test	# P	# BP	# Clause	# Nets	SAT Runtime (s)	HPWL Compare		
						Initial HPWL (mm)	Final HPWL (mm)	Improve (%)
C_1	6155	32	303	674	0.07	23.43	23.41	0.10
C_2	24356	39	484	2880	0.25	192.85	192.40	0.23
C_3	54033	39	529	6268	0.56	634.08	631.93	0.34
C_4	96078	39	541	11147	0.94	1504.87	1500.13	0.32
C_5	149951	39	541	17415	1.45	2897.26	2890.97	0.22
Ave.	66115	38	479.6	7677	0.65	1050.50	1047.77	0.24

generated, where each cell is connected with three to ten nets. Each net contains five cell instances. Width of the placement site is set to be 10 nm. The Linux version of MSUncore Max-SAT solver is used in the experiments, which is available online [69].

The results of constrained pattern assignment are shown in Table 3.1 and Table 3.2. To give more insights of the effectiveness of our approach, the approach named **FixColor** is also implemented. In this approach, the legal TPL decompositions of all cells in the library are fixed before the placement stage. The algorithm can be easily implemented since the solution graph proposed in [48] explores all legal solution spaces for any cell. For each cell, we randomly pick up a path in its solution graph, which guarantees to correspond to a legal TPL decomposition, and set it as its initial TPL decomposition. Once the TPL decompositions of all cells are known, the

layout is sequentially explored to identify the cell boundaries where coloring conflict exists. For all such cell boundaries, a minimum number of placement sites is inserted to remove the coloring conflicts. The final area overhead equals to the total area of all inserted placement sites.

Since the initial coloring solutions are randomly picked, the **FixColor** algorithm is invoked multiple times to capture the facts that different TPL decomposition leads to different area overhead. In the experiments, the **FixColor** algorithm is run 15 times for all benchmarks, and final area overhead shown in column 7 is taken as the average of the 15 runs.

The total number of polygons, boundary polygons, clauses in the SAT formulation, and nets are shown in columns 2, 3, 4, and 5 respectively. Column 6 shows the runtime for solving the weighted partial Max-SAT problem. The comparisons between the area and HPWL are detailed in the last six columns. The area of the baseline algorithm is shown in column named "$Area^*$", while the results of our approach is shown in the column "$Area$" in Table 3.1 and Table 3.2 respectively. Compared with the baseline algorithm, our approach achieves significant area overhead reductions by as much as 79.4% on average. The results suggest that fixing the cell colors beforehand could lead to much inferior results. For our approach, the solution space of all cells are also pertained, since a CPA-friendly solution graph is computed for each type of cell respectively. This leaves more flexibilities to achieve a CPA-friendly layout while simultaneously minimizing the area overhead during the detailed placement stage.

As shown in column 2, the number of boundary polygons increases only by a small amount as the layout grows. In practice, the number of boundary polygons is limited since there are limited types of cells in the cell library. Our proposed algorithm has the capability to handle very large layout as long as the number of cells in the cell library is limited, which is true for many industry designs. As shown in column 3, the number of clauses in the SAT formulation is limited for all benchmarks. The runtime for all benchmarks are within two seconds. As for the HPWL, our approach consistently reduces the HPWL for all benchmarks. Overall, it achieves a 0.24% improvement on HPWL on average. This clearly verifies the effectiveness of our algorithm.

3.6 Conclusions

In this chapter, we integrate the flow of detailed placement and TPL decompositions, which guarantees a CPA-friendly layout in the early design stage and avoids costly modifications after placement and routing. We propose a partial weighted Max-SAT based approach which guarantees to compute a CPA-friendly layout while minimizing the area overhead. A novel graph model is also proposed to find the exact locations of all cells with optimal HPWL in a standard cell row. Compared with a the approach of fixing the cell colors beforehand, the area overhead reduction is as much as 79.4% on average for all the benchmarks. Better results are also reported on HPWL for all benchmarks.

CHAPTER 4

AN EFFICIENT LINEAR TIME TRIPLE PATTERNING SOLVER

4.1 Introduction

As the semiconductor industry is advancing to the 14/10 nm technology node, numerous technology difficulties have to be resolved before turning the technology into readily available products. Lithography is among one of the most challenging difficulties. Double patterning lithography (DPL) [17, 29] is already reaching its limit at the 20 nm technology node, and cannot be further pushed to the next technology node. Several other techniques have been proposed, such as *extreme ultra-violet* (EUV) [35] lithography, *E-beam* direct write [10, 12] and DSA [70] techniques. For EUV, source power is still an unresolved issue before making it happen. The low throughput of the E-beam limits its ability to be massively used in industry. DSA is a new emerging technique to conquer the physical limitation of traditional lithography techniques. However, currently it can only handle 1 D patterns and is not ready to be used in practice. TPL naturally extends the merits of DPL with one more mask, which triples the printing resolution of the widely used 193 nm immersion lithography. It is one of the most promising techniques which enables 14/10 nm designs.

The general TPL problem is a 3-coloring problem, which is a well-known NP-complete problem. There have been many research efforts on TPL problems [25, 27, 48, 53, 62, 63, 71, 72, 73]. An ILP based algorithm is proposed in [25] to handle general TPL decompositions. A semidefinite programming based approximation algorithm is also proposed to accelerate the runtime. However, the ILP-based algorithm is difficult to scale up and the modified semidefinite programming is losing the optimality. A graph-based heuristic is proposed in [27], but it cannot guarantee finding a solution when one exists. A TPL algorithm for standard-cell-based designs is proposed in [48], which is

guaranteed to find a solution when one exists. Nevertheless, their algorithm involves an excessive amount of recomputations, which incurs significant runtime penalties. Moreover, the algorithm utilizes stitch candidates which are located at corners. In practice, corner stitches are not preferred since they are highly vulnerable to process variations and could lead to functional errors in the chip. Some graph heuristics are proposed in [53]. An approach which finds all legal TPL stitch candidates is also proposed. Lookup tables are constructed to improve the runtime of the algorithm. Similar to [27], optimality is also not guaranteed in this approach.

In this chapter, we propose an integrated flow for standard-cell-based triple pattering lithography, which guarantees to find a TPL decomposition if one exists. Unlike the previous work which simply place stitch candidates on corners, we seamlessly integrate the stitch identification method in [53] into our algorithm, which enables us to compute a legal TPL decomposition with optimal number of stitches. The contributions are summarized as follows:

- A TPL algorithm is proposed which essentially explores all solution space incorporating all legal stitch candidates, and guarantees to compute a TPL decomposition with the optimal number of stitches if one exists.

- A novel graph model is proposed to minimize the number of vertices in the solution graph. A fast approach is also proposed which achieves simultaneous memory and runtime improvement compared with the state-of-the-art TPL algorithm in [48].

- Our proposed algorithm is very efficient and achieves 39.1% runtime improvement and 18.4% memory reductions compared with the state-of-the-art TPL algorithm on the same problem.

The rest of the chapter is organized as follows: preliminaries of the TPL problem are discussed in Section 4.2. Our algorithm will be presented in Section 4.3 and Section 4.4. Section 4.5 shows the experimental results, followed by a conclusion in Section 4.6.

4.2 Preliminaries

Preliminaries of standard-cell-based designs and the previous TPL algorithm are discussed here.

4.2.1 Standard-Cell-Based Designs

We use the same assumptions as what are used in previous works [48, 62]. A layout is composed of different standard cells from the cell library. All standard cells in the library have exactly the same height. There are power rails going from the leftmost to the rightmost of a cell. A layout consists of multiple standard cell rows, with power rails perfectly isolate the features within different rows and cells aligned adjacently within the same row. As with previous works, features on the M1 layer are used since it is the densest layer with the most complex features. An example of a standard-cell-based layout is shown in Fig. 4.1. The six instances are composed of three types of cells from the cell library.

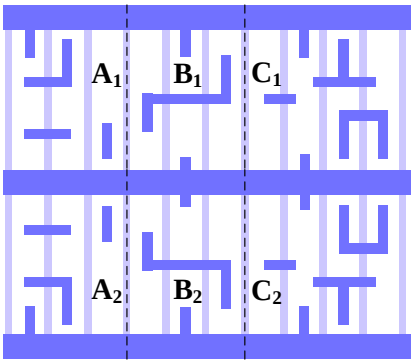


Figure 4.1: Layout of standard-cell-based designs with two rows. The six instances are composed of three types of cells, A, B, and C from the cell library.

4.2.2 Previous TPL Algorithm

A TPL algorithm targeting on standard-cell-based designs is proposed in [48], which guarantees finding a solution if one exists for stitch-free designs. A set of *cutting lines* are constructed based on the left boundaries of all the

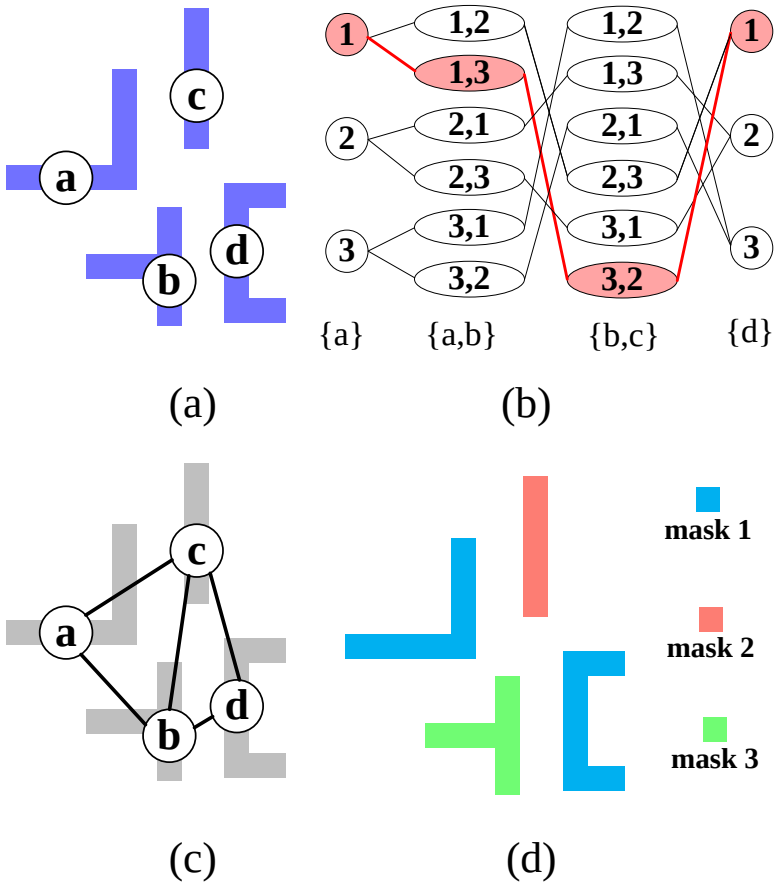


Figure 4.2: Previous TPL algorithm in [48]. (a) Input layout. There are four cutting lines, with their cutting line sets as $\{a\}$, $\{a, b\}$, $\{b, c\}$, and $\{d\}$ respectively. (b) Solution graph. Different numbers here denote different masks. The highlighted path is a legal TPL decomposition. (c) Constraint graph. (d) Final decomposition which corresponds to the above highlighted path. Different colors represent different masks.

features. The polygons that intersect with the same cutting line are defined as *cutting line set*. TPL solutions for each cutting line are enumerated based on the *constraint graph*, with each solution corresponding to a vertex in the *solution graph*. Compatible vertices on adjacent cutting lines are connected together. A more detailed description of the algorithm is discussed in a previous paper [48].

We show a simple example in Fig. 4.2 to illustrate how the previous TPL algorithm works. There are four features in the layout, with its solution graph and constraint graph shown in Fig. 4.2 (b) and Fig. 4.2 (c) respectively. Final TPL decomposition is shown in Fig. 4.2 (d).

4.2.3 Problem Definition

Given a standard-cell-based row structure layout and a minimum coloring distance d_{min}, our objective is to find a legal triple patterning decomposition while minimizing the number of stitches.

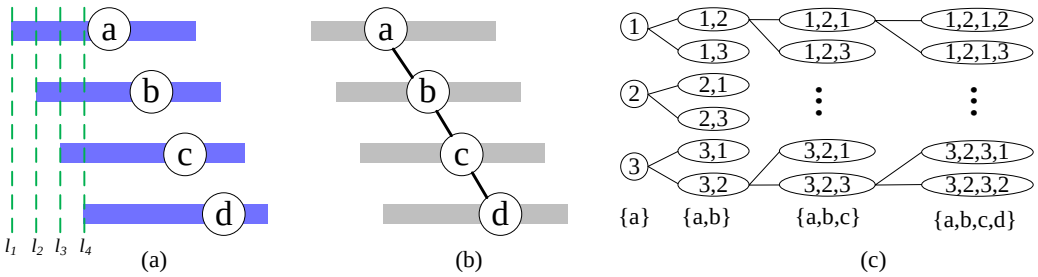


Figure 4.3: Example of how the previous algorithm works. (a) Input layout with four features. The four cutting lines are shown in green dotted lines, with the cutting line sets are $\{a\}$, $\{a, b\}$, $\{a, b, c\}$, and $\{a, b, c, d\}$ respectively. (b) Constraint graph of the input layout. (c) Solution graph computed by the previous algorithm [48]. There are totally 45 vertices in the graph.

4.3 An Optimal Algorithm

In the following, we will formally introduce the optimal TPL algorithm, which is guaranteed to find a TPL solution with optimal number of stitches if one exists. The novel graph model which minimizes the number of vertices in the solution graph is also discussed. Since we are addressing the same problem as the previous paper [48], the same terminologies including *cutting line*, *cutting line set*, *constraint graph*, and *solution graph*, are reused for consistency. The four concepts are illustrated in Fig. 4.2. As different rows are separated by power tracks and can be solved independently, the following discussions are based on the layout in a single standard cell row.

4.3.1 Limitations of Previous Approach

Although the algorithm in [48] is able to find a stitch-free decomposition if one exists, it may uses an excessive amount of runtime and memory than

necessary. The example in Fig. 4.3 is used to show the limitations of the previous approach. There are four features in the examples, with the constraint graph shown in Fig. 4.3 (b). The corresponding solution graph is shown in Fig. 4.3 (c). One can easily observe that a huge number of nodes are computed to explore all legal solution spaces.

In the following, we propose a novel graph model which minimizes the number of vertices in the solution graph without losing the optimality of the approach. An approximation approach is also proposed to achieve simultaneous runtime and memory reductions.

4.3.2 A Novel Graph Model

To reduce the number of nodes in the graph, we need to carefully compute the cutting line sets for each of the cutting lines. Intuitively, cutting line sets with smaller number of features indicate less number of nodes in the graph, thus reducing memory and runtime. Our objective is to compute the cutting line sets for all cutting lines, which leads to minimum number of vertices in the solution graph.

Denote the set of polygons as $P = \{p_1, p_2, ..., p_n\}$ where n is the number of features in the layout. We assume the set of polygons are already sorted in non-decreasing order according to their left boundaries. Denote the set of cutting lines as $L = \{l_1, l_2, ..., l_m\}$ where m is at most n. After performing polygon dummy extension which is proposed in [48], the cutting line sets for all the cutting lines can be computed. Denote the cutting line sets as $S = \{s_1, s_2, ..., s_m\}$, where s_i is the cutting line set which corresponds to the cutting line l_i. The graph can be constructed as follows.

We sequentially go through all the cutting line sets. For cutting line set s_i, all subsets of s_i which contains p_i are enumerated. For the jth subset of s_i^j, a vertex v_i^j is created in the graph. Denote the set of polygons for s_i^j as p_i^j. For any two adjacent vertices v_i^j and v_{i+1}^k, an edge is added if for any polygon in p_{i+1}^k, all its conflicting polygons appears in p_i^j. Denote the number of legal TPL solutions for s_i^j as n_i^j. The weight of the edge is assigned according to the following two scenarios:

1. If p_i^j is a subset of p_{i+1}^k, the weight is assigned as $n_{i+1}^k - n_i^j$.

2. Otherwise, the weight is assigned as n_{i+1}^k.

If p_i^j is a subset of p_{i+1}^k, the cutting line of l_i^j and l_{i+1}^k can be merged together. Therefore, the weight of the edge is subtracted by n_i^j to reflect the real number of vertices needed for that cutting line set. Otherwise, the weight of the edge is simply assigned as n_{i+1}^k, which is the number of vertices needed to represent all legal TPL solutions of p_{i+1}^k.

There is also a source node s which connects to all vertices in the first cutting line, and a sink t which connects to all vertices in the last cutting line. By applying the shortest path algorithm from s to t, the cutting line sets which minimize the number of vertices in the solution graph are obtained. After that, any two adjacent cutting line sets are merged together if one of the set is a subset of the other one.

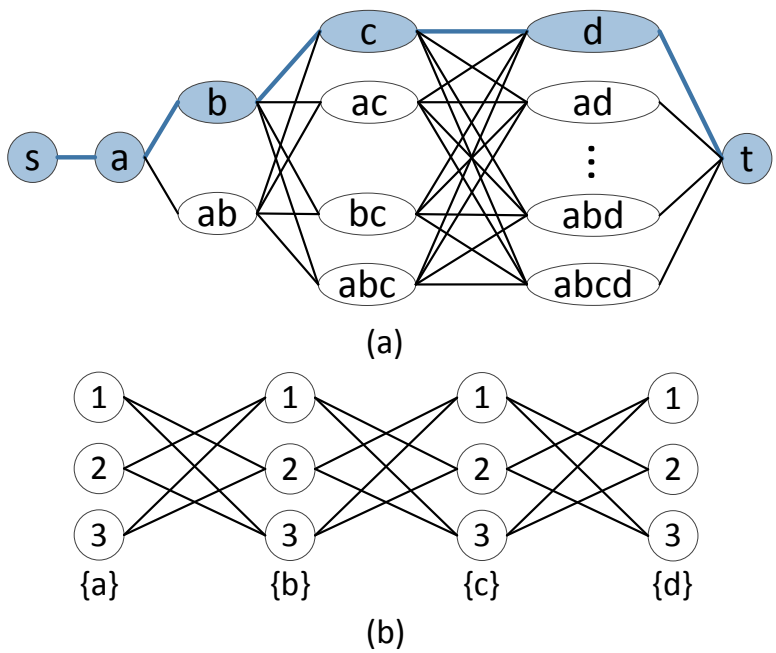


Figure 4.4: (a) The graph model for the layout in Fig. 4.3. Cost of the edges are not shown here for simplicity. The highlighted path corresponds to the cutting line sets that lead to the minimum number of vertices in the solution graph. (b) Solution graph corresponding to the highlighted path. There are totally 12 vertices in the graph.

The layout in Fig. 4.3 is used to illustrate the procedures. There are four cutting line sets $\{a\}$, $\{a, b\}$, $\{a, b, c\}$, and $\{a, b, c, d\}$ in the example. For cutting set $\{a\}$, there is one vertex in the graph. Similarly, there are two subsets for the second cutting line sets, which are $\{b\}$, and $\{a, b\}$ respectively. Two vertices are created in the graph. Edges are added between compatible

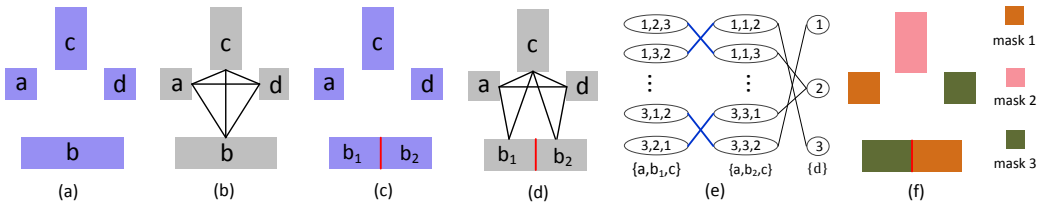


Figure 4.5: (a) Input layout. (b) Constraint graph. (c) Layout after inserting stitch candidate. (d) Constraint graph after inserting stich candidates. (e) Solution graph, with the thick blue edges of weight 1. (f) Final TPL decomposition with one stitch.

nodes for these two cutting lines. The same procedures are repeated for all cutting lines, and the complete graph is shown in Fig. 4.4 (a). The new cutting line sets for each cutting line are extracted based on the shortest path in the graph. Based on the new cutting line sets, the solution graph with minimum number of vertices are constructed, which is shown in Fig. 4.4 (b). Compared with the solution graph in Fig. 4.3 (c), the number of vertices is reduced by 73.3% while the completeness of the algorithm is not affected.

4.3.3 Computing Cutting Line Sets

By applying the shortest path algorithm on previous graph model, the cutting line sets which lead to minimum number of vertices in the solution graph are extracted. However, constructing such a graph model is expensive. For each original cutting line set, we need to compute all its subsets and enumerate the number of legal TPL solutions for all the subsets. It is very expensive to accurately compute the number of legal TPL solutions for a given graph. If enumerations are applied, the proposed graph model suffers from high runtime penalties. The key observation here is that to reduce the number of TPL solutions for a cutting line set, we need to limit the number of features in the cutting line set. Intuitively, a feature should appear in as less cutting lines as possible. The proposed approximation approach works as follows.

Given a cell-based layout, the constraint graph and the set of cutting lines $L = \{l_1, l_2, ..., l_m\}$ are computed. Define the *Influenced Cutting Line(ICL)* of a polygon as the set of cutting lines that intersect with the polygon. For polygon a in Fig. 4.3 (a), the ICL is $ICL_a = \{l_1, l_2, l_3, l_4\}$. Similarly, $ICL_b = \{l_1, l_2, l_3, l_4\}$. Note that the ICL of a polygon is always continuous.

The size of ICL is minimized for each polygon using the following technique.

For each polygon p_i, all polygons that conflict with itself are identified. Denote the one with the largest left boundary as p_j. Denote the last cutting line in ICL_i as l_t^i and the first cutting line in ICL_j as l_s^j. ICL_i is expanded (if $l_t^i < l_s^j$) or shrunk (if $l_t^i \geq l_s^j$) to l_{s-1}^j, which is the cutting line right before l_s^j. When the left boundaries of p_i and p_j are the same or there is no conflicting polygon with p_i, its ICL is simply shrunk to include only the leftmost cutting line in the original ICL. The cutting line set of l_i is computed as all polygons whose ICL contains the cutting line l_i. Any two adjacent cutting line sets are merged together if one of them is a subset of the other.

Lemma 4. *For any feature p_i which first appears in the cutting line set s_j, all conflicting features of p_i with smaller left boundaries are included in the cutting line set s_{j-1}.*

Proof. For any feature p_i, denote the set of features with smaller left boundaries and conflict with p_i as c_f^i. Denote the cutting line of p_i as l_j. Clearly, the ICL of any feature in c_f^i will be expanded to l_{j-1}, which means that all features in c_f^i appear in cutting line set s_{j-1}. □

For any feature p_i, denote all its conflicting features with smaller left boundaries as c_f^i. Assume p_i first appears in cutting line set s_j. The way we compute the cutting line sets ensures that all features in c_f^i appear in the cutting line sets s_{j-1}. Since all conflicting polygons of p_i appears in s_{j-1}, it enables an incremental computation of the solution graph as we are walking through all cutting line sets. Each time a new cutting line set comes in, it is sufficient to look back one cutting line set to ensure the legality of the solution graph.

For feature a in the layout shown in Fig. 4.3 (a), $ICL_a = \{l_1, l_2, l_3, l_4\}$. After applying the above procedures, $ICL_a = \{l_1\}$. Similarly, the ICL of feature b, c, and d can be computed as $ICL_b = \{l_2\}$, $ICL_c = \{l_3\}$, and $ICL_d = \{l_4\}$ respectively. The cutting line set corresponding to $L = \{l_1, l_2, l_3, l_4\}$ are $\{a\}$, $\{b\}$, $\{c\}$, and $\{d\}$ respectively. The solution graph based on these cutting line sets is exactly the same with the one shown in Fig. 4.4 (b).

4.3.4 Stitch Candidates

The approach of finding all legal TPL stitch candidates in [53] is embedded into our algorithm. After computing the stitch candidates, all the techniques discussed above can be applied here. When stitches exist, the solution graph becomes a weighted graph, with the weight of an edge computed as the number of stitches needed for the connected vertices. The shortest path algorithm is applied to get a TPL decomposition with optimal number of stitches.

An example is shown in Fig. 4.5 to further illustrate the algorithm. There are four features in the layout, where the constraint graph forms a clique. One stitch candidate is computed, which is shown in thick red edge in Fig. 4.5 (c). After inserting the stitch candidate, the new constraint graph and solution graph are computed, which are shown in Fig. 4.5 (d) and Fig. 4.5 (e) respectively. Final result is shown in Fig. 4.5 (f), where there is one stitch in the decomposition.

4.3.5 Analysis of the Algorithm

The time complexity of the algorithm is $O(n + s)$, where n is the number of features and s is the number of stitch candidates in a layout. We first show that the size of any cutting line set s_i is bounded. Denote the x coordinate of current cutting line as x_i. For any feature, it is extended by at most d_{min}, which is a constant for 193 nm immersion lithography. For any feature appearing in cutting line set s_i, either it physically intersects with cutting line $l = x_i$, or the extended part intersects with $l = x_i$. For the first case, the number is bounded as the height of a standard cell is fixed. For the latter case, the x coordinates of the right boundaries for the polygons has to be within $[x_i - d_{min}, x_i]$, which is also limited. Denote the maximum number of features for the two cases as u, which is just a constant. For any cutting line, at most 3^u number of solutions are enumerated. For two adjacent cutting lines, at most 3^{2u} number of checks are needed to find compatible solutions. By going through at most $n + s$ cutting lines, the time complexity is $O((3^{2u} + 3^u)(n + s))$, which is $O(n + s)$.

4.3.6 Reducing Stitch Candidates

The number of stitch candidates computed by the approach in [53] are huge. Since all features in a layout are possibly segmented and stitches are inserted, an excessive number of stitches candidates are computed. In the following, we show that our algorithm can accurately identify the features that need stitch on the fly, thus significantly reducing the number of stitch candidates with neglectable runtime penalties.

After computing all the cutting line sets $S = \{s_1, s_2, ..., s_m\}$, the solution graph are constructed. For any two adjacent cutting line sets s_i and s_{i+1}, if there are no compatible decompositions for the two cutting line sets, stitches have to be inserted. All the features within s_i and s_{i+1} are features that potentially need stitches, and denote the features as $F_{i,i+1}$. After that, all legal stitch candidates are computed for polygons in $F_{i,i+1}$. We go back to the cutting line set where none of the feature belongs to $F_{i,i+1}$, and the solution graph are recomputed from that point on.

The algorithm can also be applied statically, where the techniques in [25, 72] are used to find relevant polygons for inserting stitches. All legal stitches are computed before applying our algorithm to get legal decompositions. Note that the algorithm is highly flexible which does not depend on a particular set of stitch candidates. Given any legal stitch candidates, the algorithm guarantees to compute a TPL decomposition with optimal number of stitches if one exists.

4.3.7 Power Tracks

As power and ground rails go through the whole layout and appear in all cutting line sets, they can be preassigned to some masks to avoid recomputations. Similar technique are also adopted in the previous paper [48]. There are two base cases. We can assign all power rails on mask 1 and ground rails on mask 2 respectively, or we can assign them on mask 1 at the same time. All other combinations can be obtained from the base cases by rotating colors.

4.4 Hierarchical Approach

For standard-cell-based designs, millions of chip elements are typically composed from several hundreds or thousands types of cells in the standard cell library. To further speed up the algorithm, the solution graphs of different types of cells can be computed and stored in a lookup table. The solution graph are reused when constructing a whole chip decomposition. When reusing the solution graphs, boundary adjacency between different cell could introduce additional constraints and stitch candidates. In the following, we will discuss the details involved in the hierarchical approach.

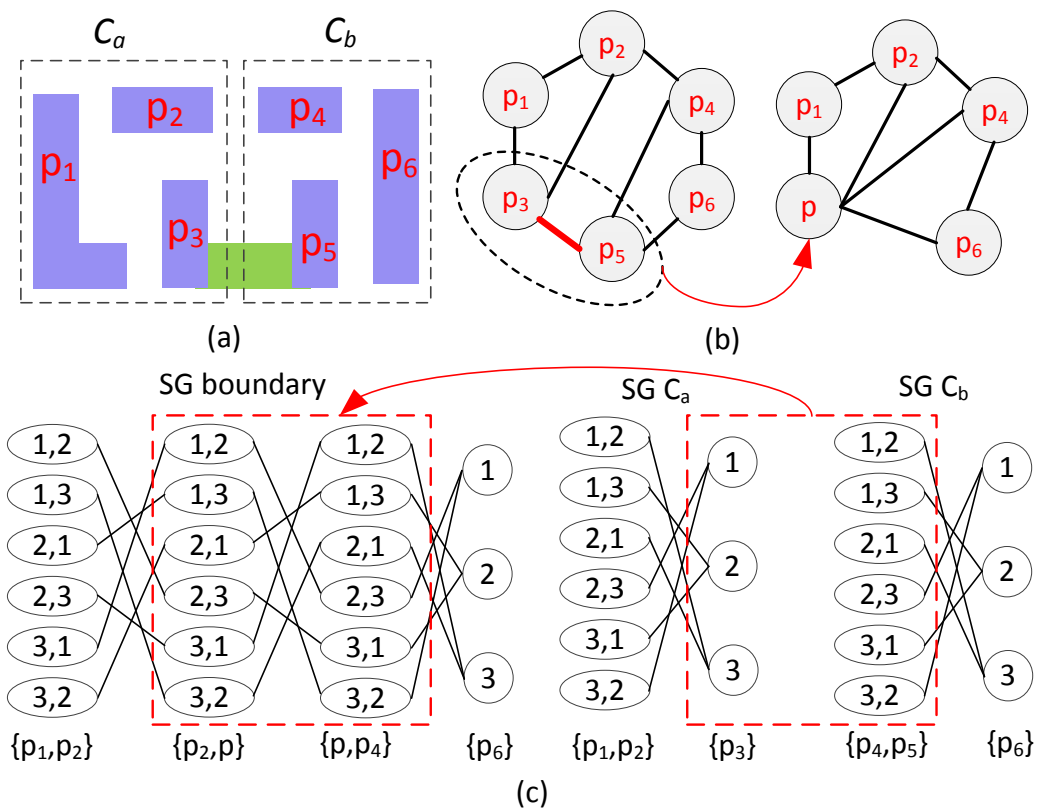


Figure 4.6: (a) Input layout with two cells c_a and c_b respectively. There are four boundary polygons, p_2, p_3, p_4, and p_5 respectively. Boundary connections between p_3 and p_5 are highlighted in green. (b) The updated constraint graph. Connected vertices are merged together. (c) Final solution graph, with the solution graph on cell boundaries recomputed.

4.4.1 Boundary Constraints

Boundary constraints exist between two adjacent standard cells in the given layout, either due to boundary connections, or due to boundary conflicts. Define a *boundary polygon (BP)* as a polygon that conflicts or connects with other polygons in adjacent cells. If features are connected to each other, they are treated as one single feature, since they have to be assigned to the same mask to avoid stitches. For any two adjacent cells, the boundary polygons are computed. The cutting line sets along the boundaries are also recalculated. For the cutting line sets that are different from original ones at cell boundaries, the corresponding part of the solution graph are calculated from scratch. For the unchanged cutting line sets, the part of the solution graph are reused from the previously stored lookup table. By reusing the solution graph for each type of cell, a limited part of the whole chip solution graph are recomputed, which improves the overall runtime.

In Fig. 4.6, there are two cells with six features in total. The boundary polygons are p_2, p_3, p_4, and p_5 respectively. The constraint graph for cell boundaries are recomputed since conflicts and connections exist between cell boundaries, and the updated graph is shown in Fig. 4.6 (b). p_3 and p_5 are merged into one feature since connection exists between these two features. The final solution graph is shown in Fig. 4.6 (c), where the solution graph of cell boundaries is updated based on the new constraint graph.

4.4.2 Boundary Stitch Candidates

If features that potentially need stitch candidates appear between cell boundaries, the stitch candidates can be updated on the fly. This step is similar to boundary constraints, except that the cutting line sets are computed after inserting all legal stitch candidates. Note that boundary stitch candidates are only needed when features that need stitches exist between two adjacent cells. If compatible nodes exist between two adjacent cutting lines on cell boundaries, no stitches are needed.

By reusing the solution graph and considering boundary constraints and stitches, the solution graph is incrementally updated while the correctness of the graph is maintained. For designs without stitches, the completeness of the solution graph is not affected. For complex designs with stitches, the

number of stitch candidates is minimized as they are only considered when stitches are needed at potential locations.

Table 4.1: Comparisons of Runtime and Memory with Previous Algorithm

Test Cases	n	Tracks	Runtime			Memory		
			t_1 (s)	t_2 (s)	Improve (%)	m_1 (MB)	m_2 (MB)	Improve (%)
C1	106690	143	14	8	40.8	7.9	6.7	15.6
C2	674841	358	70	39	45.0	13.8	11.9	13.7
C3	2695803	715	273	149	45.5	31.8	28.9	9.3
C4	10782073	1429	1088	589	45.8	99.0	93.7	5.4
C5	26949406	715	2709	1481	45.3	264.1	240.0	9.1
C6	179201	143	32	22	33.2	12.5	9.4	24.5
C7	904292	322	147	98	33.3	24.6	16.2	34.2
C8	2695803	715	722	471	34.8	57.7	41.7	27.8
C9	10031115	1072	1589	1050	33.9	91.8	70.0	23.8
C10	17813611	1429	2833	1901	32.9	136.9	108.9	20.5
Ave.	7548983	633	948	581	39.1	82.3	62.7	18.4

Note: t_1 and m_1 are the runtime and memory in the previous paper [48], while t_2 and m_2 are the results of our proposed algorithm.

4.5 Experimental Results

The algorithm is implemented in C++ and run on a Linux server with 8GB RAM and a 3.0 GHZ CPU. The algorithm in [48] is also implemented to compare with our approach. For fair comparisons, the same benchmarks are used as that used in the previous paper [48], which are generated from NanGate FreePDK45 Generic Open Cell Library [28]. d_{min} is set to be 82 nm. Wires on the M1 layer are used for all experiments. For generating stitch candidates, the same setting are used as that in [53]. The results are discussed as follows.

A comprehensive comparisons of our algorithm with the previous algorithm in [48] are detailed in Table 4.1 and Table 4.2. Hierarchical implementations are used to generate these results. The comparisons between the runtime and memory are shown in Table 4.1, and the comparisons between runtime and stitches are shown in Table 4.2. The same stitch candidates are used in both algorithms for a fair comparison.

Table 4.2: Comparisons of Runtime and Stitches with Previous Algorithm

Test Cases	n	Tracks	Runtime			Stitches		
			t_1 (s)	t_2 (s)	Improve (%)	s	s_1	s_2
C1	106690	143	14	8	40.8	0	0	0
C2	674841	358	70	39	45.0	0	0	0
C3	2695803	715	273	149	45.5	0	0	0
C4	10782073	1429	1088	589	45.8	0	0	0
C5	26949406	715	2709	1481	45.3	0	0	0
C6	179201	143	32	22	33.2	82736	3420	3420
C7	904292	322	147	98	33.3	419907	17146	17146
C8	2695803	715	722	471	34.8	2064944	83916	83916
C9	10031115	1072	1589	1050	33.9	4655445	188854	188854
C10	17813611	1429	2833	1901	32.9	8254596	334642	334642
Ave.	7548983	633	948	581	39.1	1547762	62798	62798

Note: t_1 and s_1 are the runtime and the number of stitches in the previous paper [48], while t_2 and s_2 are the results of our proposed algorithm. "s" shows the number of stitches candidates in the benchmark.

Compared with the previous algorithm, the runtime is further improved by 39.1% on average. For the benchmark $C5$ with over 26 million features, the runtime is approximately reduced by half. In terms of the memory usage, the improvements are more significant on designs with stitches than that without stitches. For designs with stitches, the size of the cutting line set is typically larger, which would introduce more vertices in the graph. If the size of a cutting line set is increased by one, the number of vertices could increase up to three times. Since the sizes of all cutting line sets are optimized and redundant cutting line sets are eliminated, the memory reductions are more prominent on complex designs with stitches. On average, our algorithm uses 18.4% less memory than the previous algorithm.

The comparisons of the number of stitches are shown in the last three columns. All the stitch candidates are generated dynamically when features that potentially need stitches are found in the circuit. The same stitch candidates are embedded into the previous approach to compute the solution with optimal number of stitches. Not surprisingly, the number of stitches achieved by both algorithms are exactly the same for all benchmarks, since optimality in terms of stitches is guaranteed in both approaches. Overall, the new approach is able to achieve simultaneous runtime and memory re-

ductions while guaranteeing the optimality in the number of stitches. This clearly verifies the effectiveness of the new algorithm.

4.6 Conclusions

In this chapter, we propose a linear time triple patterning solver that guarantees to compute a TPL decomposition with optimal number of stitches if one exists. A novel graph model is proposed to reduce the memory requirement of the algorithm. A fast approach is also proposed to achieve simultaneous memory and runtime reductions compared with state-of-the-art TPL algorithm. To reduce the number of stitch candidates, features that potentially need stitches are accurately identified, where all legal stitch candidates are computed. This algorithm is expected to relieve the manufacturing bottleneck in advanced technology node.

CHAPTER 5

PERFORMANCE EVALUATION CONSIDERING MASK MISALIGNMENT IN MULTIPLE PATTERNING DECOMPOSITION

5.1 Introduction

As the feature size of the transistors keeps shrinking, the difficulties of fabricating the small features also keep increasing. Traditional optical lithography faces great challenges when dealing with these small features, mainly due to the inherent physical limitations of light diffusions. Single exposure lithography is already reaching its limit beyond the 20 nm technology node. Several next-generation lithography techniques have been studied, such as *extreme ultra-violet* (EUV) lithography [35] and *E-beam* [12]. However, source power remains challenging for EUV, and low productivity of E-beam makes it difficult to be used in volume production in industry. Multiple patterning decomposition coupling with tradition litho technologies serves as one of the most cost-effective ways to tackle the manufacturing bottlenecks.

For the most widely used 193 nm immersion lithography, double patterning (DPL) is required at the 20 nm technology node, where the features are decomposed into two masks and go through two litho exposures. For the 14/10 nm technology node and beyond, triple patterning decomposition (TPL) comes into picture where the features are decomposed into three masks, with each mask going through a separate litho process. There have been several works on MPL decompositions [17, 25, 27, 48, 53, 56, 63] in the literature. However, none of the these works are considering mask alignment explicitly, which is prominent and inevitable in advanced technology node.

On the other hand, there are several works in the literature that analyzed the effects of mask misalignment on circuit performance for a given decomposition [16, 74, 75, 76, 77]. The authors in [16, 74] studied the adverse effects on timing for overlay errors in DPL. However, mask misalignment is not explicitly captured in their models, and the analysis is not fed back

into the decomposer to get a better decomposition. Commercial tools like StarRC [75] from Synopsys Inc. simply extract the min/max capacitance to evaluate the timing degenerations, which often leads to pessimistic estimations. Recent work in [76] focused on the evaluating the influence of mask misalignment for DPL decompositions on static timing analysis. However, the approach is only targeting on DPL and cannot be extended to MPL with $k > 2$.

Mask misalignment will adversely affect the quality of a circuit in two folds. On one hand, timing closure becomes more challenging considering mask misalignment, since misalignment between different masks leads to significant variations in total coupling capacitance, thus complicates the process of performing timing analysis. Previous study shows that a 6 nm misalignment causes a 15% error in coupling capacitance and a 5% error on total capacitance, whereas a 2 nm displacement creates approximately a 5% error for coupling capacitance and 2% error for total capacitance [78]. On the other hand, MPL decomposers unaware of mask misalignment could lead to increased power dissipations and reliability issues. Even worse, if designers are doing timing analysis unaware of the adverse effects of mask misalignment, timing violations may occur for certain decompositions and they potentially lead to functional errors of the circuit.

An example is shown in Fig. 5.1. There are three features, with the normal spacing 100 nm. Assume DPL is used and there is a worst-case 10% mask misalignment, the possible coupling capacitance variations are shown in the right part of Fig. 5.1. One can clearly see that the worst-case capacitance deviates as much as 10% from the normal capacitance.

In this chapter, we analyzed the effects of mask misalignment for MPL, and mathematically proved the upper bound of the coupling capacitance induced by mask misalignment. We aimed at computing a tight upper bound on the worst-case coupling capacitance, which gives the designers the insight of varying performance evaluations for different MPL decompositions. Moreover, the obtained upper bound can be further used as an upper bound in power/timing analysis. Our contributions can be summarized as follows:

- We mathematically proved the worst-case scenarios of coupling capacitance incurred by mask misalignment in MPL decompositions.

- We proposed a graph model that guarantees to compute the tight upper

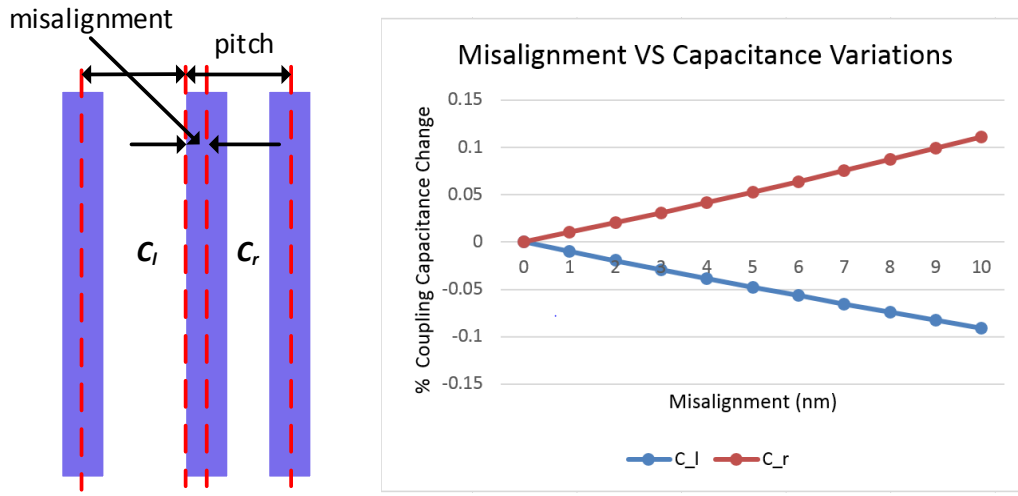


Figure 5.1: Variations of coupling capacitance due to mask misalignment. There are three features in the layout, with the normal spacing (pitch) of 100 nm. The misalignment is assumed to be 10% in the worst-case, which is 10 nm. The percentage of the variations in coupling capacitance is shown in the right figure, where C_l and C_r refer to the capacitance of the left two features and right two features respectively. Only lateral capacitance is shown here [79]. The capacitance C is calculated as $C = \frac{\epsilon S}{d}$, where ϵ is the permittivity of the intermediate material and S is the area of the two parallel metallic plates.

bound on the worst-case coupling capacitance of any MPL decompositions for a given layout.

The rest of the chapter is organized as follows. Some preliminaries of the problem are discussed in Section 5.2. The problem description is presented in Section 5.3. Our algorithm is presented in Section 5.4, followed by the experimental results in Section 5.5. Finally, we give the conclusions in Section 5.6.

5.2 Preliminaries

Some preliminaries of mask misalignment are discussed here. Existing approaches considering the effects of mask misalignment are also reviewed.

5.2.1 Mask Misalignment in MPL Decomposition

In MPL decompositions, the features are decomposed into different masks and go through separate litho exposures. However, due to process variations, the alignments of different masks are never perfect. As shown in Fig. 5.1, the spacing between features on different masks could be increased/decreased due to misalignment between different masks. Mask misalignment leads to significant variations of the coupling capacitance [78], which further leads to power and timing degenerations [75, 76]. Two popular approaches comprehending mask misalignment on power/timing are discussed in the following.

5.2.2 Min/Max Extraction

To capture the coupling capacitance variations due to mask misalignment, min/max capacitance can be extracted as follows. For each pair of parallel lines, the coupling capacitance is extracted as a triplet with the form "min:nom:max", where min corresponds the coupling capacitance with minimum mask misalignment, nom corresponds to the coupling capacitance with no mask misalignment, and max corresponds to the coupling capacitance with maximum mask misalignment.

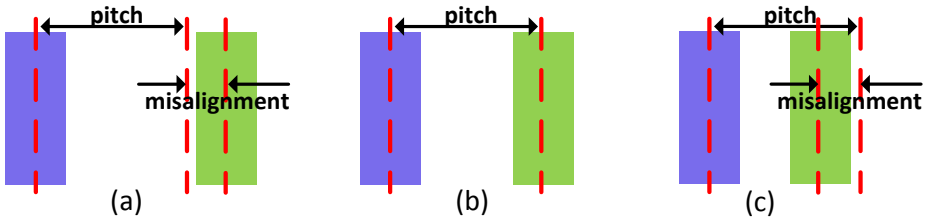


Figure 5.2: Min/Max coupling capacitance extraction considering mask misalignment. (a) Minimum coupling capacitance. (b) Normal coupling capacitance. (c) Maximum coupling capacitance.

Figure 5.2 shows an example of extracting coupling capacitance under different misalignment scenarios. Each entry in the triplet "min:nom:max" is computed based on different misalignment values.

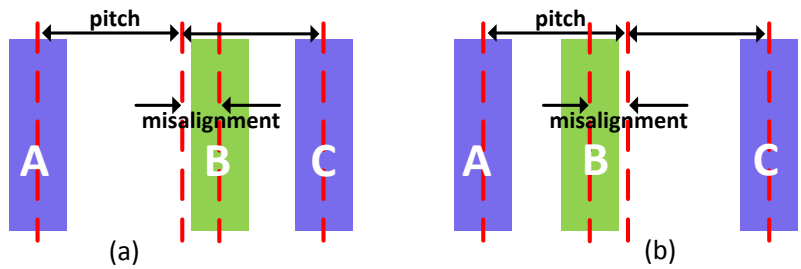


Figure 5.3: Pos/Neg coupling capacitance extraction considering mask misalignment. (a) Coupling capacitance with positive mask misalignment. (b) Coupling capacitance with negative mask misalignment.

5.2.3 Positive/Negative Extraction

The min/max extraction of coupling capacitance could be overpessimistic. One example is shown in Fig. 5.3, where there are three features in the layout. It is obvious that the min/max coupling capacitance between AB and BC cannot happen at the same time. Thus, the authors in [76] proposed a new model to capture coupling capacitance variations for DPL. As the techniques target on DPL, mask misalignment only comes from the second mask in DPL (assuming the second mask is aligned relative to the first mask). For every net, its coupling capacitance is extracted as a triplet named "pos:nom:neg". "pos" refers to the capacitance with positive misalignment, "nom" refers to the capacitance with no misalignment, while "neg" refers to capacitance with negative misalignment respectively.

The major difference of the "pos:nom:neg" and "min:nom:max" representation is that if you group the "pos", "nom", and "neg" values together, the overall capacitance is not over pessimistic, and the actual capacitance is physically feasible in silicon. Assume there are three nets in the layout, A, B and C respectively, as shown in Fig. 5.3. The representation of coupling capacitance for this layout looks like the following. For features A and B, the triplet is $C_{pos}^{AB} : C_{nom}^{AB} : C_{neg}^{AB}$. For B and C, the triplet is $C_{pos}^{BC} : C_{nom}^{BC} : C_{neg}^{BC}$. Here, C_{pos}^{AB} and C_{pos}^{BC} correspond to the coupling capacitance with positive mask misalignment, C_{nom}^{AB} and C_{nom}^{BC} for zero misalignment, and C_{neg}^{AB} and C_{neg}^{BC} for negative mask misalignment respectively.

However, in the "min:nom:max" form of representations, the same parameters look like this: $C_{pos}^{AB} : C_{nom}^{AB} : C_{neg}^{AB}$ and $C_{neg}^{BC} : C_{nom}^{BC} : C_{pos}^{BC}$. Clearly, the capacitance C_{pos}^{AB} and C_{neg}^{BC} cannot appear at the same time, as they require

the mask misalignment to be both positive and negative, which is physically infeasible. However, this approach only applies for DPL, and is applied for a per-net analysis in [76].

In this chapter, we studied the worst-case coupling capacitance scenarios for MPL decompositions. Compared with the "pos:neg" approach which is usually optimistic and "min:max" which is always pessimistic, our results are tight and physically achievable.

5.3 Problem Description

In this problem, we are given the input layout, the minimum coloring distance d_{min}, a capacitance model to extract the coupling capacitance between two parallel lines, and a constant k, which denotes how many masks are available to decompose the layout. In the following, we discussed how to extract the coupling capacitance in the layout, which will be used in evaluating the worst-case coupling capacitance for a given layout.

We followed the same assumptions as in [76], where lateral coupling capacitances are extracted considering mask misalignment. The rest of the coupling capacitances are extracted corresponding to zero misalignment. Namely, non-lateral capacitance and ground capacitance are always extracted as single values corresponding to the zero misalignment case. The parallel plate model is used to extract the lateral capacitance, which has shown to be accurate and correlated well to the real capacitance [80]. For two parallel metallic plates of area S and spacing d, the capacitance is calculated by $C = \frac{\epsilon S}{d}$, where ϵ is the permittivity of the intermediate material.

In this chapter, we focused on standard-cell-based designs, which is one of the most popular design styles in industry and has been studied in several previous works [48, 62, 81, 82]. The problem can be formally defined as follows.

Performance Evaluation Considering Mask Misalignment in MPL Decomposition: Given a cell-based row structure layout, a minimum coloring distance d_{min}, the number of available masks k, the maximum mask misalignment values s of each mask, our objective is to compute a tight upper bound on the worst-case coupling capacitance for the given layout.

5.4 Algorithm

A high-level description of our approach is as follows. The characteristics of coupling capacitance variations induced by mask misalignment are first analyzed. Based on the analysis, we proved that the worst-case coupling capacitance only happens at the boundaries. Based on this observation, our algorithm works as follows. We build a solution graph similar to the one proposed in [48]. Weights of the edges are assigned as the worst-case coupling capacitance between the two connected decompositions. It is shown in [48] that any path in the solution graph corresponds to a legal TPL decomposition, and any legal decomposition corresponds to a path in the solution graph. Thus, we can run a longest path algorithm on the solution graph, which guarantees to compute a tight upper bound for the worst-case coupling capacitance. The details are discussed as follows.

5.4.1 Coupling Capacitance Due to Mask Misalignment

Coupling capacitance variations occur when there is mask misalignment in either the X or Y direction. Mask misalignment in X and Y directions is assumed to be independent here. For standard-cell-based layout, there are power tracks isolating different rows [48], where the power tracks are preferred to be assigned to the same mask. When the misalignment in the X direction is small, i.e. 10% variation which is 10 nm for 100 nm pitch, we assume that it only affects coupling capacitance of two parallel lines with no intermediate features within the two lines. The following discussions are based on mask misalignment in X directions. The same principle applies for misalignment in Y direction as well.

Denote all the vertical edges in the layout as $E = \{e_1, e_2, ..., e_n\}$, where the edges are sorted in non-decreasing X coordinates and n is the total number of vertical edges in the layout. Define an indicator variable δ_{ij} as follows:

$$\delta_{ij} = \begin{cases} 1 & \text{If } e_i \text{ and } e_j \text{ are adjacent} \\ 0 & \text{Otherwise} \end{cases} \tag{5.1}$$

There are k masks available, and assume all masks are aligned relative to mask 0. Thus, misalignment for different masks can be denoted as $\{m_0, m_1,$

..., $m_{k-1}\}$, where m_i refers to the mask misalignment for mask i, and m_0 will be always 0. Denote L_{ij} as the overlapping length between edges e_i and e_j, d_{ij} as the normal spacing between e_i and e_j. Define another indicator variable $I^s_{e_i}$ as follows:

$$I^s_{e_i} = \begin{cases} 1 & \text{If } e_i \text{ is on mask } s \\ 0 & \text{Otherwise} \end{cases} \tag{5.2}$$

The total coupling capacitance can be represented as

$$C = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{\sigma \delta_{ij} L_{ij}}{d_{ij} - \sum_{h=0}^{k-1} m_h I^h_{e_i} + \sum_{h=0}^{k-1} m_h I^h_{e_j}} \tag{5.3}$$

where σL_{ij} equals to ϵS in the parallel plate model. Note that σ is a constant which is unrelated to any of the mask misalignment variable m_i. Now we have the following lemma.

Lemma 5. *C is a convex function with respect to m_i.*

Proof. Denote $f(i,j)$ as follows:

$$f(i,j) = \frac{\sigma \delta_{ij} L_{ij}}{d_{ij} - \sum_{h=0}^{k-1} m_h I^h_{e_i} + \sum_{h=0}^{k-1} m_h I^h_{e_j}} \tag{5.4}$$

then we have

$$C = \sum_{i=1}^{n} \sum_{j=i+1}^{n} f(i,j) \tag{5.5}$$

Thus, proving $f(i,j)$ is convex is sufficient since C is a linear combination of $f(i,j)$.

Note that $f(i,j)$ has $k-1$ variables, as m_0 will always be 0. Therefore, the Hessian matrix H of $f(i,j)$ has dimensions of $(k-1) \times (k-1)$. Computing the Hessian matrix of $f(i,j)$ yields a matrix with $H(s,t)$ as follows:

$$H(s,t) = 2\sigma \delta_{ij} L_{ij} \frac{(I^s_{e_j} - I^s_{e_i})(I^t_{e_j} - I^t_{e_i})}{(d_{ij} - \sum_{h=0}^{k-1} m_h I^h_{e_i} + \sum_{h=0}^{k-1} m_h I^h_{e_j})^3} \tag{5.6}$$

Denote the matrix M as follows:

$$\mathbf{M} = \begin{pmatrix} (I^1_{e_j} - I^1_{e_i})^2 & (I^1_{e_j} - I^1_{e_i})(I^2_{e_j} - I^2_{e_i}) & \dots \\ (I^2_{e_j} - I^2_{e_i})(I^1_{e_j} - I^1_{e_i}) & (I^2_{e_j} - I^2_{e_i})^2 & \dots \\ \vdots & \vdots & \ddots \\ (I^{k-1}_{e_j} - I^{k-1}_{e_i})(I^1_{e_j} - I^1_{e_i}) & \dots & \dots \end{pmatrix} \tag{5.7}$$

We have $H = \mu M$, where

$$\mu = \frac{2\sigma\delta_{ij}L_{ij}}{(d_{ij} - \sum_{h=0}^{k-1} m_h I^h_{e_i} + \sum_{h=0}^{k-1} m_h I^h_{e_j})^3} \tag{5.8}$$

$\mu \geq 0$ and is a constant given any e_i and e_j.

Next, for any $k-1$ dimensional vector of $Z = \{z_1, z_2, ..., z_{k-1}\}$, we have the following scenarios when computing $Z^T MZ$:

- If e_i and e_j are on the same mask, we have $I^h_{e_i} = I^h_{e_j}$ for any h. Thus, $Z^T MZ = 0$.
- If e_i is on mask 0, and e_j is on mask h where $h \neq 0$, we have $Z^T MZ = z_h^2 \geq 0$. Similarly, we have $Z^T MZ = z_h^2 \geq 0$ when e_i is on mask h, and e_j is on mask 0.
- If e_i is on mask s, and e_j is on mask h where $h \neq 0$, $s \neq 0$, and $s \neq h$, we have $Z^T MZ = (z_h - z_s)^2 \geq 0$.

Clearly, $Z^T MZ \geq 0$ for any vector Z, which immediately indicates that $Z^T HZ \geq 0$ for any vector Z. Thus, $f(i, j)$ is convex, which means C is also convex. The proof is complete. □

The same argument holds when considering mask misalignment in Y direction. The convexity of function C brings great convenience when computing the worst-case coupling capacitance for a given decomposition, as the worst-case only occurs at the boundaries. Namely, when computing the worst-case coupling capacitance, we only care about boundaries where there are worst-case X and Y misalignments. This key property enables us to enumerate all possible worst-case scenarios and build up a solution graph which guarantees to compute a tight upper bound of the worst-case coupling capacitance. Details are introduced in the following section.

5.4.2 Some Terminologies

In this section, we will introduce an algorithm that guarantees a tight upper bound on the worst-case coupling capacitance for a given layout. All power tracks are assumed on the same mask. The approach of solving one row is first presented, followed by combining solutions for different rows. The details are introduced in the following.

Some terminologies used in the algorithm are first introduced. Like many of the previous works [25, 27, 48, 53, 56], a conflict graph $CG = (V, E)$ is defined for the input layout, where each vertex corresponds to a polygon in the layout, and there is an edge connecting two vertices if their distance is less than d_{min}. Besides the conflict graph, an adjacency graph $AG = (V, E^{AG})$ is also defined. AG has the same vertices as that in CG, but with more edges. There is an edge connecting two vertices in AG if there is an edge connecting them in CG, or any of their parallel edges form a capacitor.

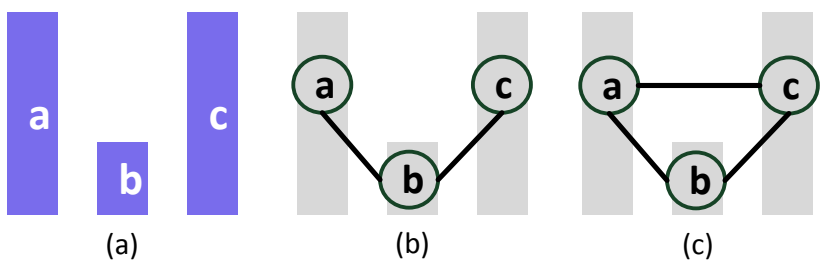


Figure 5.4: (a) Input layout. (b) Conflict graph. (c) Adjacency graph.

Figure 5.4 illustrates the concept of CG and AG. There are three features in the layout, where the distance of ab and bc is less than d_{min} respectively. The CG is shown in Fig. 5.4 (b), while the AG is shown in Fig. 5.4 (c). Note that in CG, there is no edge connecting a and c as their distance is larger than d_{min}. However, a and c are connected in AG, as their edges form a parallel capacitor.

Next, *polygon dummy extension* is performed on the layout, where the right boundary of a feature is virtually extended to the left boundary of its rightmost conflicting feature. Different from the previous work [48] where polygon dummy extension is based on CG, our polygon dummy extension is based on AG. The difference between the two methods is illustrated in Fig. 5.5. For the layout in Fig. 5.5 (a), the layout after polygon dummy extension is the same as the original one, as shown in Fig. 5.5 (b). Using our

approach, the layout is shown in Fig. 5.5 (c), where the right boundary of feature a is virtually extended to the left boundary of feature c.

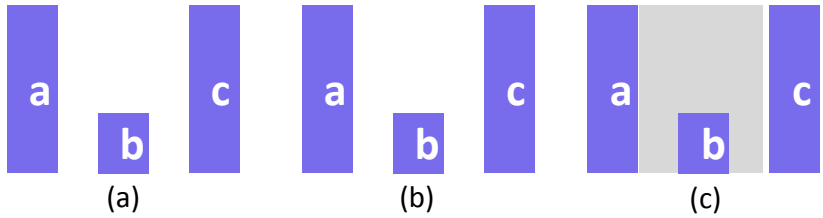


Figure 5.5: (a) Input layout. (b) Polygon dummy extension in [48]. (c) Polygon dummy extension for our approach.

We reuse the definitions of cutting line and cutting line set from the previous work [48]. The concepts are illustrated as follows. A cutting line refers to a vertical line that is aligned with the left boundary of a feature in the layout. A cutting line set refers to the set of polygons that intersect with the same cutting line. Different from [48], we will recursively merge adjacent cutting line sets if one cutting line set is a subset of its adjacent ones. By merging these redundant cutting line sets, the size of the solution graph is reduced while the completeness of the graph is not affected.

5.4.3 Graph Model for Worst-Coupling Capacitance Computation

With all the cutting line sets available, a solution graph can be built as follows. For each cutting line set, enumerate all its possible solutions. For each solution, create up to 4^{k-1} vertices[1] in the graph. Note that all these vertices have exactly the same coloring solution, but with different mask misalignment values. Currently, triple patterning lithography ($k = 3$) is one of the most promising options for 14/10 nm technology node. For 7 nm technology node, quadruple patterning ($k = 4$) could be used. But it is unlikely that people goes to $k > 4$ due to cost and some technical issues. Thus, the number of vertices per cutting line set is limited.

After that, compatible vertices are connected for adjacent cutting line sets. Compatible vertices mean that no two features that are connected in CG are

[1]For x direction, there are up to 2^{k-1} combinations. Similarly, there are up to 2^{k-1} combinations in y direction. Thus, the number is 4^{k-1}.

assigned to the same mask, and the worst-case mask misalignment for the same mask is identical. Weight are assigned to the edges in the graph, where the weight is the worst-case coupling capacitance of the two connected decompositions. Intuitively, the weight means that how much extra coupling capacitance is needed to transit from one decomposition to the other one. Since the worst-case mask misalignments are already known in each vertex, the worst-case coupling capacitance can be easily computed. Finally, a virtual source and virtual sink are constructed. The source connects to all vertices of the first cutting line set, while the sink connects to all vertices of the last cutting line set.

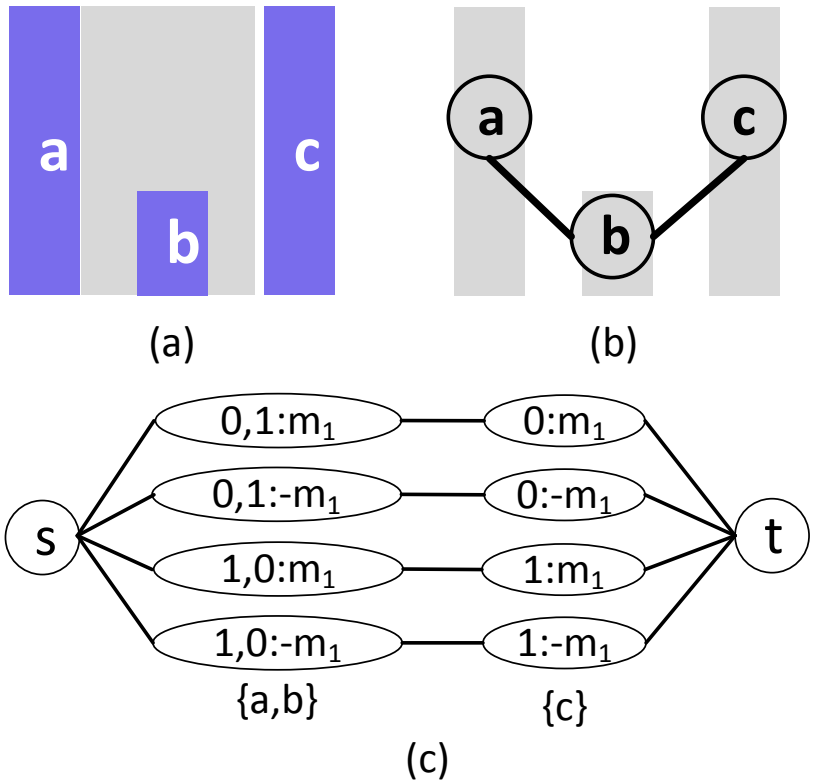


Figure 5.6: (a) Input layout after polygon dummy extension. (b) Conflict graph (CG). (c) DPL solution graph. m_i in this example means mask misalignment in X direction for mask i. Mask misalignment in Y direction is not considered, since it does not affect the effective coupling capacitance. The tuple $\{1 : m_1\}$ for cutting line set $\{a\}$ means that feature a is on mask 1, and the mask misalignment for mask 1 is m_1. Note that m_0 is always 0, which is not shown in the picture. Same principle applies for other cutting line sets. Weights of the edges are not shown here for simplicity.

Figure 5.6 shows a simple example of how to construct the DPL ($k = 2$) solution graph for a given layout. For a given layout, its CG and AG are first constructed. Polygon dummy extension is then performed. After that, all cutting lines and cutting line sets are computed. For each cutting line sets, all solutions are enumerated. Compatible solutions of adjacent cutting lines are connected. Finally, a longest path algorithm is used to get the tight upper bound on the worst-case coupling capacitance for all decompositions. Although longest path algorithm on general graph is NP-hard, it is solvable in polynomial time for directed acyclic graph.

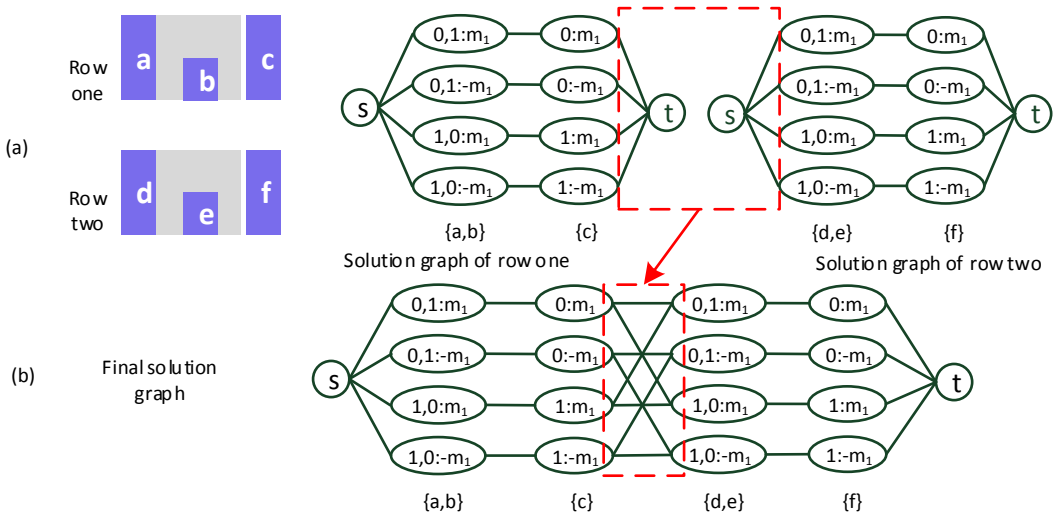


Figure 5.7: (a) Input layout after polygon dummy extension, and solution graph for each row. (b) Final solution graph of the two rows. Power tracks and weights for edges are not shown here for simplicity.

We have the following lemma for the solution graph. The proofs are omitted due to page limits.

Lemma 6. *Every path in the solution graph is a legal decomposition with physically feasible worst-case mask misalignments, and vice versa.*

5.4.4 Final Decomposition

For each row, all its decompositions are incorporated in the solution graph. Computing a solution for one row is straightforward. Longest path algorithm

Table 5.1: Comparisons with Previous Works

Test Case	# Rows	# Polygons	Ours Tight Bound	Pos/Neg Bound	Min/Max Bound	Runtime (s)
C1	8	284	1	0.62	1.40	0.82
C2	13	716	1	0.62	1.49	1.21
C3	16	999	1	0.63	1.34	1.68
C4	22	1805	1	0.65	1.45	3.23
C5	36	4878	1	0.64	1.38	9.44
C6	72	19110	1	0.64	1.38	46.16
C7	108	42295	1	0.64	1.38	118.44
C8	143	74630	1	0.64	1.37	239.50
C9	179	116650	1	0.64	1.37	421.23
C10	215	167003	1	0.64	1.37	667.48
Avg.	81.2	42837	1	0.63	1.39	150.92

can be used to compute the decomposition with a tight upper bound coupling capacitance.

Some modifications to the solution graph are needed to compute the upper bound of the whole layout, where multiple rows usually exist. The solution graph of different rows are sequentially connected, forming the final solution graph for the whole layout. To connect two different solution graphs S_A and S_B, we add compatible edges between the last cutting line set vertices in S_A and the first cutting line set vertices in S_B. The weight of the edges are assigned to be the coupling capacitance of the corresponding vertices in S_B. As features in different rows are isolated by the power tracks, two vertices will be compatible with each other as long as their mask misalignments are identical.

After merging the solution graphs in different rows into one graph, the worst-case capacitance can be computed by running a longest path algorithm on the merged graph. The correctness of the algorithm is guaranteed as follows. First consider two rows and their solution graphs S_A and S_B. From Lemma 6, we know that for each row, all legal solutions are incorporated in its solution graph. By adding all-pair compatible edges between S_A and S_B, any solution in one row is guaranteed to be connected to another solution in the other row, as long as they have the same mask misalignments. Therefore, the merged solution graph explores all solution space of the two rows. The same argument applies for multiple rows as well. Thus, the longest path

algorithm gives us the decomposition with a tight upper bound on worst-case coupling capacitance. The "tight" here means that the coupling capacitance is physically achievable, which is the key difference with "min:nom:max" notations.

Figure 5.7 shows an example of combining the solution graphs of two different rows. The final solution graph for the whole layout is shown in Fig. 5.7 (b). As we can see that the two solution graphs are merged based on the last cutting line set of the first row, and the first cutting line set of the last row. Any two nodes with exactly the same mask misalignments are connected together, forming the final solution graph of the given layout.

For performance purposes, enumerating all possible mask misalignment while constructing the solution graph is "over-killed", as the solution graph follows the same pattern under different mask misalignment scenarios. The solution graph can be thought as being split into $2^{2(k-1)}$ portions, where k is the number of available masks. Each portion of the graph corresponds to one corner case of the mask misalignment scenarios. For each portion of the graph, the structures are identical, and the only difference is the weight of the edges.

To speed up the construction of the solution graph, weight of the edges is assigned after constructing an unweighted solution graph. As analyzed above, for different mask misalignment scenarios, the structure of the graph remains the same. After constructing an unweighted solution graph, we loop through all corner cases of mask misalignment, and each time reassign the weight of the edges in the graph. For each corner case, the tight upper bound is identified using longest path algorithm. Final upper bound is chosen by comparing the upper bounds in all possible corner cases.

5.5 Experimental Results

The algorithm is implemented in C++ and run on a Linux server with 4GB RAM and a 3.00 GHZ CPU. All benchmarks are generated using NanGate FreePDK45 Generic Open Cell Library [28], which is available online. The 10 benchmarks are generated by randomly aligning the cells into different rows. They are generated in increasing size to better reflect the scalability of our approach. We evaluate our approach based on triple patterning de-

composition where $k = 3$ and $d_{min} = 100$ nm. Maximum mask misalignment is set to be 10 nm in X directions and 10 nm in Y directions for all masks except for mask 0. Power tracks are assigned to mask 0. Wires on the M1 layer are used for all experiments.

Detailed results are shown in Table 5.1. The first column of the table shows the name of the benchmarks. The number of rows and the number polygons are shown in the second and third columns. The upper bound on worst-case capacitance is shown in column four, with the worst-case capacitance for "pos:nom:neg" shown in column five. The bound computed by "min:nom:max" approach is shown in column six. The last column shows the runtime of the algorithm.

As the original "pos:nom:neg" approach is only applied for DPL, we extend it to TPL by performing random walks on the solution graph. In particular, we approximate its value by randomly picking up 100 decompositions in the solution graph, and averaging their coupling capacitances. Therefore, column five shows how good the bounds are when naively extending "pos:nom:neg" approach. Note that values of column four, five and six are obtained by subtracting the nominal capacitance[2] and then normalizing the values based on tight bound on the worst-case capacitance.

We can clearly see that "min:nom:max" tends to overestimate the coupling capacitance variations, while "pos:nom:neg" tends to underestimate the effects due to mask misalignments. When decompositions are not known beforehand, optimistic estimations on coupling capacitance could lead to timing violations while pessimistic estimations imposes unnecessary constraints during the design stage. As indicated in the table, "min:nom:max" approach could overestimates the capacitance variations by as much as 39% on average, while "pos:nom:neg" approach could underestimate the capacitance variations by as much as 37% on average. For our approach, the bound computed is tight, as the way we compute the maximum coupling capacitance guarantees that it is physically achievable. Not surprisingly, the runtime in the last column indicates that the runtime increases as the size of the benchmark increases. However, we can see that the runtime roughly has a linear correlation with the size of the benchmark. This clearly shows the effectiveness of our approach.

[2]Capacitance with zero mask misalignment.

5.6 Conclusions

In this chapter, we studied capacitance variations in MPL decompositions considering mask misalignment, which is prominent and inevitable in advanced technology nodes. We mathematically proved that worst-case coupling capacitance only occurs at the boundaries of different mask misalignment, and proposed an algorithm that guarantees to compute a tight upper bound on the worst-case coupling capacitance. Compared with the "pos:nom:neg" approach and the "min:nom:max" approach, experimental results show that the first approach tends to underestimate the capacitance variations by as much as 37% while the latter approach tends to overestimate the capacitance variation by as much as 39% on average. Our approach guarantees to find the tight capacitance upper bound for any decompositions for a given layout. Our approach is expected to help engineers better understand the qualities of different decompositions, and brings convenience for advanced technology nodes.

CHAPTER 6

FUTURE DIRECTIONS ON TRIPLE PATTERNING DECOMPOSITION

In this chapter, we will discuss some of the possible future directions for triple patterning decomposition, and show some of our preliminary results.

6.1 Pattern-Based Triple Patterning Decomposition

As illustrated in previous chapters, extensive research efforts have been devoted to TPL [25, 27, 48, 54]. An ILP-based algorithm is proposed by Bei Yu et al. [25], which is not capable of handling larger layout due to the exponential time complexity of the ILP approach. They also proposed a semidefinite programming technique to reduce the runtime. However, the semidefinite programming technique is trading off the runtime with the optimality of the algorithm. The decomposition results obtained are no longer guaranteed to be optimal. A graph-based approach is proposed by Fang et al. [27], which cannot guarantee to find a solution if one exists. Moreover, the approach typically generated more stitches compared with the work by Bei Yu et al. Stitches increase manufacturing cost and can potentially lead to function errors of the chip due to the line end errors. The high number of stitches makes the algorithm difficult to be employed in practice. For our previous algorithm [48], it runs in polynomial time and guarantees to find a solution if one exists. When there are stitches, the algorithm guarantees to compute an optimal solution with minimum number of stitches.

Given the importance of TPL, a fast, robust, and accurate evaluator is needed for the designers to evaluate the printability of a layout. The evaluator gives some insights to the designers to modify or redesign the circuit based on the printability of the layout. To qualify for the evaluator, the algorithm needs to be fast, accurate, and guarantees to find a solution if one exists. None of the algorithm by Bei Yu et al. and Fang et al. satisfy all the

above requirements. Due to the low time complexity and optimality of our algorithm, our previous approach well fits into these requirements, and can be potentially used as an evaluator for chip designers.

For all the previous works, a single conflicting distance d_{min} is used to decide whether two features can be assigned to the same mask. However, in practice, the printability of different patterns can never be clearly separated by a constant distance. On the contrary, they should be involved with a comprehensive analysis based on different distances, different geometry patterns, and different process-dependent parameters. A local pattern aware cost model is needed to capture different printability of various patterns.

None of the previous works capture the pattern aware TPL decomposition problem. To be able to used by the designer to evaluate their designs, we need to test the extendibility of our previous optimal TPL algorithm. For our previous optimal TPL algorithm, it is inherently a pattern aware formulation. When constructing the solution graph, proper cost can be assigned to the edges to capture local pattern aware costs. By doing this, we can construct a weighted directed solution graph, and then utilize a shortest path algorithm to compute the optimal solution. Due to the efficiency and easy extendibility of the approach, it can be used as an evaluation tool to evaluate the decomposability of any customer designed layout, given a user-specified local pattern aware cost model.

In this section, we will discuss cost-driven TPL decompositions, and show some experimental results how our approach helps in reducing printing variations.

6.1.1 Criteria Guiding TPL Decomposition

TPL decomposition can be guided using a distance-driven model and local pattern aware cost-driven model respectively. Many of the previous works are based on a single minimum coloring distance d_{min}, which are not enough to capture different pattern scenarios. For our previous optimal algorithm, the distance-driven scheme is already incorporated and well addressed. In the following, we will use a local pattern aware cost-driven framework to test the effectiveness and extendibility of our previous algorithm. The distance-driven and cost-driven triple patterning decompositions are detailed in this

section.

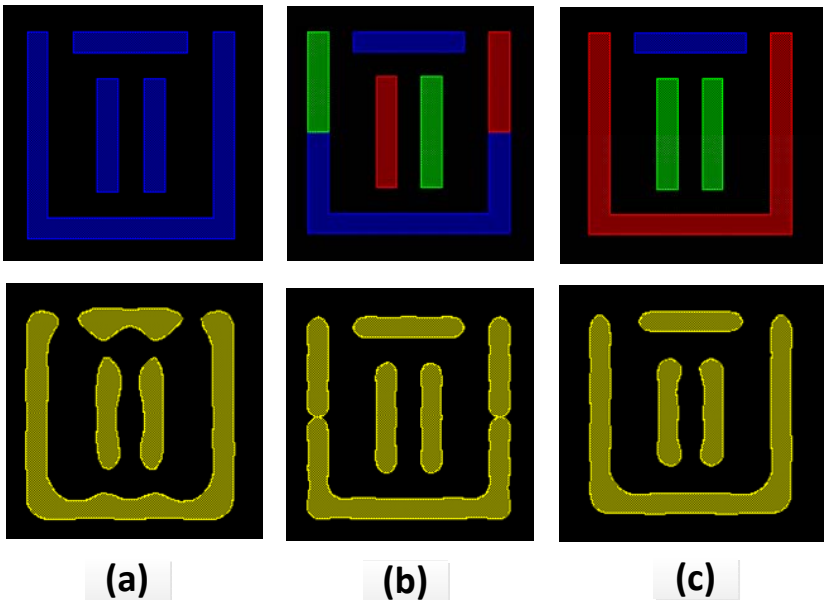


Figure 6.1: A decomposition comparison for the M1 layer pattern with 40 nm width, and their lithography simulation with best focus and 0 misalignment. (a) Single mask decomposition (higher image) and its printed pattern on the wafer (lower image). (b) Tradition TPL decomposition and its printed pattern on the wafer. (c) Local pattern aware TPL decomposition and its printed pattern on the wafer. Note that different colors here denote different patterns.

Constant Distance Criteria

Using a single distance d_{min} to differentiate the printability of different patterns can effectively reduce the complexity of the multiple patterning problems. Once d_{min} is known, feature within distance d_{min} cannot be assigned to the same mask, while features with distance larger than d_{min} can be freely assigned to any masks.

The previous algorithms by Bei [25], Fang [27], and ours [48] are all based on a single distance d_{min}. A comprehensive comparisons between these three algorithms are already discussed in Chapter 1. Due to the high time complexity of the ILP-based algorithm [25], it is not practical to use it as an evaluator to check the printability of large layout. For the graph-based algorithm [27], it cannot guarantee to find a solution if one exists, which also

limits its usage in practice to evaluate the printability of a layout. Our approach guarantees to find a solution if one exists, and can compute the TPL solution with the minimum number of stitches. It runs in polynomial time, and is capable of handling very large layout. Thus, it can be used an effective evaluator to characterize the printability of any standard-cell-based layout.

Since the constant distance rule is not seeing the difference between different patterns, it could lead to some degenerated results. Naively adopting the minimum distance rule could possibly lead to stitches, even when the pattern itself is actually decomposable. Figure 6.1 shows a simple example to demonstrate the limitation of the single-constant-distant criteria, where four features conflict with each other when using a single minimum distance. Figure 6.1(a) shows the result of printing the four features in one mask, and there are serious line width degenerations. Figure 6.1(b) shows the decomposition result and its simulation with tradition TPL algorithm. Here, two stitches are needed to resolve the conflicts, which could have severe reliability issues [16, 17, 18]. However, with a pattern aware distance criteria, we can achieve an acceptable decomposition result as shown in Fig. 6.1(c).

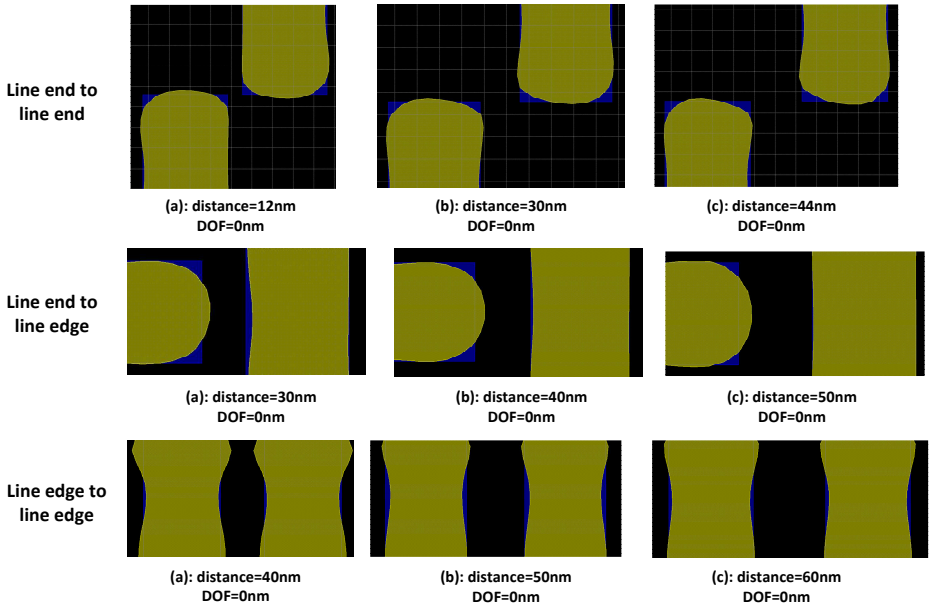


Figure 6.2: Printed patterns for different geometry features with different spacing values and different depth of focus (DOF) values. These results are obtained using Calibre WORKBench simulations.

Cost Metric for Printed Patterns

In reality, a single distance is not enough to characterize the printability of different patterns. The no-print distance and best-print distance can never be clearly separated by a constant distance value. Indeed, this decomposition criteria should involve a complex analysis, which is a function of lithography printing parameters, pattern types and geometry distances.

We have performed some preliminary simulations using Calibre WORKBench on some patterns. The experimental results clearly indicate that different patterns have very much different tolerances in printability even when they are the same distance apart. We show some preliminary data based on the simulations with different patterns. The results are shown in Fig. 6.2. For the line end to line end local pattern, very well printing quality is observed even when the distance is 12 nm, as shown in the upper figure of Fig. 6.2. When the distance increase, no significant improved printing quality is observed. For the line edge to line edge local pattern, the printing quality is almost unacceptable when the distance is 40 nm. It becomes better when the distance between the features is larger. This set of simulation data clearly shows that the best-print distance and no-print distance of different patterns can be very different.

To best capture the cost related to different local patterns, a through and complete analysis of various process related parameters and a complete set of critical local patterns are needed, which is beyond the scope of this thesis. We are focusing on testing the robustness and extendibility of our algorithm in handling local pattern aware TPL decompositions. We adopt a simple cost-aware model based on Calibre WORKBench simulations to test the robustness of our algorithm. Three of the patterns used are shown in Fig. 6.2. Note that in practice, a more through and precise model is expected from the user. The optimality of our algorithm does not depend on a specific model. Given any user-defined local pattern aware cost model, the algorithm is able to compute optimal solutions with the minimum cost with respect to that model. However, the accuracy of the decompositions do largely depend the accuracy of the model provided.

The simple cost aware model used to evaluate our TPL algorithm is based on the local patterns. Generally, the further apart of the patterns, the better printing quality they will be, and therefore the lower cost there will be. The

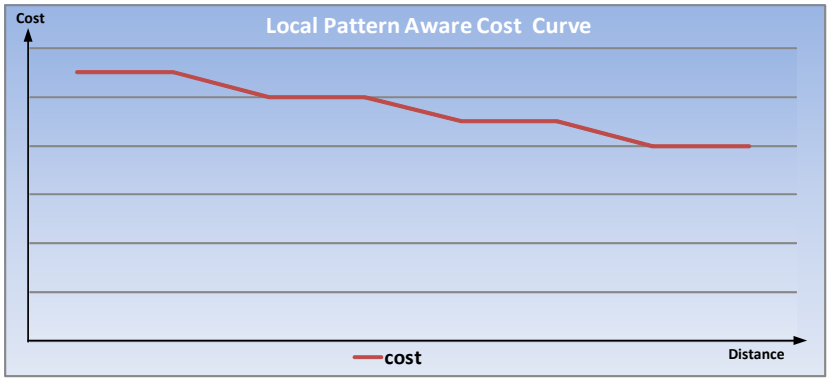


Figure 6.3: A typical trend for the local pattern aware cost curve.

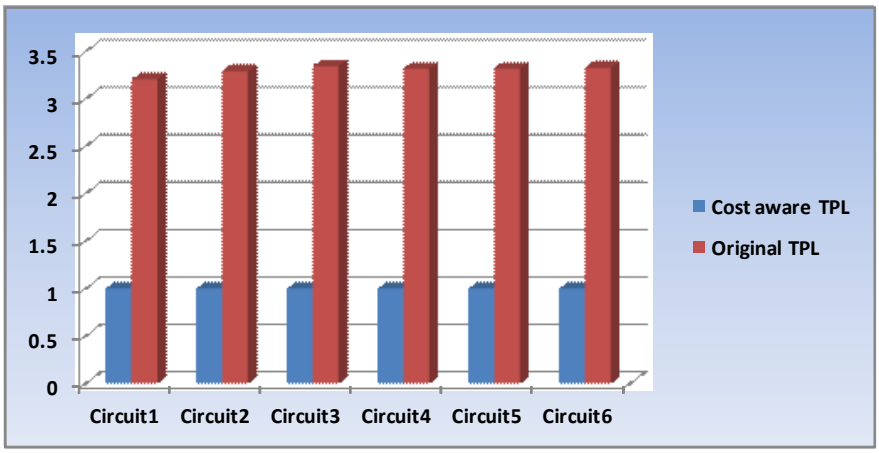


Figure 6.4: Figure showing the cost reduction compared with the previous optimal TPL algorithm. The results of cost aware approach are scaled as 1. An average of 3.3x reduction is achieved.

general trend for the local pattern aware cost model is shown in Fig. 6.3. Note that for every pattern, we need a cost curve that are only applicable to that particular pattern. Ideally, the cost curve should involve a complete analysis of process related parameters and lithography simulations. The curve obtained characterizes the printing quality of the pattern under different distance values.

Utilizing the local pattern aware cost model in our TPL algorithm requires some modifications of the problem formulation. Previously, we only have one type of edge, conflicting edge, to indicate the conflicting relations between different features. All of the conflicting edges together formulate the constraint graph. To accommodate the local pattern aware cost model, we add

one more type of edge correspond to each type of pattern in the constraint graph. Therefore, two features now are connected by multiple types of edges, with each edge indicating the type of patterns current feature is involved.

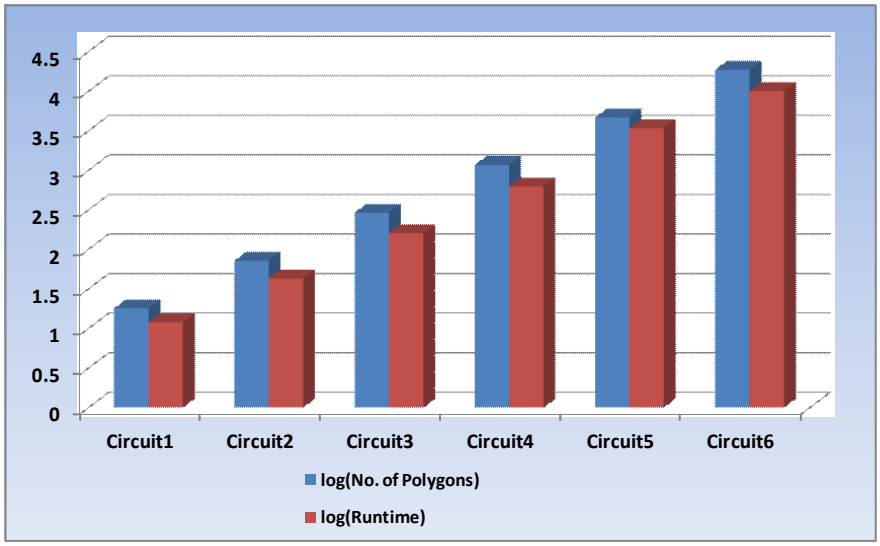


Figure 6.5: Comparisons of the runtime/number of polygons ration. The ratios are almost the same with minor variations, which indicates that it is an polynomial time algorithm.

6.1.2 Pattern-Based TPL Results

Based on our simple local pattern aware cost model, we run the optimal TPL algorithm to compare the cost-aware TPL results with the results without using the cost model. The algorithm is implemented in C++ and run on a Linux machine with 4GB RAM and a 2.8GHZ CPU. All the benchmarks are generated using NanGate FreePDK45 Generic Open Cell Library [28]. The wires on metal 1 are used in the experiment to validate our algorithm.

Comparisons with the Results without Cost-Driven Optimization

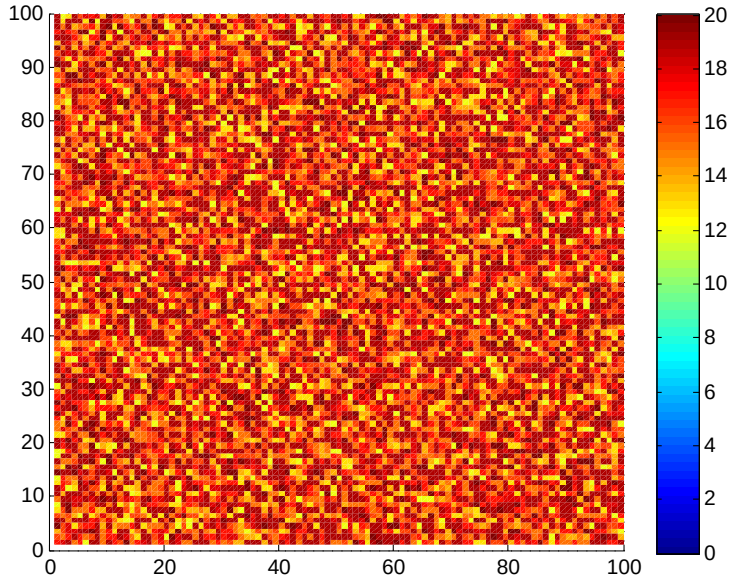
We compare our work with the previous optimal TPL algorithm [48]. We maximize the minimum distance used in their algorithm such that their approach is able to solve all the benchmarks without introducing stitches. For each benchmark, we arbitrarily choose an optimal solution produced in the

previous work [48], and compare its cost with the optimal solution generated using our proposed algorithm. The detailed results are shown in Fig. 6.4 and Fig. 6.5 respectively. Figure 6.4 shows that the cost aware algorithm always achieves superior results compared with the previous TPL algorithm which doesn't consider different cost for different patterns. Figure 6.5 shows that the runtime/polygons ratios changes with small fluctuations, which indicates that the algorithm is a polynomial time algorithm. For benchmarks with over 18 million polygons, the runtime is within three hours. Compared with the results in previous optimal TPL algorithm [48], we can reduce the cost by as much as 3.3x on average.

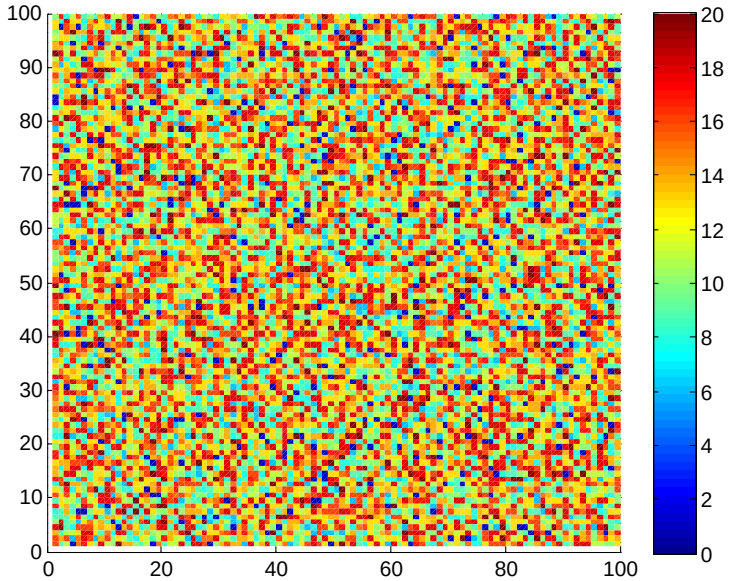
6.2 Color Balancing for Triple Patterning Lithography

There are several works on triple patterning lithography in the literature [25, 26, 27, 48, 49, 50, 51, 52, 53, 54]. Bei Yu et al. [25] proved that the general TPL decomposition problem is NP-hard, and proposed an ILP-based approach for general TPL decompositions. A semidefinite programming approach is also proposed to further improve the runtime. S. Fang et al. [27] proposed a graph-based heuristic to solve general TPL problems and achieves good results while greatly reducing the runtime. Tian et al. [48] recently proposed a polynomial time TPL algorithm for standard-cell-based row structure designs. Given a pre-computed set of stitches, the approach guarantees to find a solution with the optimal number of stitches if one exists. Kuang and Young [53] present an approach which is able to find all legal stitch positions in TPL. For all the previous works except our previous work [48], balancing the usage of three masks are not considered. In practice, people are not only interested in achieving a legal TPL decomposition, but also concerned with properly balancing the three masks. Masks with features well distributed on each mask maximally utilize the mask resources, and is beneficial for the manufacturing process.

We [48] proposed a heuristic to globally balance the usage of different masks. The approach is very efficient and achieves very good color balancing results. However, the approach can only handle simple layout with no stitches. Few existing works focus on balancing different masks for complex layout with stitches.



(a) Balancing map for TPL decompositions without color balancing



(b) Balancing map for TPL decompositions with color balancing

Figure 6.6: Balancing map for a simple circuit.

In this section, we focused on color balancing for complex layout with stitches. In practice, the designers are more interested in finding a decomposition with none of the three masks overwhelms the other while minimizing the number of stitches. The color balancing issue is of crucial importance to ensure that consistent and reliable printing qualities can be achieved. By bal-

ancing the usage among different masks, the process variations of the printed features are well controlled, and well-behaved printing characteristics can be expected. With three balanced masks, we can maximally benefit from the manufacturing process, and minimize the printing interference of the features in the same mask. The algorithm is divided into two steps. In the first step, the algorithm in the previous paper [48] is adopted to computed a solution graph of a layout. In the second step, a path-finding algorithm is used to get a balanced TPL decomposition with the optimal number of stitches.

In practice, there are many considerations for TPL decompositions, among which minimizing the number of stitches and properly balancing the usage of the three masks are of great importance. By evenly distributed features on different masks, each mask is properly utilized, and the features are better printed. To visually see the difference of decompositions with and without color balancing, we show two TPL balancing map for the same circuit in Fig. 6.6. The circuit is equally divided by a 100 X 100 grid, where the balancing value in every grid is calculated. The balancing value is calculated as $MAX\{a_i - a_j\}$, where $i = \{1, 2, 3\}$, $j = \{1, 2, 3\}$, and a_i and a_j denote the total area on mask i and j respectively. These values are scaled by the area of the cells within that grid. The lower value of the balancing, the more balanced a decomposition is. We can see that the decomposition considering color balancing is much balanced than the one without considering balancing different masks.

6.2.1 Our Approach

In the following sections, the two steps of our approaches are introduced. In the first step, the previous algorithm [48] is used to compute a solution graph for a given layout. In the second step, a path-finding algorithm is invoked to compute a balanced TPL decomposition with the optimal number of stitches.

Constructing a Weighted Solution Graph

Since we are targeting on complex designs, several steps are involved before getting a weighted solution graph. Many of the concepts are already covered in the previous chapters. Here we briefly go through the terminologies and algorithm for completeness.

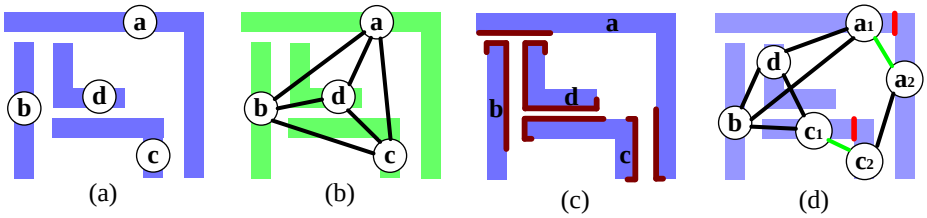


Figure 6.7: A simple example of stitch candidate identification. (a) Input layout. (b) Constraint graph. (c) Node projection results. (d) Final legal stitch candidates.

Stitch Identification: For complex designs, stitches are needed to resolve the coloring conflicts among different features in the layout. Currently, we follow the technique that is adopted in previous papers [25, 48]. Node projection is performed to find all possible legal stitch candidates.

A simple example of how to find all stitch candidates is illustrated in Fig. 6.7. Based on the given layout, the constraint graph is computed, followed by node projection which is shown in Fig. 6.7 (c). All legal stitch candidates are identified based on node projection results, which are shown in Fig. 6.7 (d).

Weighted Solution Graph Construction: Given a layout, we find all the legal stitch candidates and construct its constraint graph. Based on the constraint graph, the weighted solution graph can be computed. The weight of an edge is assigned as follows. If no stitch is needed between the two vertexes it connects, the weight of the edge is assigned to be zero. If m stitches are needed, the weight of the edge is assigned to be $m * L_w * L_h$, where L_w and L_h are the width and height of the input layout respectively. Note that the weight of a stitch is assigned to be the area of the input layout. This guarantees that our path-finding algorithm is able to compute a balanced decomposition with the optimal number of stitches, which will be explained in the following sections.

A simple example of how to construct a weighted solution graph is illustrated in Fig. 6.8. Originally there are four features in the layout. After identifying all legal stitch candidates, there are five features, which is shown in Fig. 6.8 (b). Based on previous TPL algorithm, the solution graph is computed, which is shown in Fig. 6.8 (d). Different from the previous approach, the cost of a stitch is computed as $L_w * L_h$, which ensures that our path-

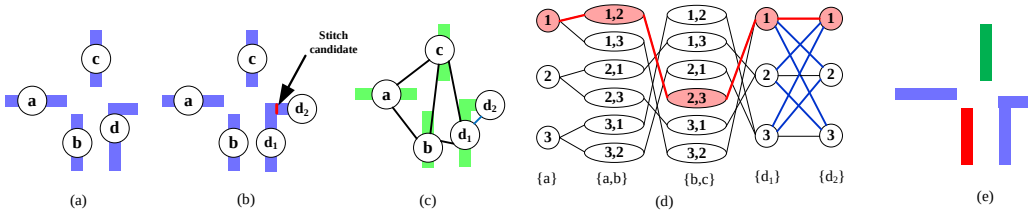


Figure 6.8: An example of constructing weighted solution graph. (a) The input layout with four features. (b) Layout after finding all stitch candidates. (c) Constraint graph. (d) Weighted solution graph for the layout with stitch candidates. The bold blue edges are stitch edges with weight, while other edges are initially assigned zero weight. The highlighted path is a TPL solution after running the path-finding algorithm. (e) Final TPL solution corresponds to the highlighted path. Different colors denote different masks.

finding algorithm achieves optimal number of stitches while maintaining a balanced decomposition.

Table 6.1: Comparisons with Previous Color Balancing Approach

Test Cases	n	Tracks	Stitch Candidates	Our Stitches	Previous Stitches	Our Area Ratio	Previous Area Ratio
C6	179201	143	78102	3420	32534	1:1:1	1:1:1
C7	904292	322	394349	17146	164725	1:1:1	1:1:1
C8	4449681	715	1940587	83916	806357	1:1:1	1:1:1
C9	10031115	1072	4382524	188854	1815545	1:1:1	1:1:1
C10	17813611	1429	7778321	334642	3218362	1:1:1	1:1:1

Path-Finding Algorithm

After the weighted solution graph is constructed, computing a balanced decomposition can be done using a path-finding algorithm. The detailed procedures are discussed as follows.

For each standard cell row, there are three variables, a_1, a_2, and a_3, which denotes the total area of the features which are assigned to mask 1, mask 2, and mask 3 respectively. Initially, the three variables are all zero. We go through the solution graph from the left to right, and assign the new features to the mask that is legal and of the highest priority. The priority

of the three masks are determined as follows: the variable with the highest value is assigned the lowest priority while the variable with the smallest value is assigned the highest priority. If a feature is assigned a mask, the variable denotes the area of the features in that mask will be increased by the area of that feature. For example, if feature f is assigned to mask 2 and the area of feature f is $area_f$, the variable a_2 will be increased by $area_f$.

Now consider the extreme case, where all the features are assigned to mask 1. The difference of a_1 and a_2 is smaller than the cost of a stitch, which is $L_w * L_h$. Therefore, the optimal number of stitches is guaranteed.

Table 6.2: Comparisons with Previous Approach of Computing Optimal Number of Stitches

Test Cases	n	Tracks	Stitch Candidates	Our Stitches	Previous Stitches	Our Area Ratio	Previous Area Ratio
C6	179201	143	78102	3420	3420	1:1:1	1:1.68:1.13
C7	904292	322	394349	17146	17146	1:1:1	1:1.68:1.12
C8	4449681	715	1940587	83916	83916	1:1:1	1:1.68:1.12
C9	10031115	1072	4382524	188854	188854	1:1:1	1:1.68:1.11
C10	17813611	1429	7778321	334642	334642	1:1:1	1:1.68:1.12

TPL Considering Color Balancing Results

Our algorithm is implemented in C++ and tested on a Linux server with 4GB RAM and a 2.8 GHZ CPU. The same benchmarks are used as in the previous paper [48]. Metal 1 is used since they have the most complex shapes.

We compared our approach with the previous simple approach, which neglects the effects of stitches. The detailed results are shown in Table 6.1. The names of the benchmark are kept the same as in the previous paper [48]. The number of polygons are shown in column 2, while the number of standard cell tracks are shown in column 3. Columns 4, 5, and 6 detail the number of stitch candidates, the number of stitch based on our approach, and the number of stitches of the previous approach respectively.

Note that the number of stitches computed by our approach are the same with the optimal results in the previous paper [48]. For the previous color balancing approach, the number of stitches for each benchmark is shown in

column 6. Compared with the previous approach, our new approach significantly reduces the number of stitches while maintaining balanced decompositions among different masks.

Comparisons with the previous approach which computes optimal number of stitches are shown in Table 6.2. The previous approach which computes the optimal number of stitches has no control of balancing different masks. The area ratio for the three masks are shown in the last column of Table 6.2. Compared with the previous approach, our algorithm achieves the optimal number of stitches and maintains a very balanced decomposition at the same time. This clearly verifies the effectiveness of our approach.

6.3 Hybrid Lithography for Triple Patterning Decomposition and E-beam Lithography

Currently, even TPL is not giving satisfying performance for the 14/10 nm technology node and beyond. For complex designs, stitches are still needed to fight with the native conflicts. There has been various new ways to cope with the shrinking feature size in semiconductor fabrications. Several next-generation lithography techniques, such as DSA [39, 40, 41, 42], *extreme ultra-violet* (EUV) [35, 37, 38, 83, 84, 85] and *E-beam* [8, 9, 47], have been studied to resolve the manufacturing difficulties. However, the source power remains an unresolved issue for EUV. E-beam suffers for its low productivity in practice. DSA is still under calibration in research labs and is not mature to be used in practice as a feasible lithography technique.

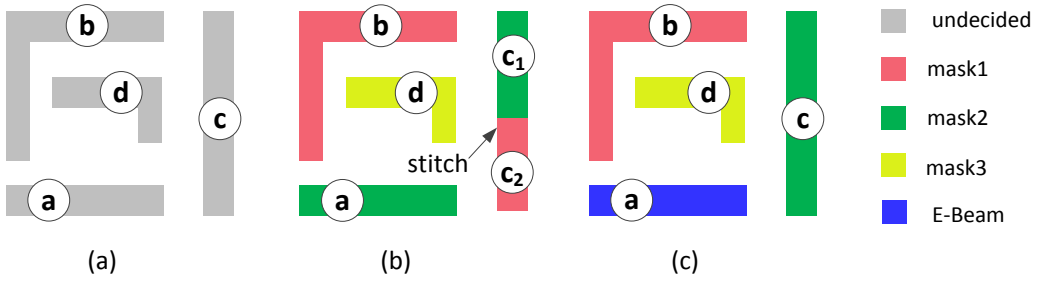


Figure 6.9: Example of hybrid lithography. Different colors denote different masks. (a) Input layout. (b) TPL decomposition with one stitch. (c) Hybrid lithography decomposition with no stitches.

Given the unsatisfying performance of different lithography techniques, people have studied to combine different techniques to cope with the ever increasing difficulties in fabricating the small features. Particularly, optical lithography combined with E-beam lithography has drawn people's attentions [8, 11]. By combining the high throughput optical lithography and high resolution E-beam lithography, both high throughput and high resolution can be achieved. A simple example is shown in Fig. 6.9. For the layout in Fig. 6.9 (a) which is not TPL decomposable, one stitch is needed for TPL decomposition. However, by combining E-beam and TPL, a stitch-free decomposition is achieved by assigning one feature to E-beam lithography, which is shown in Fig. 6.9 (d).

In this section, we studied the pros and cons of two most promising techniques, TPL and E-beam, and investigated combining the merits of the two techniques to provide satisfying solutions for semiconductor fabrications in advanced technology node. Firstly, our previous TPL algorithm is extended to compute a graph that essentially explores all the solution space for the hybrid lithography. Secondly, shortest path algorithm is utilized to compute the decomposition with minimum number of E-beam shots.

6.3.1 Hybrid Lithography

As technology continues to move forward and the feature size keeps shrinking, more and more demanding challenges begin to emerge for semiconductor fabrications. Among them, high throughput and low cost are desired properties for any lithography techniques. However, both E-beam and TPL suffer from several drawbacks, which limit their abilities in practice.

E-beam lithography has been extensively studied in both academic and industries for many years. E-beam lithography is an attractive tool for semiconductor fabrications since it is able to generate patterns at practically very high resolution that is beyond the physical limitations of traditional optical techniques. E-beam is a maskless technique where a charged particle beam is shot directly into the silicon wafer, thus forming the desired layout patterns. There are several types of E-beam techniques while the experiments of this chapter is based on variable shaped beam (VSB). For VSB, the layout is decomposed into a set of rectangles, where all the rectangles are fabricated

sequentially via electronic shot. For E-beam lithography, even with several technological improvements, the low throughput is still one of its main challenges. This limitation has been addressed in many previous papers, and will still be an unresolved issue in the near future.

The problem associated with TPL include the mask image placement, mask-to-mask matching, and CD control for edges defined by multiple separate exposures. Mask making capabilities and cost escalation are also critical for future progress. According to ITRS 2011 [34], to accommodate the optical pattern correction to achieve sub-wavelength imaging, the data growth per node is as high as 2.7X per technology node. Therefore, TPL is much expensive compared with both single patterning and double patterning techniques. Additionally, stitches may be needed to resolve the coloring conflicts of different features, which further increases the manufacturing cost and may lead to yield lost. Moreover, for some complex designs, even TPL fails to generate a legal decomposition.

E-beam has the nice property of very high resolutions while very high throughput can be achieved using TPL. Motivated by the high throughput of traditional immersion lithography and the high resolution of E-beam, we proposed to combine E-beam and TPL together to achieve simultaneous high throughput and high resolutions. The details are discussed in the following sections.

6.3.2 Our Approach

In this section, we are focusing on the standard-cell-based row structure layout, which is the same as in the previous paper [48]. In the standard-cell-based layout, there are power tracks going from the leftmost of the layout to the rightmost of it in each standard cell row. Features in different rows are isolated by the power tracks, thus having no coloring conflicts with each other. Since different rows only share the power tracks, they can be colored independently.

As TPL may introduce stitches, or even fail to resolve coloring conflicts for some designs, it is preferable to incorporate the high-resolution E-beam technique to combat with the manufacturing difficulties. If a feature is fabricated by E-beam, we call it an E-beam feature. Techniques are needed

to minimize the usage of E-beam to ensure both high throughput and high resolution, which will be fully discussed in the following sections.

Building a Weighted Solution Graph

Inspired by our previous TPL algorithm, we first compute a weighted solution graph, where the weight of an edge denotes the number of VSBs needed from one decomposition to another. After that, the shortest path algorithm is adopted to get the optimal number of VSBs needed for a layout. The terminologies of cutting line, cutting line set, constraint graph and solution graph are the same as in the previous paper [48]. Polygon dummy extension is also performed to ensure the correctness of the solution graph. The graph is constructed as follows.

All the cutting lines are traversed from left to right, while the solution graph is dynamically updated. For any cutting line set S_i, all its legal coloring solutions are enumerated. Denote it as N_i. Note that when a feature is assigned as an E-beam feature, no feature will conflict with it. On the contrary, if there is an edge connecting two features in the constraint graph and they are assigned to the same regular mask, coloring conflict occurs and the corresponding decomposition is an illegal one.

Denote the jth solution in N_i as N_i^j. For any two adjacent cutting line solutions N_i^j and N_{i+1}^k and for any feature p_m in cutting line set S_i and p_n in S_{i+1}, we define the compatibility of N_i^j and N_{i+1}^k as follows:

- If p_m conflicts with p_n and they are assigned to the same regular masks, N_i^j and N_{i+1}^k are incompatible.
- If p_m and p_n corresponds to the same feature and they are assigned to different regular masks, N_i^j and N_{i+1}^k are incompatible.
- If p_m and p_n corresponds to the same feature, and p_m is assigned as a E-beam while p_n is assigned to a regular mask, N_i^j and N_{i+1}^k are incompatible.
- For all remaining cases, N_i^j and N_{i+1}^k are compatible.

For any two compatible nodes N_i^j and N_{i+1}^k, an edge is added in the solution graph. The weight of the edge is assigned as the number of VSBs needed

Algorithm 5: Algorithm of Coloring a Single Row

```
1  begin
2      Initialize solution graph G to be empty;
3      P ← all polygons in a standard cell row;
4      Compute constraint graph;
5      Polygon dummy extension;
6      X ← x coordinates of the left boundaries of all polygons in P;
7      w ← size of X;
8      for i ← 1 to w do
9          Find all polygons intersecting with x = X_i;
10         Enumerate solutions for these polygons;
11         Add the solutions into the solution graph G;
12     end
13     Shortest path algorithm on the graph G;
14 end
```

from N_i^j to N_{i+1}^k. As we gradually scan all the cutting lines, the solution graph is incrementally updated. For all adjacent cutting lines, all possible solutions are enumerated, and the weight of all edges are properly captured. Therefore, the solution graph essentially explores all the solution space for a layout. Every path in the solution graph corresponds to legal decomposition, and every legal decomposition corresponds to a path in the graph. Similar observations have been made in the previous paper [48], and proofs are also given in the previous paper [48].

Minimizing the Usage of E-beam

Once we have the weighted solution graph, computing a decomposition becomes straightforward. The shortest path algorithm can be used to compute a decomposition with the minimum number of VSBs needed. Note that since the solution graph is incrementally updated, if we visit all the cutting lines one by one, all the vertices visited are already in topological order. This simplifies the implementation of the shortest path algorithm. The solution computed by the shortest path algorithm guarantees the minimum number of VSBs for a layout. The overall algorithm is shown in Algorithm 5.

Algorithm 6: Hierarchical Algorithm

```
1  begin
2      C_l ← all standard cells in the library;
3      C_r ← all standard cells in a row;
4      foreach Cell C_i in C_l do
5          Build constraint graph G_i;
6          Polygon dummy extension of C_i;
7          Build solution graph S_i;
8      end
9      w ← size of C_r ;
10     for j ← 1 to w do
11         Build partial solution graph G for the first jth cells in C_r;
12     end
13     Shortest path algorithm on the graph G;
14 end
```

Hierarchical Approach

In standard-cell-based designs, millions of the circuit elements are composed from hundreds or thousands type of cells in the cell library. The solution graph of all the cells can be precomputed and reused in a hierarchical way to speed up the algorithm. Techniques of combining the solution graph of different cells are discussed in Chapter 1 and Chapter 5, where all the techniques also apply here. Note that the hierarchical implementation does not affect the optimality of the approach. The number of VSBs computed is still optimal. The overall algorithm are shown in Algorithm 6.

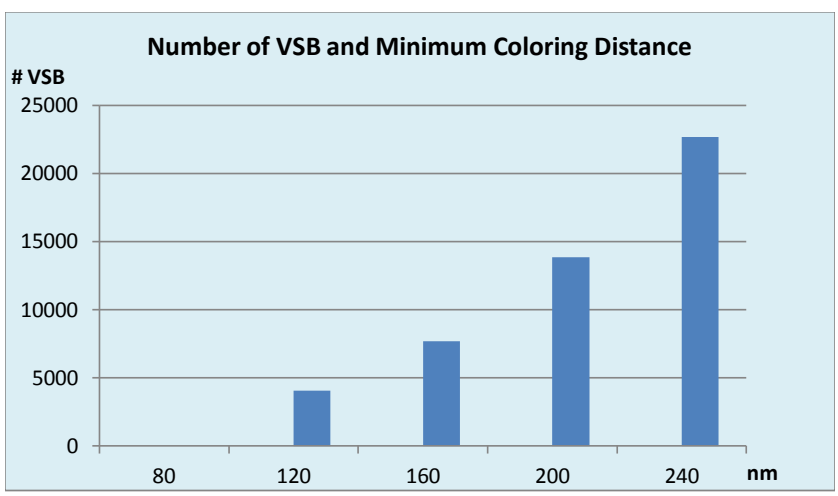


Figure 6.10: Number of VSBs and minimum coloring distance d_{min}.

6.3.3 Hybrid Lithography Results

The algorithm is implemented in C++ and runs on a Linux server with 4GB RAM and four 3.00 GHZ CPU. All benchmarks are from NanGate FreePDK45 Generic Open Cell Library [28], which is available online. Totally 89 types of standard cells are selected from the cell library to evaluate the necessity of using the proposed hybrid lithography. All the cells are TPL decomposable without stitches when $d_{min} = 80$ nm. All the benchmarks are generated by randomly aligning the cells adjacently from the cell library. We have done experiments by shrinking the size of the cells, which mimics the shrinking feature size in more advanced technology nodes. However, instead of shrinking the size of the cell, the minimum coloring distance, d_{min}, is monotonically increased while keeping the size of all cells constant. Note that it has the same effects as shrinking the size of the cells. Wires on metal one layer are used for all experiments. All the power tracks are assumed to be on mask one, and the VSB technique is used to evaluated the number of E-beam shots needed for a design. Note that other E-beam techniques can be easily incorporated to reflect different manufacturing cost. All the results are obtained using the hierarchical implementation.

To show the effectiveness of the hybrid lithography, a quadruple patterning approach is also implemented. For each benchmark with different minimum coloring distances, its solution graph for quadruple patterning lithography is constructed. Note that the way of constructing a quadruple solution graph is very similar with constructing a TPL solution graph, except that for each feature, there are four possible mask assignments.

The detailed results are shown in Table 6.3. The name of the benchmark, the number of rows in the benchmark, and the number of polygons in the benchmarks are shown in columns 1, 2, and 3 respectively. The minimum number of VSBs needed are shown in column 4, while the minimum coloring distance d_{min} is shown in column 5. The runtime is shown in column 6. Whether the layout can be successfully decomposed using triple patterning and quadruple patterning are shown in the last two columns respectively. We can clearly see that, as the minimum coloring distance increases, the minimum number of VSBs also increases. The same trend applies for all benchmarks. The trends on all different benchmarks are also shown using graphs illustrated in Fig. 6.10. In Fig. 6.10, the number of VSBs for all

benchmarks are grouped together based on different d_{min} for clarity.

We can see that for all benchmarks, the number of VSB shots increases consistently as d_{min} increases. When d_{min} is equals to 120 nm, all the layout fails to be decomposed with triple patterning lithography while quadruple patterning works. When d_{min} is larger than 160 nm, all the layout fails to be decomposed even with quadruple patterning, while they can be successfully decomposed using the hybrid lithography with limited number of E-beam shots.

Since increasing d_{min} has the same effects as shrinking the feature sizes, it demonstrates that TPL alone is not enough in more advanced technology nodes. The hybrid lithography is one of the options to achieve simultaneous high resolution and high throughput in advanced technology nodes.

Table 6.3: E-beam and Triple Patterning Decomposition Results

Test Cases	# Row	# p	# VSB	d_{min} (nm)	Runtime (s)	Triple Patterning	Quadruple Patterning
C1	15	1133	0	80	9.8	√	√
			70	120	8.0	×	√
			126	160	7.6	×	×
			228	200	7.0	×	×
			359	240	6.5	×	×
C2	29	4605	0	80	12.7	√	√
			232	120	9.8	×	√
			440	160	9.1	×	×
			921	200	8.3	×	×
			1488	240	7.6	×	×
C3	40	8808	0	80	18.3	√	√
			503	120	12.4	×	√
			909	160	11.5	×	×
			1642	200	10.1	×	×
			2849	200	8.8	×	×
C4	58	18652	0	80	26.4	√	√
			1056	120	18.1	×	√
			2027	160	16.6	×	×
			3631	200	14.1	×	×
			5698	240	11.4	×	×
C5	86	40534	0	80	44.4	√	√
			2180	120	28.4	×	√
			4186	160	26.2	×	×
			7431	200	21.5	×	×
			12285	240	17.5	×	×

6.4 Conclusions

In this chapter, we investigated the pattern-based TPL decompositions. Given a cost aware pattern library, our approach is able to efficiently compute optimal TPL solutions. We also studied color balancing problem in complex designs. Experimental results show that our approach achieves much balanced decompositions than previous algorithms. We also proposed a hybrid lithography technique which combines the merits of TPL and E-beam for advanced technology node. The approach is able to compute the decomposition with minimum number VSB shots for a row structure layout. Extensive experiments are performed for different advanced technology nodes. The results clearly indicate the necessity and the effectiveness of our proposed hybrid lithography framework. Our approach allows engineers to minimize the usage of E-beam, which optimizes the tradeoff between high throughput and high printing resolutions. This work serves as an exploration of hybrid lithography combining TPL and E-beam for advanced technology nodes.

REFERENCES

[1] I.-Y. Kang, H.-S. Seo, B.-S. Ahn, D.-G. Lee, D. Kim, S. Huh, C.-W. Koh, B. Cha, S.-S. Kim, H.-K. Cho et al., "Printability and inspectability of programmed pit defects on the masks in EUV lithography," in *SPIE Advanced Lithography*, vol. 7636, 2010.

[2] E. Spiller, S. L. Baker, P. B. Mirkarimi, V. Sperry, E. M. Gullikson, and D. G. Stearns, "High-performance Mo-Si multilayer coatings for extreme-ultraviolet lithography by ion-beam deposition," *Appl. Opt.*, vol. 42, no. 19, pp. 4049–4058, Jul 2003.

[3] C. H. Clifford, T. T. Chan, and A. R. Neureuther, "Compensation methods for buried defects in extreme ultraviolet lithography masks," in *Proc. of SPIE*, vol. 7636, 2010.

[4] C. H. Clifford and A. R. Neureuther, "Smoothing based fast model for images of isolated buried euv multilayer defects," in *SPIE Advanced Lithography*, 2008, pp. 692 119–692 119.

[5] C. H. Clifford and A. R. Neureuther, "Smoothing based model for images of buried EUV multilayer defects near absorber features," in *Photomask Technology*, 2008.

[6] J. Burns and M. Abbas, "EUV mask defect mitigation through pattern placement," in *SPIE Photomask Technology*, 2010, pp. 782 340–782 340.

[7] B. Lin et al., "Successors of ArF water-immersion lithography: EUV lithography, multi-e-beam maskless lithography, or nanoimprint?" in *J Micro/Nanolith. MEMS MOEMS*, 2008.

[8] Y. Du, H. Zhang, M. D. Wong, and K.-Y. Chao, "Hybrid lithography optimization with e-beam and immersion processes for 16nm 1d gridded design," in *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 707–712.

[9] K. Yuan, B. Yu, and D. Z. Pan, "E-beam lithography stencil planning and optimization with overlapped characters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 2, pp. 167–179, 2012.

[10] K. Yuan and D. Pan, "E-beam lithography stencil planning and optimization with overlapped characters," in *Proceedings of the International Symposium on Physical Design*, 2011, pp. 151–158.

[11] S. Steen, S. J. McNab, L. Sekaric, I. Babich, J. Patel, J. Bucchignano, M. Rooks, D. M. Fried, A. W. Topol, J. R. Brancaccio et al., "Hybrid lithography: The marriage between optical and e-beam lithography. a method to study process integration and device performance for advanced device nodes," *Microelectronic Engineering*, vol. 83, no. 4, pp. 754–761, 2006.

[12] B. Yu, K. Yuan, J.-R. Gao, and D. Pan, "E-blow: E-beam lithography overlapping aware stencil planning for MCC system," in *2013 Design Automation Conference*, 2013, pp. 1–7.

[13] H. Levinson, "Extreme ultraviolet lithography's path to manufacturing," *Journal of Micro*, 2009.

[14] H. Zhang, Y. Du, M. Wong, and R. Topaloglu, "Self-aligned double patterning decomposition for overlay minimization and hot spot detection," in *ACM/EDAC/IEEE Design Automation Conference*, 2011.

[15] H. Zhang, Y. Du, M. Wong, and R. Topaloglu, "Hot spot detection for indecomposable self-aligned double patterning layout," in *Proceedings of SPIE*, 2011.

[16] J. Yang and D. Pan, "Overlay aware interconnect and timing variation modeling for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 488–493.

[17] A. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 465–472.

[18] D. Pan, J. Yang, K. Yuan, M. Cho, and Y. Ban, "Layout optimizations for double patterning lithography," in *IEEE International Conference on ASIC*, 2009, pp. 726–729.

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

[20] J. Yang, K. Lu, M. Cho, K. Yuan, and D. Pan, "A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography," in *Asia and South Pacific Design Automation Conference*, 2010, pp. 637–644.

[21] Y. Xu and C. Chu, "A matching based decomposer for double patterning lithography," in *Proceedings of the International Symposium on Physical Design*, 2010, pp. 121–126.

[22] K. Yuan and D. Pan, "WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 32–38.

[23] S. Chen and Y. Chang, "Native-conflict-aware wire perturbation for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 556–561.

[24] C. Hsu, Y. Chang, and S. Nassif, "Simultaneous layout migration and decomposition for double patterning technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 284–294, 2011.

[25] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Pan, "Layout decomposition for triple patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design*, 2011.

[26] Q. Li, P. Ghosh, D. Abercrombie, P. LaCour, and S. Kanodia, "14nm M1 triple patterning," in *Proceedings of the SPIE*, 2012.

[27] S. Fang, Y. Chang, and W. Chen, "A novel layout decomposition algorithm for triple patterning lithography," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1185–1190.

[28] *Si2 Open Cell Library*, http://www.si2.org/openeda.si2.org/projects/nangatelib.

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

[30] H. Zhang, Y. Du, M. D. F. Wong, and K.-Y. Chao, "Mask cost reduction with circuit performance consideration for self-aligned double patterning," in *2011 16th Asia and South Pacific Design Automation Conference*, 2011, pp. 787–792.

[31] Z. Xiao, Y. Du, H. Zhang, and M. Wong, "A polynomial time exact algorithm for overlay-resistant self-aligned double patterning (SADP) layout decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1228–1239, 2013.

[32] Y. Du, Q. Ma, H. Song, J. Shiely, G. Luk-Pat, A. Miloslavsky, and M. D. F. Wong, "Spacer-is-dielectric-compliant detailed routing for self-aligned double patterning lithography," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 93:1–93:6.

[33] Z. Xiao, Y. Du, H. Zhang, and M. D. Wong, "A polynomial time exact algorithm for self-aligned double patterning layout decomposition," in *Proceedings of the 2012 ACM International Symposium on Physical Design*, 2012, pp. 17–24.

[34] *International Technology Roadmap for Semiconductors: Lithography*, 2011.

[35] H. Zhang, Y. Du, M. D. Wong, Y. Deng, and P. Mangat, "Layout small-angle rotation and shift for EUV defect mitigation," in *IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 43–49.

[36] Y. Du, H. Zhang, and M. D. Wong, "Linear time EUV blank defect mitigation algorithm considering tolerance to inspection inaccuracy," in *SPIE Photomask Technology*, 2012, pp. 85 221R–85 221R.

[37] H. Zhang, Y. Du, M. D. Wong, and R. O. Topalaglu, "Efficient pattern relocation for EUV blank defect mitigation," in *17th Asia and South Pacific Design Automation Conference*, 2012, pp. 719–724.

[38] Y. Du, H. Zhang, M. Wong, and R. Topaloglu, "EUV mask preparation considering blank defects mitigation," in *Proceedings of SPIE*, 2011.

[39] J. Y. Cheng, D. P. Sanders, H. D. Truong, S. Harrer, A. Friz, S. Holmes, M. Colburn, and W. D. Hinsberg, "Simple and versatile methods to integrate directed self-assembly with optical lithography using a polarity-switched photoresist," *ACS Nano*, vol. 4, no. 8, pp. 4815–4823, 2010.

[40] C. Bencher, J. Smith, L. Miao, C. Cai, Y. Chen, J. Y. Cheng, D. P. Sanders, M. Tjio, H. D. Truong, S. Holmes et al., "Self-assembly patterning for sub-15nm half-pitch: A transition from lab to fab," in *SPIE Advanced Lithography*, 2011, pp. 79 700F–79 700F.

[41] G. Schmid, R. Farrell, J. Xu, C. Park, M. Preil, V. Chakrapani, N. Mohanty, A. Ko, M. Cicoria, D. Hetzer et al., "Fabrication of 28nm pitch Si fins with DSA lithography," in *SPIE Advanced Lithography*, 2013, pp. 86 801F–86 801F.

[42] J. Nam, E. S. Kim, D. Kang, H. Yu, K. Kim, S. Yi, C.-H. Shin, and H.-K. Kang, "Patterning process for semiconductor using directed self assembly," in *SPIE Advanced Lithography*, 2013, pp. 868 011–868 011.

[43] Z. Xiao, Y. Du, H. Tian, M. D. Wong, H. Yi, H.-S. P. Wong, and H. Zhang, "Directed self-assembly (DSA) template pattern verification," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[44] Y. Du, Z. Xiao, M. D. Wong, H. Yi, and H.-S. P. Wong, "DSA-aware detailed routing for via layer optimization," in *SPIE Advanced Lithography*, 2014, pp. 90 492J–90 492J.

[45] Z. Xiao, D. Guo, M. D. F. Wong, H. Yi, M. C. Tung, and H.-S. P. Wong, "Layout optimization and template pattern verification for directed self-assembly (DSA)," in *Proceedings of the 52th Annual Design Automation Conference*, 2015.

[46] Y. Du, D. Guo, M. D. F. Wong, H. Yi, H. S. P. Wong, H. Zhang, and Q. Ma, "Block copolymer directed self-assembly (DSA) aware contact layer optimization for 10 nm 1D standard cell library," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 186–193.

[47] D. Lam, D. Liu, and T. Prescop, "E-beam direct write (EBDW) as complementary lithography," in *SPIE Photomask Technology*, 2010.

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

[49] H. Tian, H. Zhang, Q. Ma, and M. D. Wong, "Evaluation of cost-driven triple patterning lithography decomposition," in *SPIE Advanced Lithography*, 2013.

[50] C. Cork, J. Madre, and L. Barnes, "Comparison of triple-patterning decomposition algorithms using aperiodic tiling patterns," in *Proceedings of SPIE*, vol. 7028, 2008, p. 702839.

[51] Y. Chen, P. Xu, L. Miao, Y. Chen, X. Xu, D. Mao, P. Blanco, C. Bencher, R. Hung, and C. Ngai, "Self-aligned triple patterning for continuous IC scaling to half-pitch 15nm," in *Proceedings of SPIE*, vol. 7973, 2011, p. 79731P.

[52] B. Mebarki, H. Chen, Y. Chen, A. Wang, J. Liang, K. Sapre, T. Mandrekar, X. Chen, P. Xu, P. Blanko et al., "Innovative self-aligned triple patterning for 1x half pitch using single spacer deposition-spacer etch step," in *Proceedings of SPIE*, vol. 7973, 2011, p. 79730G.

[53] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 69:1–69:6.

[54] Q. Ma, H. Zhang, and M. D. F. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology," in *Proceedings of the 49th Annual Design Automation Conference*, 2012.

[55] N. Een and N. Sorensson, "The minisat page," http://minisat.se/Main.html.

[56] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in *Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 170–177.

[57] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Young, "Ripple: A robust and effective routability-driven placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1546–1556, 2013.

[58] M.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang, "Routability-driven analytical placement for mixed-size circuit designs," in *IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 80–84.

[59] N. Viswanathan and C.-N. Chu, "Fastplace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 722–733, 2005.

[60] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *IEEE/ACM International Conference on Computer-Aided Design,*, 2005, pp. 48–55.

[61] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 50–60, 2012.

[62] B. Yu, X. Xu, J.-R. Gao, and D. Z. Pan, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

[63] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. D. Wong, "Constrained pattern assignment for standard cell based triple patterning lithography," in *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

[64] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E. F. Young, "Ripple: An effective routability-driven placer by iterative cell movement," in *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 74–79.

[65] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: Placement based on novel rough legalization and refinement," in *Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 357–362.

[66] A. B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *Proceedings of Asia and South Pacific Design Automation Conference*, 1999, pp. 241–244.

[67] U. Brenner and J. Vygen, "Faster optimal single-row placement with fixed ordering," in *Proceedings of Design Automation and Test in Europe Conference and Exhibition*, 2000, pp. 117–121.

[68] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and details of a high quality, large-scale analytical placer," in *IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 891–898.

[69] "Msuncore max sat solver," http://logos.ucd.ie/wiki/doku.php?id=msuncore.

[70] Y. Du, D. Guo, M. D. Wong, H. Yi, P. Wong, H. Zhang, and Q. Ma, "Block copolymer directed self-assembly (DSA) aware contact layer optimization for 10 nm 1d standard cell library," in *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

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

[72] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

[73] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. D. Wong, "Triple patterning aware detailed placement with constrained pattern assignment," in *IEEE/ACM International Conference on Computer-Aided Design*, 2014.

[74] M. Gupta, K. Jeong, and A. B. Kahng, "Timing yield-aware color reassignment and detailed placement perturbation for bimodal CD distribution in double patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 8, pp. 1229–1242, 2010.

[75] Synopsys Inc., "Design solutions for 20nm and beyond," 2012.

[76] A. N. V. and A. Mandal, "Timing analysis comprehending mask misalignment due to double patterning," in *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 2014, pp. 82–84.

[77] K. Jeong, A. B. Kahng, and R. O. Topaloglu, "Is overlay error more important than interconnect variations in double patterning?" in *Proceedings of the International Workshop on System Level Interconnect Prediction*, 2009, pp. 3–10.

[78] K. Chow, "Are multi-patterning corners really needed for 16/14 nm?" in *EE Times*, 2014.

[79] N. D. Arora, K. V. Raol, R. Schumann, and L. M. Richardson, "Modeling and extraction of interconnect capacitances for multilayer VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 1, pp. 58–67, 1996.

[80] R. S. Ghaida and P. Gupta, "Within-layer overlay impact for design in metal double patterning," *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 3, pp. 381–390, 2010.

[81] H.-A. Chien, S.-Y. Han, Y.-H. Chen, and T.-C. Wang, "A cell-based row-structure layout decomposer for triple patterning lithography," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 67–74.

[82] H. Tian, H. Zhang, Z. Xiao, and M. D. Wong, "An efficient linear time triple patterning solver," in *Asia and South Pacific Design Automation Conference*, 2015, pp. 208–213.

[83] Y. Du, H. Zhang, and M. D. Wong, "Linear time EUV blank defect mitigation algorithm considering tolerance to inspection inaccuracy," in *Proc. of SPIE*, vol. 8522, 2012, pp. 85 221R–1.

[84] H. Zhang, Y. Du, M. D. Wong, and R. O. Topalaglu, "Efficient pattern relocation for EUV blank defect mitigation," in *Asia and South Pacific Design Automation Conference*, 2012, pp. 719–724.

[85] Y. Du, H. Zhang, M. D. Wong, and R. O. Topaloglu, "EUV mask preparation considering blank defects mitigation," in *SPIE Photomask Technology*, 2011.