© 2014 Daifeng Guo

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

BY

DAIFENG GUO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor Martin D. F. Wong

ABSTRACT

Electron beam lithography (EBL) is a very promising candidate for integrated circuit (IC) fabrication beyond the 10 nm technology node. To address its throughput issue, the Character Projection (CP) technique has been proposed, and its stencil planning can be optimized with awareness of overlapping characters. However, the top-level 2D stencil planning problem has been proven to be an NP-hard problem. As its most essential step, the 1D row ordering is believed hard as well, and no polynomial time optimal solution has been provided so far. Previous research formulates the problem as the travelling salesman problem, which is NP-hard and solves it by heuristics. In this thesis, we formulate the problem as a matching problem and propose a polynomial time optimal algorithm, which serves as the major subroutine for the entire stencil planning problem. The optimality of the algorithm is proved, and experimental results are also provided to show that our work makes a great improvement in efficiency and correctness to solve the row ordering problem.

To my parents, for their love and support

ACKNOWLEDGMENTS

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

The most intuitive version of EBL is electron beam direct write (EBDW), which shoots the desired patterns pixel by pixel, and thus has very low throughput. One essential improvement of EBL is the variable shaped beam (VSB) [4, 5, 6], which can print an arbitrarily sized rectangle with one single shot. However, since current layout designs contain billions of rectangles, the throughput of VSB is still incapable of meeting the requirement. To further improve the throughput of EBL, Character Projection (CP) (later multi-column cell (MCC)) has been proposed [7, 8], which is capable of printing an entire character (e.g. a standard cell) with one shot.

There are two major challenges in CP. First, how to design the set of projection characters; second, how to plan the stencil to pack as many characters as possible. The former problem is investigated by [8, 9, 10]. For the latter problem, placement optimization should be performed based on

the fact that the characters can overlap at the blank margins located at the character boundaries, as illustrated in Fig. 1.1. The blank margin is used to reserve some space for the scattered electrons after they pass through the aperture [7]. By sharing the blank area in Figs. 1.1(b) and (c), the characters occupy less stencil area than those in Fig. 1.1(a). Obviously, different placements of the characters result in different area occupation as illustrated by Figs. 1.1(b) and (c), because the shared blank margins in total are different among different placement solutions. For a given set of characters, it is a challenging problem to find their optimal placement, such that they occupy the smallest stencil area and leave more room to insert additional characters or features.

(a) Stencil planning without blank margin sharing

(b) Stencil planning with blank margin sharing

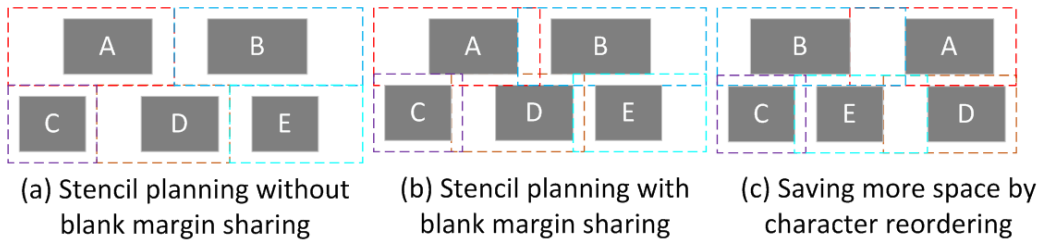(c) Saving more space by character reordering

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

In the stencil planning problem, it is reasonable to assume that the characters are selected from standard cells or vertical slices of cells, which have the same heights. In addition, those standard cell characters also share very similar top and bottom blank margins, because a standard cell usually has power tracks on the top and bottom, and the distance that scattered electrons can travel outside the character is highly dependent on the pattern near its boundaries. With such assumptions, we do not need to consider the vertical placement constraints, and in consequence, the original character placement problem can be reduced as a row ordering problem, which has been proposed as the 1D overlapping aware stencil planning (OSP) problem in previous works [11, 12, 13]. Several attempts have been made to solve this problem. However, Yuan, Yu and Pan [12] formulated this problem as an NP-hard problem, and they either provided a heuristic approach or made quite a number of assumptions to support their solutions, i.e. the difference between left and right blank margins is very small. Those assumptions not only need to be proved by realistic litho-experimental results, but also make

the problem much easier. In this thesis, we neatly solve the general row ordering problem by a polynomial time optimal algorithm and we prove its optimality rigorously. This algorithm can be adopted as the key subroutine for character selection and distribution in higher-level EBL stencil planning.

The rest of the thesis is organized as follows. Chapter 2 formulates the overall optimization problem. Chapter 3 explain the previous attempt to address the problem. Then the polynomial time optimal algorithm is provided in Chapter 4. In Chapter 5, we prove the optimality of our algorithm and analyze its complexity. Experimental results are reported in Chapter 6, and finally, Chapter 7 concludes the thesis.

CHAPTER 2

PROBLEM FORMULATION

In this thesis, we target solving the 1D row ordering problem for stencil planning in EBL. Given a set of n characters $C = \{c_1, c_2, ..., c_n\}$, where each character c_i has left blank margin l_i and right blank margin r_i as shown in Fig. 2.1 (a), we can generate a set of blank margin pairs associated with C, denoted by $C_p = \{(l_1, r_1), (l_2, r_2), ..., (l_n, r_n)\}$. By reordering C_p, a sequence of pairs can be obtained, which is denoted by $S_p = \{(l_{s_1}, r_{s_1}), (l_{s_2}, r_{s_2}), ..., (l_{s_n}, r_{s_n})\}$. We define its cost by the total length of blank margins occupied by all characters after blank margin sharing, as described in Eq. 2.1:

$$Cost_{S_p} = l_{s_1} + \Sigma_{i=1}^{n-1} max(r_{s_i}, l_{s_{i+1}}) + r_{s_n} \tag{2.1}$$

For example in Fig. 2.1(b), if we place the three characters in the order of $\{c_i, c_k, c_j\}$, the sequence cost would be $l_i + max(r_i, l_k) + max(r_k, l_j) + l_j$. On the other hand, if we reorder them to be $\{c_j, c_i, c_k\}$, the sequence cost can be reduced accordingly. Based on that, we define the row ordering problem below.
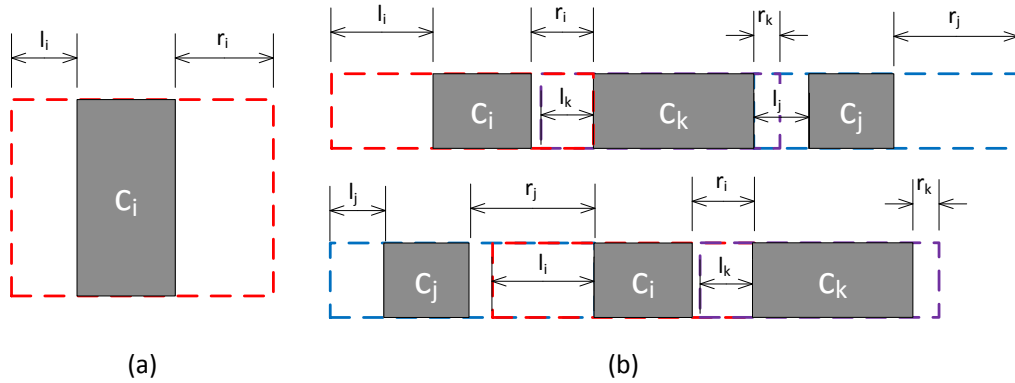

Figure 2.1: Blank margin overlapping

Row Ordering Problem (ROP): *Given a set of blank margin pairs C_p, find its optimal order S_p, such that the sequence cost $Cost_{S_p}$ is minimal.*

CHAPTER 3

PREVIOUS WORK

Previously, the problem was studied and heuristics were provided. In this chapter, the former method of [12] was reviewed and discussed.
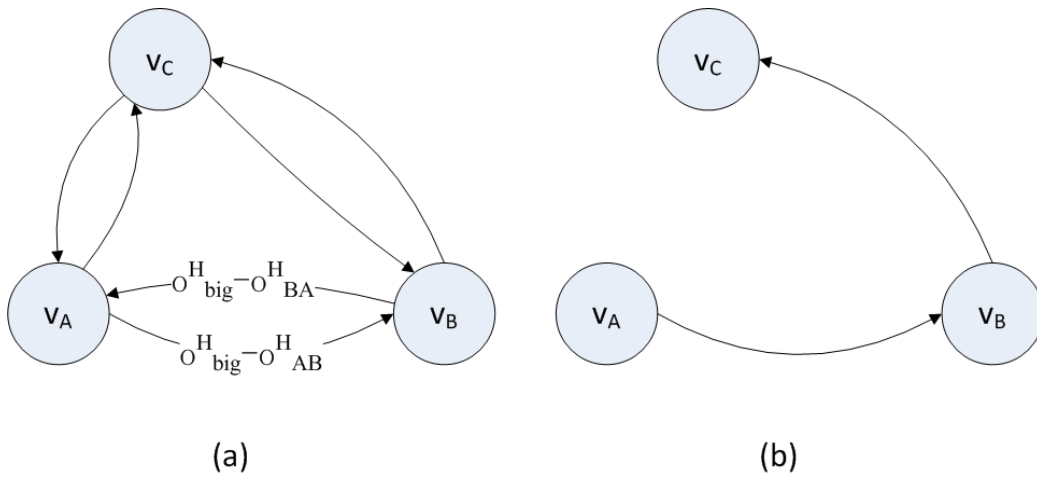

Figure 3.1: Graph for finding the min cost Hamiltonian path

The goal of the problem is to minimize the total length of all characters in a row, namely $Cost$, such that the remaining capacity of the row is maximized. $o_{i,j}^H$ is defined as the maximum allowed overlap when two candidate characters c_i and c_j are put adjacent to each other horizontally. In other words, $o_{i,j}^H = min(r_i, l_j)$. For the same reason illustrated later in Section 4.2, minimizing the length is the same as maximizing the overall overlapped blanking margins $\Sigma_{i=0}^{n-1} o_{i,i+1}^H$. Based on the observation, Yuan, Yu and Pan [12] formulated a minimum cost Hamiltonian path problem. First, a graph is constructed as follows. Each character c_i is represented by a vertex v_i. Two directed edges e_{ij} and e_{ji} are added between two vertices v_i and v_j. The cost of those edges are $o_{big}^H - o_{i,j}^H$ and $o_{big}^H - o_{j,i}^H$ respectively. If c_i locates to the left of c_j, the shared space is $o_{i,j}^H$, and if c_i locates to the right of c_j, the shared space is $o_{j,i}^H$. Additionally, o_{big}^H is a constant which is bigger than any of $o_{i,j}^H$ and $o_{j,i}^H$. So a path that visits each vertex exactly once defines an ordering of

the characters, and the total weight of that path, namely $\Sigma_{e \in Path}(o^H_{big} - o^H_{i,j})$ should be minimized in order to maximize $\Sigma_{i=0}^{n-1} o^H_{i,i+1}$. Note that there is one difference from our problem formulation in Chapter 2. The blank margins at the beginning and the end are ignored here. More evidence is still needed to determine whether those blank margins are necessary, however, it does not affect the problem nor our solution, because both cases with and without blank margins at the beginning and the end can be addressed by our algorithm according to Section 4.2.1.

As shown in Fig. 3.1(a), a graph of all characters of A, B and C is constructed. Fig. 3.1(b) shows the resulting min cost Hamiltonian path in the graph. The corresponding ordering is shown in Fig. 3.2.

Figure 3.2: Resulting order of characters by the min cost Hamiltonian path method

The problem of the minimum cost Hamiltonian path is NP-hard and solving the whole row at one time would not be practical. Yuan, Yu and Pan [12] use heuristics that partition the row into multiple segments, and solve each segment by the Hamiltonian path based method. This method will be compared with our algorithm presented in this thesis in Chapter 6 by experiment results. Generally, the heuristic cannot provide an accurate order of characters with minimal length and it also lacks of efficiency. The reason is that Yuan, Yu and Pan [12] formulate the real problem into a much harder problem and could not solve it without taking advantages of our problem's conditions. Our algorithm will be presented in Chapter 4.

CHAPTER 4

ALGORITHM

In this chapter, we will illustrate and discuss our algorithm step by step. In Chapter 5, we will prove the optimality and the time efficiency.

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

4.1   From Order to Matching

We create a complete bipartite graph G, namely $K_{N,N}$ by making all left blank margins r_i as indices in one set and all right margins l_i as indices in the other set, and connect all possible l_i and r_i as shown in Fig. 4.1(a). The edge (r_x, l_y) has the weight $e_i = max(r_x, l_y)$, and means that the original pairs (l_x, r_x) and (l_y, r_y) can be connected in the order of $(l_x, r_x)(l_y, r_y)$. For an order of the pairs, there is a corresponding matching between the left and right blank margins l_i and r_i in the bipartite graph. As shown in Fig. 4.1(b), for an order $S_p = \{(l_1, r_1), (l_2, r_2), ..., (l_n, r_n)\}$, we connect the adjacent r_x with l_{x+1}, and the edges not in the matching are not shown. For instance, for two pairs (l_1, r_1) and (l_2, r_2) which are ordered as $(l_1, r_1), (l_2, r_2)$, we create an edge to connect r_1 and l_2. In this way, we have a matching as shown in Fig. 4.1(b), in which all numbers are connected by an edge except for l_1 and r_n. We call the matching with one edge less than the perfect matching as almost-perfect matching. If we add a dummy edge of l_1 and r_n, then we have a perfect matching of the graph. As a result, the optimization problem becomes the following:

   Weighted Almost-Perfect Matching Problem (WAMP): *Find an almost-perfect matching in G by deleting one of the matching edges from a*

perfect matching in G, such that a set of edges E_s of the matching is able to define an order of the given pairs with the lowest Cost, where $Cost = \Sigma_{i=1}^{N-1} e_i \forall e_i \in E_s$.

Also, we use $Cost^*$ to represent the cost of a perfect matching, and $Cost$ is the cost of almost-perfect matching. Note that, though an order has a corresponding matching, conversely an almost-perfect matching is not always able to define an order of the pairs. One counterexample is shown in Fig. 4.2(b) and will be illustrated in Section 4.2. The problem as well as the general **WAMP** will be addressed in the following sections.
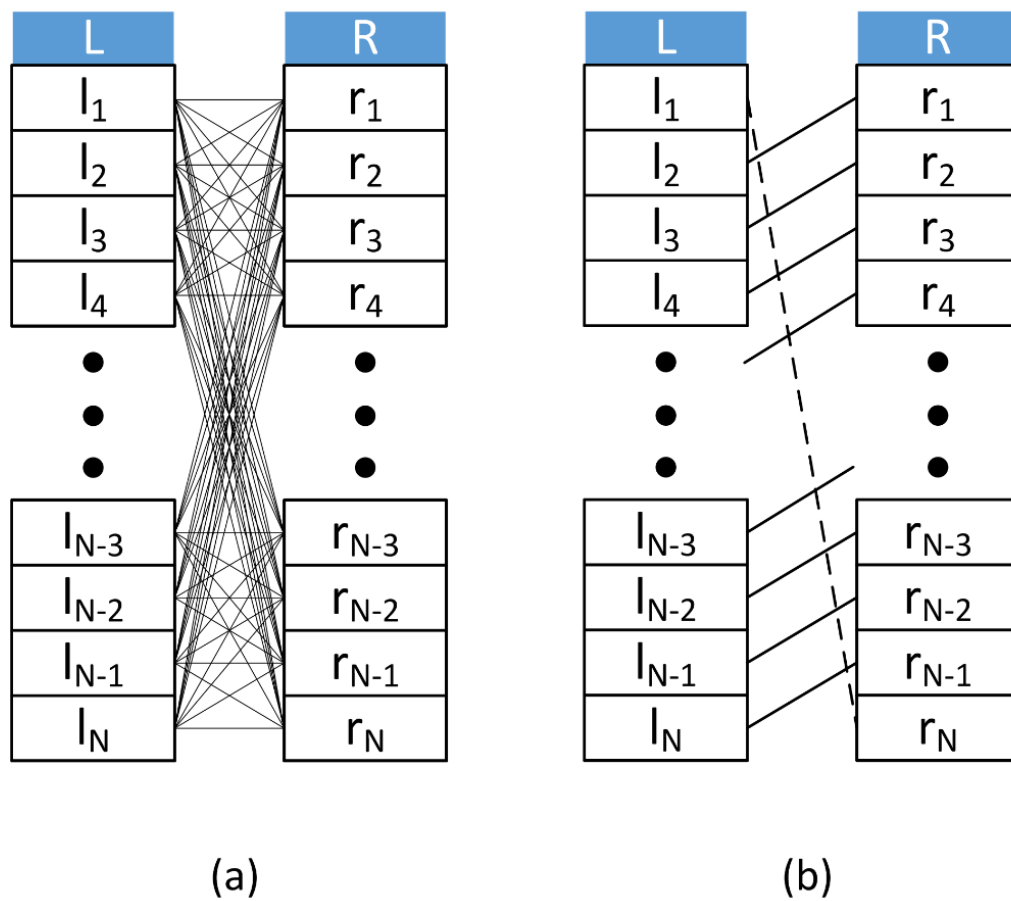

Figure 4.1: Matching between left and right components

4.2 Lower Bound by Sorting and Naive Matching

Aiming to minimize the *Cost* of **WAMP**, we notice that the summation of all left margins l_i as well as all right margins r_i will always be a constant Q. Thus, we can transform our cost function as:

$$\begin{aligned}
Cost &= \Sigma_{i=1}^{N-1} e_i \forall e_i \in E \\
&= ls_1 + \Sigma_{i=1}^{N-1} max(r_{s_i}, l_{s_{i+1}}) + rs_N \\
&= \Sigma_{i=1}^{N}(l_i + r_i) - \Sigma_{i=1}^{N-1} min(r_{s_i}, l_{s_{i+1}}) \\
&= Q - \Sigma_{i=1}^{N-1} min(r_{s_i}, l_{s_{i+1}})
\end{aligned} \tag{4.1}$$

Consequently, we can think of our target as a maximum function:

$$\min Cost \Rightarrow \max \Sigma_{i=1}^{N-1} min(r_{s_i}, l_{s_{i+1}}) \tag{4.2}$$

In Eq. 4.2, if we name the results of $min(r_{s_i}, l_{s_{i+1}})$ as the eliminated numbers, we want to select as large as possible eliminated numbers. On the other hand, for one eliminated number, there must exist one larger number which is only paired with this eliminated number. Thus, intuitively matching two numbers with a big difference is not desirable because it means a waste of potential to save more area. This leads us to first sort the numbers.

By the discussion above, we sort the left blank margins l_i and right blank margins r_i independently in the descending order of their value, as illustrated in Fig. 4.2. For future convenience, we denote the sorted array of left components as L and the sorted array of right components as R, as shown in Fig. 4.2(b). In order to match up numbers with the smallest differences, we adopt a naive matching strategy by connecting the r_{i_x} and l_{j_x} with the same index x in the sorted array. Specifically, if any pair of entries r_x and l_y have the same array index after sorting, we connect them with an edge as illustrated in Fig. 4.2(b), meaning that the original pairs (l_x, r_x) and (l_y, r_y) are arranged in the order of $(l_x, r_x)(l_y, r_y)$. Once all left and right components are connected, we have a perfect matching for all the numbers, namely an assignment for their neighbors.

However, as mentioned before, perfectly matching R and L by the same index probably will not result in a valid ordered sequence of pairs, but one or multiple cycles. For instance, if after sorting we have R and L as shown

9
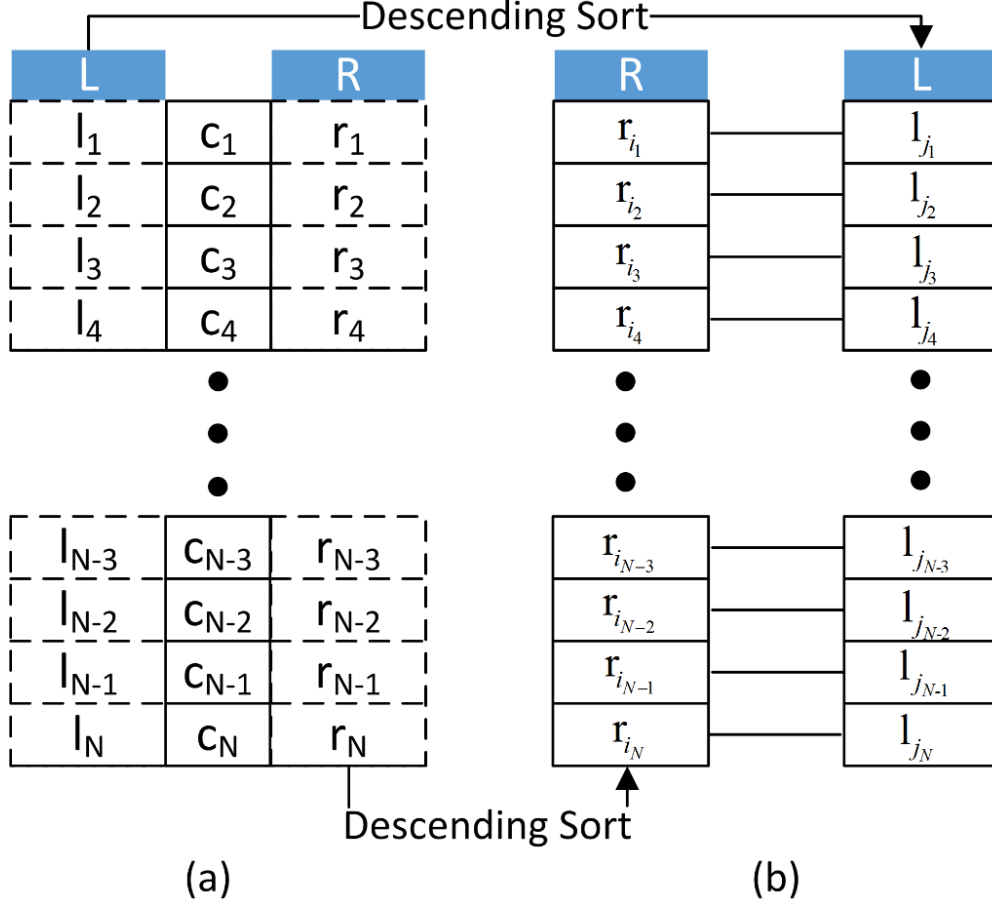
Figure 4.2: Transformation of two arrays

in Fig. 4.3(a), we will end up with one cycle of pairs corresponding to the perfect matching. On the other hand, if R and L are ordered as shown in Fig. 4.3(b), we will have three cycles.

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

Lemma 1 $Cost^*_{IDEAL} \leq Cost^*_\omega$ for all $\omega \in \Omega$

The proof will be given in Chapter 5. Then we discuss the cases of one and multiple cycles.

10

(a)

R		L
r_3		l_7
r_2		l_1
r_7		l_6
r_8		l_4
r_4		l_2
r_5		l_3
r_1		l_5
r_6		l_8

(l_8, r_8)-(l_4, r_4)-(l_2, r_2)-(l_1, r_1)

(l_5, r_5)-(l_3, r_3)-(l_7, r_7)-(l_6, r_6)

(b)

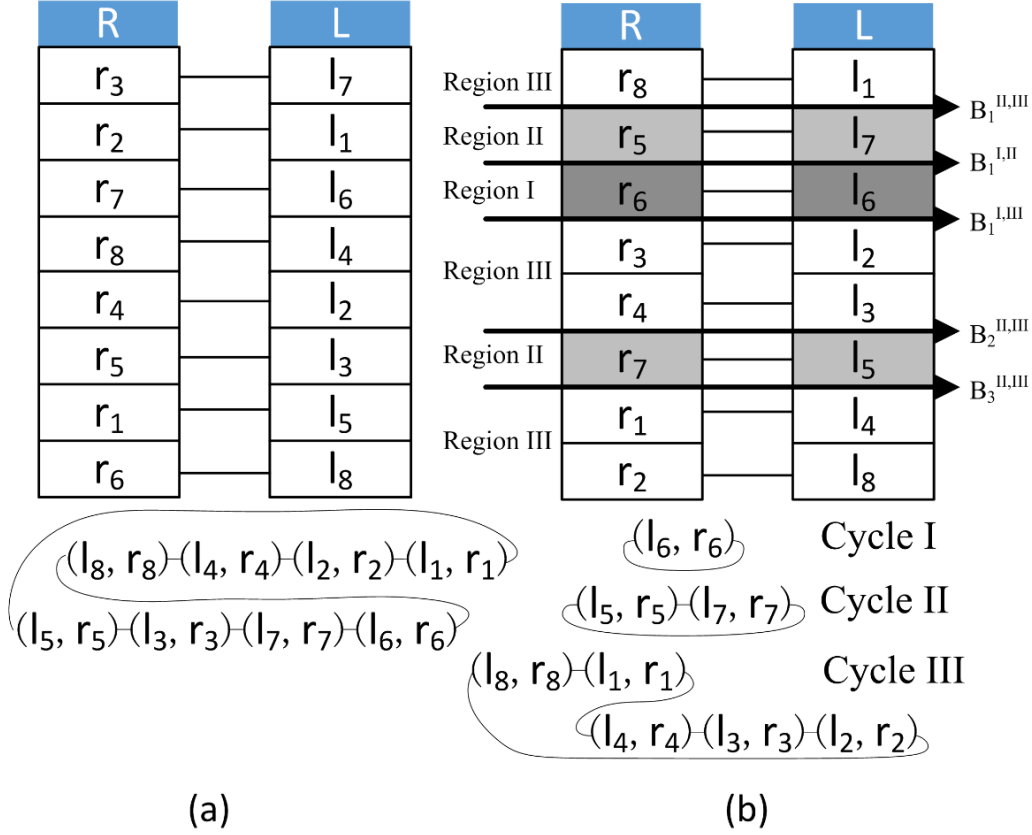|	R		L	
Region III	r_8		l_1	$B_1^{II,III}$
Region II	r_5		l_7	$B_1^{I,II}$
Region I	r_6		l_6	$B_1^{I,III}$
Region III	r_3		l_2	
	r_4		l_3	$B_2^{II,III}$
Region II	r_7		l_5	$B_3^{II,III}$
Region III	r_1		l_4	
	r_2		l_8	

(l_6, r_6) Cycle I

(l_5, r_5)-(l_7, r_7) Cycle II

(l_8, r_8)-(l_1, r_1) Cycle III

(l_4, r_4)-(l_3, r_3)-(l_2, r_2)

Figure 4.3: Cycles analysis

4.2.1 One cycle

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

$$
\begin{aligned}
Cost^*_\beta &= \max(l_{\beta_1}, r_{\beta_N}) + \Sigma_{i=1}^{N-1} max(r_{\beta_i}, l_{\beta_{i+1}}) \\
&= Cost_B - \phi \text{ , where } \phi \in L \bigcup R
\end{aligned}
\tag{4.3}
$$

To obtain the almost-perfect matching with the smallest cost increment ϕ from perfect matching, we pick the edge of the smallest number in the set of $L \bigcup R$ to delete, because deleting one edge means breaking one of the N maximization braces in Eq. 4.3, and the smaller term in the brace

would be ϕ. So ϕ has to be the smallest number in $L \bigcup R$. This gets the almost-perfect matching and a valid sequence without losing any optimality. Since the smallest number is always in the bottom of the array R and L, we just need to delete the last edge in the perfect matching. In the example of Fig. 4.3(a), we cut the edge (r_6, l_8) and have the sequence as $(l_8, r_8)(l_4, r_4)(l_2, r_2)(l_1, r_1)(l_5, r_5)(l_3, r_3)(l_7, r_7)(l_6, r_6)$.

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

The problem formulated by previous work in Chapter 3 is also solved, if we delete the edge with the largest blank margin. A cycle β defines a min cost Hamiltonian tour of the graph in Chapter 3, then a Hamiltonian path can be obtained by deleting one edge. The deleting action will take off one of the maximum braces in Eq. 4.3. Because the maximum brace with the largest value always appears in the tour, it is chosen to delete such that the resulting Hamiltonian path has the smallest cost.

4.2.2 Multiple cycles

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

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

4.3   Multiple Cycles Analysis

If we have multiple cycles after naive matching, the remaining problem would be how to get a feasible solution and guarantee the optimality at the same time. In this chapter, we will defines several notations used to address this issue in Section 4.4. In order to make it clear, we use the example shown in Fig. 4.3(b) to illustrate them.
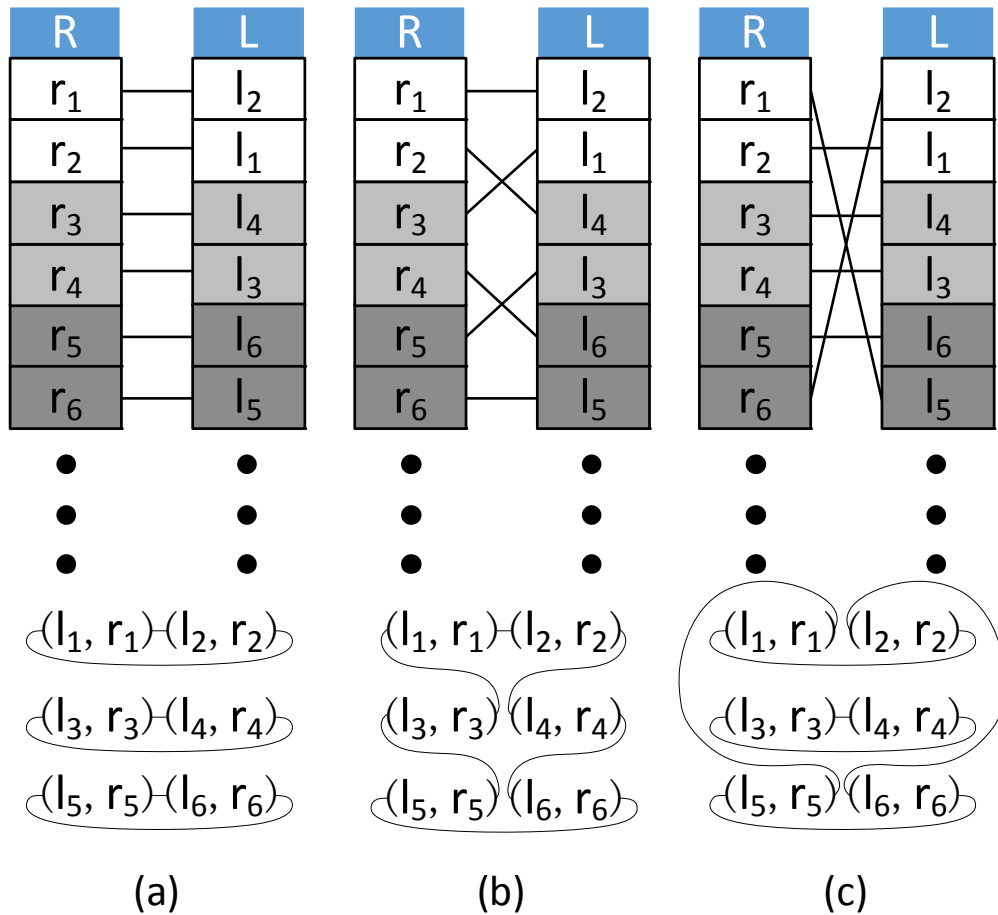

Figure 4.4: Edge-switch

4.3.1   Region

In the ideal case, we can divide the sorted array R and L into several regions such that one region represent one cycle. As shown in Fig. 4.3(b), Region I represents Cycle I and similarly for regions I, II, III, and they are

distinguished by different colors. Note that it is not necessarily true that one region is formed by consecutive matched pairs. It can consist of multiple intervals of consecutive matched pairs, i.e. region III.

4.3.2 Boundary

We use B to represent the boundary between two adjacent regions in the arrays. As shown in Fig. 4.3(b), $B_i^{\mathrm{I,II}}$ denotes the ith boundary between regions I and II.

4.3.3 Relation

We define Relation to be the value ordering of the four numbers involved in any two matching edges. As shown in Figs. 4.5(a) and (b), the numbers involved are $r_{i_u}, r_{i_x}, l_{j_v}, l_{j_y}$, and we have that $r_{i_u} > r_{i_x}$ and $l_{j_v} > l_{j_y}$. Without loss of generality, we can assume that $r_{i_u} > l_{j_v}$, because other cases with $r_{i_u} < l_{j_v}$ are just symmetrical, and we do not need to discuss them again. Then there are three cases of their relation:

Type 1: $r_{i_u} > l_{j_v} > r_{i_x} > l_{j_y}$.

Type 2: $r_{i_u} > l_{j_v} > l_{j_y} > r_{i_x}$.

Type 3: $r_{i_u} > r_{i_x} > l_{j_v} > l_{j_y}$.

4.3.4 Edge-switch and $\Delta Cost$

Edge-switch basically means the exchange between two ending points of any two matching edges. It helps us merge cycles. For example in Fig. 4.4(a), there are three cycles in the ideal case. If we do two edge-switches at the boundary between (r_2, l_1), (r_3, l_4) and the boundary (r_4, l_3), (r_5, l_6), then three cycles get merged as shown in Fig. 4.4(b). Obviously, any two matching edges can be switched, and if they are from two different regions then two cycles get merged. $\Delta Cost$ is the increment of $Cost^*$ of the matching during an edge-switch. To make it clear, we can put edge-switch into two categories to discuss following:

Type 1: Edge-switch from non-crossing to crossing.

As shown in Fig. 4.5, from (a) to (b), it is a type 1 edge-switch, since two matching edges are not crossing each other in (a) but they are crossing in (b). We discuss its $\Delta Cost$ in three different types of relation between the numbers involved in this edge-switch.

1. Type 1 Relation:

   In Fig. 4.5(a), $Cost^* = r_{i_u} + r_{i_x}$ before the edge-switch. In Fig. 4.5(b), $Cost^* = r_{i_u} + l_{j_v}$ after the edge-switch. Thus, $\Delta Cost = l_{j_v} - r_{i_x} > 0$.

2. Type 2 Relation:

   In Fig. 4.5(a), $Cost^* = r_{i_u} + l_{j_y}$ before the edge-switch. In Fig. 4.5(b), $Cost^* = r_{i_u} + l_{j_v}$ after the edge-switch. Thus, $\Delta Cost = l_{j_v} - l_{j_y} > 0$.

3. Type 3 Relation:

   In Fig. 4.5(a), $Cost^* = r_{i_u} + r_{i_x}$ before the edge-switch. In Fig. 4.5(b), $Cost^* = r_{i_u} + r_{i_x}$ after the edge-switch. Thus, $\Delta Cost = 0$.

Consequently, for type 1 edge-switch, $\Delta Cost_{Type1} \geq 0$.

Type 2: Edge-switch from crossing to non-crossing.

As shown in Fig. 4.5, from (b) to (a), it is type 2, and we also discuss its $\Delta Cost$ in three cases.

1. Type 1 Relation:

   In Fig. 4.5(b), $Cost^* = r_{i_u} + l_{j_v}$ before the edge-switch. In Fig. 4.5(a), $Cost^* = r_{i_u} + r_{i_x}$ after the edge-switch. Thus, $\Delta Cost = r_{i_x} - l_{j_v} < 0$.

2. Type 2 Relation:

   In Fig. 4.5(b), $Cost^* = r_{i_u} + l_{j_v}$ before the edge-switch. In Fig. 4.5(a), $Cost^* = r_{i_u} + l_{j_y}$ after the edge-switch. Thus, $\Delta Cost = l_{j_y} - l_{j_v} < 0$.

3. Type 3 Relation:

   In Fig. 4.5(b), $Cost^* = r_{i_u} + r_{i_x}$ before the edge-switch. In Fig. 4.5(a), $Cost^* = r_{i_u} + r_{i_x}$ after the edge-switch. Thus, $\Delta Cost = 0$.

Consequently, for type 2 edge-switch, $\Delta Cost_{Type2} \leq 0$.

From the case study above, we can find that the value of $\Delta Cost$ can be determined in Table 4.1.
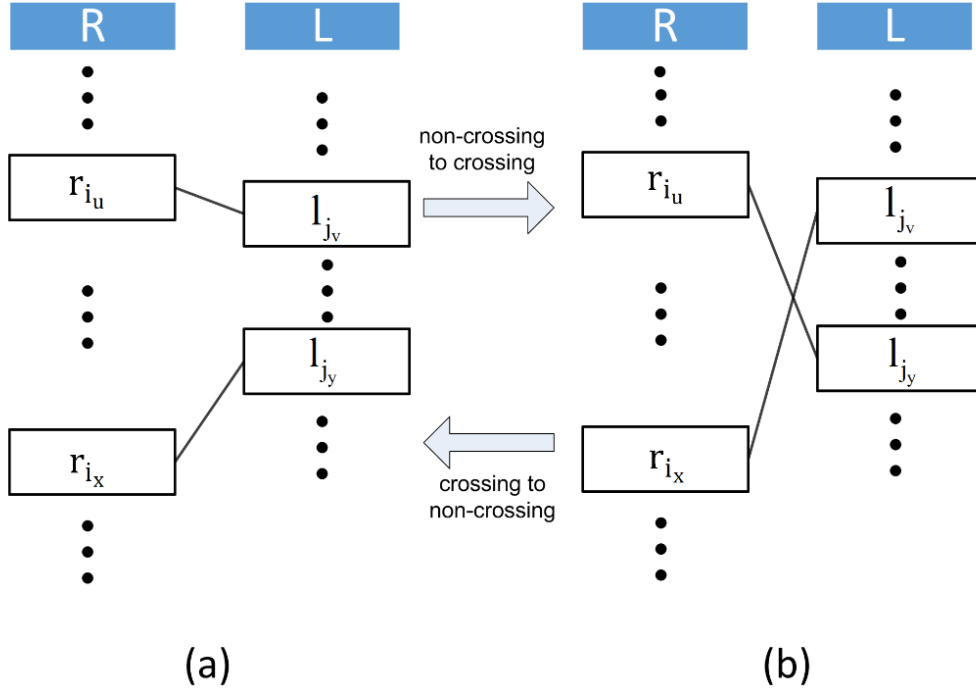
Figure 4.5: Different types of edge-switch

Table 4.1: $\Delta Cost$

$\Delta Cost$		Edge-switch	
		Type 1	Type 2
Relation	Type 1,2	$min(r_{i_u}, l_{j_v})$ $-max(r_{i_x}, l_{j_y})$	$-min(r_{i_u}, l_{j_v})$ $+max(r_{i_x}, l_{j_y})$
	Type 3	0	0

So, if the two numbers on one side are both larger than the two on the other side, namely type 3 relation, the $\Delta Cost$ of the edge-switch is always zero. Otherwise, the absolute value of $\Delta Cost$ is the difference between the smaller one of the top two numbers and the larger one of the bottom two numbers. The sign of $\Delta Cost$ is determined by the type of the edge-switch.

4.3.5 Edge-switch-on-boundary

We define edge-switch-on-boundary literally as an edge-switch which takes place right on the boundary such that all four numbers involved are located right on the boundary. As shown in Fig. 4.4(b), there are two edge-switch-on-boundaries. For other cases of edge-switch, they are edge-switch-not-on-

boundary, i.e. the only edge-switch as shown in Fig. 4.4(c).

4.4 Cycle Merging

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

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

Benefitting from the sorted array, we can merge cycles by merging regions in R and L. For any two regions, we can pick any edge from each region and switch them in order to merge the regions. But in our algorithm, we just consider the edge-switch-on-boundaries, such as the case shown in Fig. 4.4(b), because of Lemma 2.

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

The proof is given in the Chapter 5. Additionally, all edge-switch-on-boundaries in between can merge all regions in that area instead of just two regions. For instance as shown in Figs. 4.4(b) and (c), if you choose (r_1, l_2) and (r_6, l_5) to switch like (c), it would be better to switch (r_2, l_1) and (r_3, l_4) as well as (r_4, l_3) and (r_5, l_6), because they have the smaller $\Delta Cost$ by Lemma 2 and not only merge two regions but all the three regions from (a). So we only need to consider the edge-switch-on-boundary as our potential selection for edge-switch.

With all possible edge-switch-on-boundaries and their $\Delta Cost$, we can construct a graph H by assigning a vertex for each region and connect two vertices if the regions that they represents have a common boundary. Addi-

tionally, the distance of each edge in H is the $\Delta Cost$ of the edge-switch on the corresponding boundary. For example in Fig. 4.3(b), we can construct a graph as shown in Fig. 4.6(a), where $\Delta Cost_i^{\text{I,II}}$ means the $\Delta Cost$ of the edge-switch on the $B_i^{\text{I,II}}$.
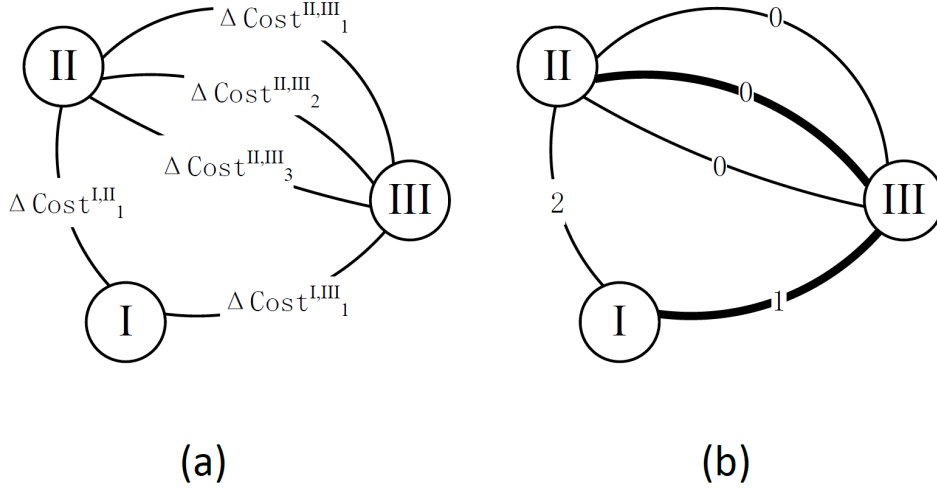

(a)   (b)

Figure 4.6: Find the optimal solution by performing the MST algorithm

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

4.4.1 Minimum spinning tree

We use the same example in Fig. 4.3(b) to explain our algorithm. If we have all pairs as $(l_1, r_1) = (15, 3)$, $(l_2, r_2) = (10, 1)$, $(l_3, r_3) = (8, 9)$, $(l_4, r_4) = (5, 6)$, $(l_5, r_5) = (7, 16)$, $(l_6, r_6) = (12, 11)$, $(l_7, r_7) = (14, 4)$, $(l_8, r_8) = (2, 17)$, then from top to bottom:

$$\Delta Cost_1^{\text{II,III}} = (r_8 + r_5) - (r_8 + r_5) = 0,$$
$$\Delta Cost_1^{\text{I,II}} = (r_5 + l_7) - (r_5 + l_6) = 2,$$
$$\Delta Cost_1^{\text{I,III}} = (r_6 + l_6) - (l_6 + l_2) = 1,$$
$$\Delta Cost_2^{\text{II,III}} = (l_3 + l_5) - (l_3 + l_5) = 0,$$
$$\Delta Cost_3^{\text{II,III}} = (l_5 + l_4) - (l_5 + l_4) = 0.$$

Then we use Kruskal's algorithm [14] to find the MST shown in Fig. 4.6(b). In this example, the MST contains $B_1^{\text{II,III}}$ and $B_2^{\text{I,III}}$.

R | **L** | | **R** | **L**

Region III — $r_8=17$ — $l_1=15$ — $B_1^{II,III}$ | r_8 — l_1
Region II — $r_5=16$ — $l_7=14$ — $B_1^{I,II}$ | r_5 — l_7
Region I — $r_6=11$ — $l_6=12$ — $B_1^{I,III}$ | r_6 — l_6
Region III — $r_3=9$ — $l_2=10$ | r_3 — l_2
$r_4=6$ — $l_3=8$ — $B_2^{II,III}$ | r_4 — l_3
Region II — $r_7=4$ — $l_5=7$ — $B_3^{II,III}$ | r_7 — l_5
Region III — $r_1=3$ — $l_4=5$ | r_1 — l_4
$r_2=1$ — $l_8=2$ | r_2 — l_8

(a)
Cycle I: $\langle l_6, r_6 \rangle$
Cycle II: $\langle l_5, r_5 \rangle$—$\langle l_7, r_7 \rangle$
Cycle III: $\langle l_8, r_8 \rangle$—$\langle l_1, r_1 \rangle$
$\langle l_4, r_4 \rangle$—$\langle l_3, r_3 \rangle$—$\langle l_2, r_2 \rangle$

(b)
$\langle l_6, r_6 \rangle$
$\langle l_5, r_5 \rangle$—$\langle l_7, r_7 \rangle$
$\langle l_3, r_3 \rangle \langle l_2, r_2 \rangle$
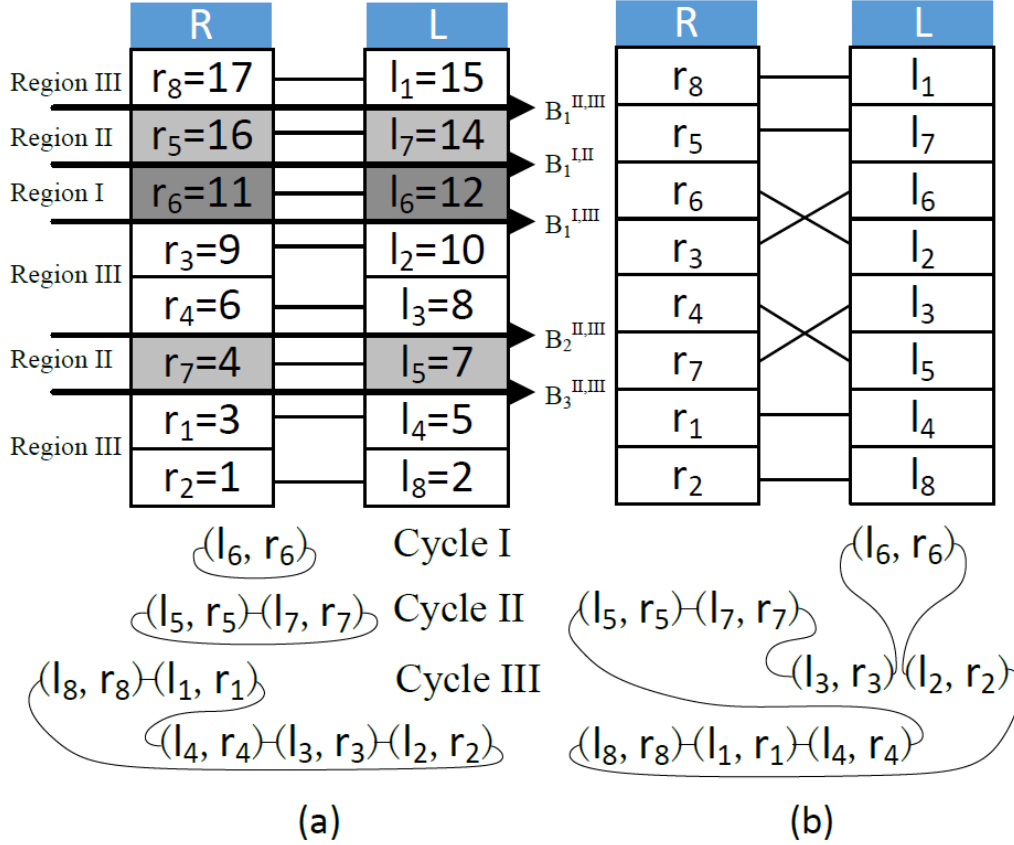$\langle l_8, r_8 \rangle$—$\langle l_1, r_1 \rangle$—$\langle l_4, r_4 \rangle$

Figure 4.7: Merge cycles based on MST

4.4.2 Edge-switch after MST

Next, we need to perform the edge-switches picked by the MST algorithm.

We switch the edge on $B_1^{II,III}$ and $B_2^{I,III}$ and thus obtain a valid solution of only one cycle, as shown in Fig. 4.7(b). Finally, we have the final $Cost^*_{ALG}$ as

$$Cost^*_{ALG} = Cost^*_{IDEAL} + \Sigma \Delta Cost(e), \forall e \in \text{MST} \qquad (4.4)$$

There is one circumstance that we need to discuss a little more. As shown in Fig. 4.8(a), after finding the MST, if we want to do edge-switches on both $B_1^{I,II}$ and $B_1^{II,III}$, then as shown in Fig. 4.8(b), after we switch edges on $B_1^{I,II}$, the problem is that we no longer have the edge-switch-on-boundaries on $B_1^{II,III}$ available. However, we can do the edge-switch between edge (r_{i_3}, l_{j_3}) and either edge (r_{i_1}, l_{j_2}) or edge (r_{i_2}, l_{j_1}).

The solution is that we always easily select the edge containing the smaller value of r_{i_2} and l_{j_2} to switch with (r_{i_3}, l_{j_3}). Without loss of generality, if
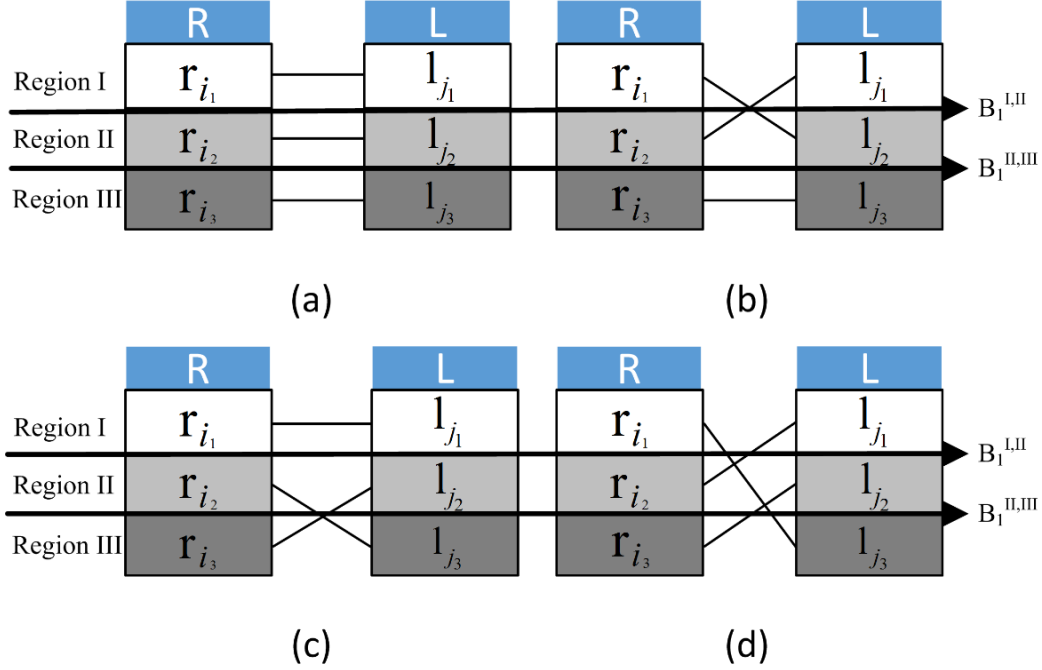
Figure 4.8: Edge-switch example

$r_{i_2} > l_{j_2}$, we select the edge (r_{i_1}, l_{j_2}) and do the switch as shown in Fig. 4.8(d). After this step, we claim that we have the same total $\Delta Cost = \Delta Cost_1^{I,II} + \Delta Cost_1^{II,III}$ as what we desire. The reason is as follows.

In Fig. 4.8, we already have $\Delta Cost_1^{I,II}$ toward the total $\Delta Cost$ from (a) to (b). So, in order to merge cycle II and cycle III, we just need to show that the $\Delta Cost$ of (b) to (d) is still $\Delta Cost_1^{II,III}$ which is the $\delta Cost$ of (a) to (c). The four numbers involved in those two edge-switch operations, $(r_{i_2}, l_{j_2}, r_{i_3}, l_{j_3})$ in (a) and $(r_{i_1}, l_{j_2}, r_{i_3}, l_{j_3})$ in (b) have the same relation type, because $r_{i_1} > r_{i_2} > l_{j_3}$ implies that the largest number among the four becomes r_{r_1} from r_{i_2} and all other numbers stay the same. According to Table 4.1, the largest value in the relation does not affect the $\delta Cost$ of the edge-switch. Thus, from (b) to (d), the $\Delta Cost$ is still $\Delta Cost_1^{II,III} = l_{j_2} - r_{i_3}$. Even if we have more consecutive edges that need to be switched, we just need to adopt this technique iteratively. As a result, we solve this problem without losing any optimality.

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

Algorithm 1: ROP's algorithm

Data: A set of margin pairs of (l_i, r_i)
Result: An order of pairs with minimal $Cost$

Construct a complete bipartite graph G;   **1**
Obtain an almost-perfect matching by solving **WAMP**;   **2**
return the order determined by the almost-perfect matching;   **3**

Algorithm 2: WAMP's algorithm

Data: Graph G built by pairs of (l_i, r_i)
Result: A minimal $Cost$ almost-perfect matching creating an order of
         pairs

Sort r_i and l_i respectively;   **1**
Naive Matching with same index;   **2**
switch *Number of cycles after the naive matching* **do**   **3**
   case *One cycle*   **4**
      Delete the last edge in the sorted G;   **5**
      return the almost-perfect matching;   **6**
   end   **7**
   case *Multiple cycles*   **8**
      Merge all cycles by solving **CMP**;   **9**
      Go to **case** *One cycle*;   **10**
   end   **11**
endsw   **12**

Algorithm 3: CMP's algorithm

Data: Cycles (regions) determined by naive matching
Result: One cycle (region) with minimal sum of all $\delta Cost$

Construct a graph H by regions and Boundaries;   **1**
Find the MST in H by Kruskal's algorithm;   **2**
Switch the edges picked by the MST;   **3**
return the cycle after edge-switches;   **4**

CHAPTER 5

PROOF

In this chapter, we will prove the optimality of the algorithm we presented in Chapter 4.

Note that there might be more than one optimal ordering, but our algorithm can only output one of them. If we have the optimal solution OPT which has smaller cost $Cost_{OPT}$ than our algorithm's $Cost_{ALG}$, we will show that this is not possible. We also use $Cost_{IDEAL}$ to represent the case just after the naive matching with the lower bound of $Cost$. Because of the reason stated in Section 4.1, we think of ordering as matching instead, in other words, proving the optimality of **WAMP** instead of proving **ROP** directly. Note that if we just have one cycle after the naive matching, then we use cut strategy to have the optimal solution. So we just need to consider the multiple cycles case and its $Cost^*_{IDEAL}$, $Cost^*_{ALG}$ and $Cost^*_{OPT}$. In order to determine the relationship between the ideal case and all possible ordering, we have the following lemma.

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

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

$$Lemma\ 3 \Rightarrow Lemma\ 1 \tag{5.1}$$

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

* **Proof of Lemma 3:**

Base step: $N = 1$. As shown in the Fig. 5.1(a), there is only one possible ordering. $N = 2$. There are two possible matching cases as shown respectively in Figs. 5.1(b) and (c). One type 1 edge-switch can be done from the ideal case (b) to (c).
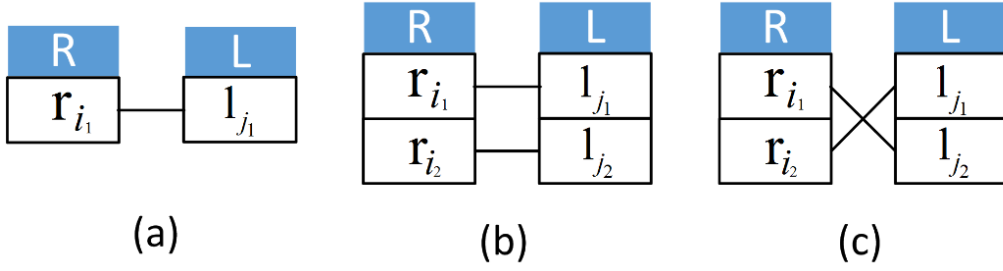
Inductive hypothesis:

22

Figure 5.1: Base cases

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

When $N = k + 1$, we have $k + 1$ pairs. Say we have an arbitrary perfect matching between R and L as shown in Fig. 5.2(a). As shown Fig. 5.2(b), we have another perfect matching in Fig. 5.2(b) which is the same as in Fig. 5.2(a) except for the edges between $r_{i_1}, l_{j_1}, r_{i_x}, l_{j_y}$. It is obvious that from Fig. 5.2(b) to Fig. 5.2(a), we just need one type 1 edge-switch step. For (b), the bottom k pair matchings, by the hypothesis, can be transformed from the ideal case by finite steps of type 1 edge-switch. Thus, with one more edge-switch of type 1, we can always achieve an arbitrary perfect matching of $k + 1$ pairs. So the lemma is true. Next, we prove Lemma 2.

Proof of Lemma 2:

Refer to Fig. 4.4. If we do an edge-switch-not-on-boundary of (r_x, l_x) with (r_y, l_y) for $0 \leq x < y \leq N$, and there are M edge-switch-on-boundaries of (r_{m_i}, l_{m_i}) with (r_{m_i+1}, l_{m_i+1}) such that $m_i \geq x$ and $m_i + 1 \leq y$, then we want to show that edge-switch-not-on-boundary $\Delta Cost^{nb}$ is larger or equal to $\Sigma_{i=1}^{i=M} \Delta Cost_{m_i}^{ob}$ where $\Delta Cost_{m_i}^{ob}$ represents the $\Delta Cost$ for the edge-switch of (r_{m_i}, l_{m_i}) with (r_{m_i+1}, l_{m_i+1}).

1. Type 3 relation of r_x, r_y, l_x, l_y

   $\Delta Cost^{nb} = 0$ by Table 4.1. Additionally, since $r_x > r_y > l_x > l_y$, $m_i \geq x$ and $m_i + 1 \leq y$, we have $\forall m_i, r_{m_i} > r_{m_i+1} > l_{m_i} > l_{m_i+1}$ and $\Delta Cost_{m_i}^{ob} = 0$. Thus, $\Delta Cost^{nb} = \Sigma_{i=1}^{i=M} \Delta Cost_{m_i}^{ob} = 0$, and the lemma is true in this case.

2. Type 1 and 2 relations of r_x, r_y, l_x, l_y

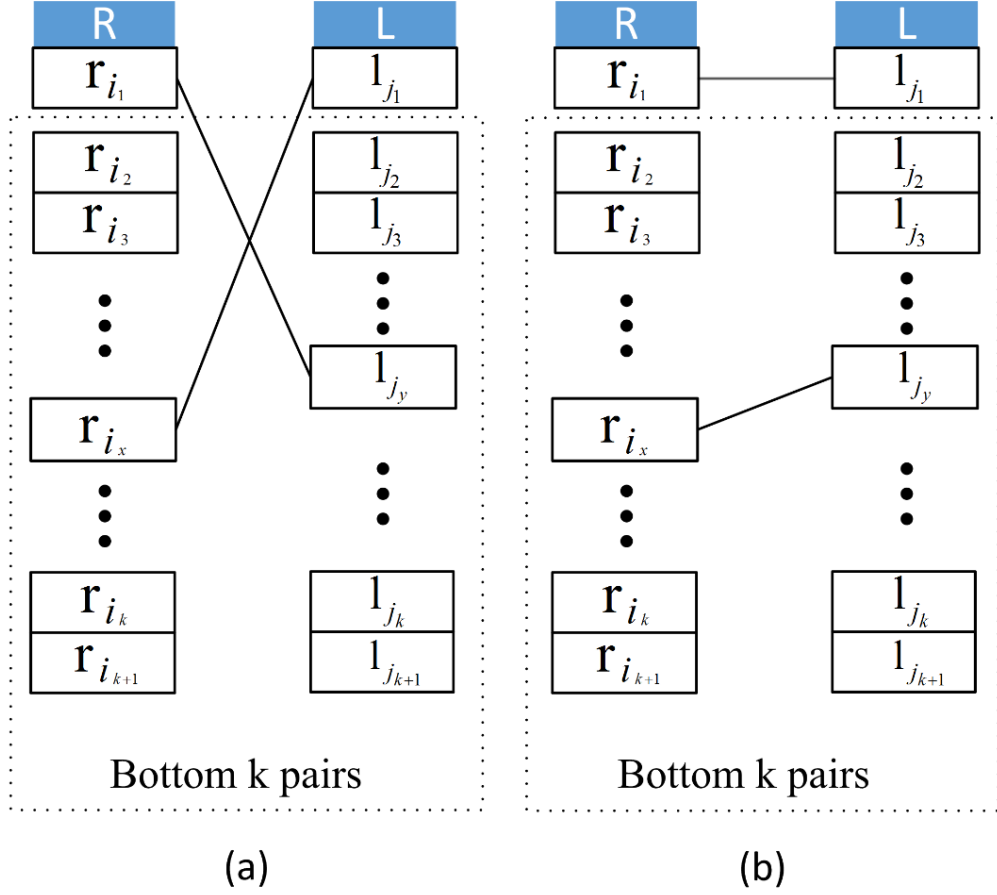   Assume that we have a counterexample such that $\Delta Cost^{nb} < \Sigma_{i=1}^{i=M} \Delta Cost_{m_i}^{ob}$.

23

Figure 5.2: Inductive steps

Then by Table 4.1, we have

$$\Delta Cost^{nb} < \Sigma_{i=1}^{i=M} \Delta Cost^{ob}_{m_i}$$

$$min(r_x, l_x) - max(r_y, l_y)$$

$$< min(r_{m_1}, l_{m_1})$$

$$\underbrace{-max(r_{m_1+1}, l_{m_1+1}) + min(r_{m_2}, l_{m_2})}_{<0}$$

$$\underbrace{-max(r_{m_2+1}, l_{m_2+1}) + ... + min(r_{m_M}, l_{m_M})}_{<0}$$

$$- max(r_{m_M+1}, l_{m_M+1})$$

$$< min(r_{m_1}, l_{m_1}) - max(r_{m_M+1}, l_{m_M+1})$$

Note that $-max(r_{m_i+1}, l_{m_i+1}) + min(r_{m_{i+1}}, l_{m_{i+1}}) \leq 0, \forall i \leq M$, since r_{m_i+1} and l_{m_i+1} are above the $r_{m_{i+1}}$ and $l_{m_{i+1}}$ in the sorted R and L

arrays. But it is not possible because $min(r_x, l_x) \geq min(r_{m_1}, l_{m_1})$ and $max(r_y, l_y) \leq max(r_{m_M+1}, l_{m_M+1})$. Thus, there is no counterexample and the lemma is true in this case.

So, Lemma 2 is true.

Proof of optimality:

If we have the optimal solution OPT, by Lemma 3, it can be achieved by finite steps of type 1 edge-switch from the ideal case. At the starting point of the ideal case, we have multiple cycles, but for any one of those switches, if its four numbers involved are all inside the region, it cannot make any contribution to merge the cycles. Thus, some of those switches must be cross two regions. Besides, all regions must be merged, so those switches must touch all cycles. Thus, those switches can construct an spinning tree in a graph H' where there is one vertex for each cycle (region), and there is an edge (u, v) with $\Delta Cost_{(u,v)}$ for one of all possible switches that crosses any two different regions u and v. Next, we just need to prove that this spinning tree in H' has the total $\Delta Cost$ larger than or equal to the total $\Delta Cost$ of the MST in H defined in Section 4.4.

By Lemma 2, H' can be transformed into H by a way that for every edge of edge-switch-not-on-boundary, replace it by one or more edges of edge-switch-on-boundary, and then merge the edges with the identical $\Delta Cost$. Additionally by the Lemma 2, after the transformation, the smallest $\Delta Cost$ between any two vertices stays the same. Thus, the MST in H is also the MST in H'. As a result, we prove that the spinning tree of OPT in H' has the total $\Delta Cost$ larger than or equal to the total $\Delta Cost$ of the MST in H by our algorithm. By Eq. 4.4, $Cost^*_{OPT} \geq Cost^*_{ALG}$. Then, by Eq. 4.3 $Cost_{OPT} \geq Cost_{ALG}$. OPT could not be more optimal than ALG, so our algorithm can output an optimal solution.

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

Generally, we have found that all possible solutions can be transformed from the ideal case and our algorithm can give the optimal solution which

complete the transformation with the smallest *Cost* increment. Thus, our algorithm can generate the optimal solution for the stencil row planning problem.

CHAPTER 6

EXPERIMENTAL RESULTS

We implement our algorithm in C++ and test it on a Linux workstation 2.5G Hz CPU and 126 GB memory. Since the previous works [11, 12, 13] use some assumptions to generate characters having similar left and right blank margins, which might not be realistic, we create our own benchmarks with a set of characters with blank margins generated randomly. They are controlled to be less than the actual character size (the area impossible to be overlapped), equaling to 1000 in our benchmark. Input is a certain set of characters, and output is the minimum total length of those characters in a row. We run our algorithms and only algorithms of [11, 12] on our benchmarks, since [13] does not improve the row planning and cannot insert more characters than [11, 12]. The comparison result is shown in Table 6.1. The number of characters is reported in column 1. The total length of blank margins, namely $Cost$, and running time are reported for all three algorithms. Speed-up and length improvement are also calculated. Because of the running time issue in [12], last two test cases are not reported for it.

Table 6.1: Comparison results

#CP	[12]		[11]		ALG					
	Cost	CPU(s)	Cost	CPU(s)	Cost	CPU(s)	Improve [12]	SpdUp [12]	Improve [11]	SpdUp [11]
96	104000	6.54	132400	0.099	95924	0.00178	7.7%	x3666	27.5%	x55.5
294	306000	306.3	389980	0.104	288664	0.005627	5.6%	x54427	26.0%	x18.5
495	505000	2376.3	657980	0.122	478280	0.009335	5.2%	x254555	27.3%	x13.1
581	619000	3949.3	819000	0.136	582730	0.010847	5.8%	x364094	28.8%	x12.5
767	NA	Hours	1033000	0.157	766197	0.015108	NA	NA	25.8%	x10.4
929	NA	Hours	1271000	0.172	937129	0.017747	NA	NA	26.3%	x9.7

As shown in Table 6.1, compared to [12], we can improve the result by around 6% with a huge speed-up, because [12] uses a Hamiltonian path based method, which approximates the result but is still not efficient. By the comparison to [11], we can improve the result a lot and also have good speed-up. And the runtime confirms that the time complexity of our algorithm is $O(n \log n)$ where n denotes the number of characters. Because of the limited

data set, those heuristics might perform poorly in future industry data. On the other hand, since we have already proved the optimality of our algorithm, it can always achieve the best solution theoretically and the experiments also confirm that. Adopting our algorithm, we save space in a row, thus more characters can be inserted into the stencil and further reduce the number of shots needed to print the layout. Additionally, our algorithm becomes essential if the number of characters is large.

CHAPTER 7

CONCLUSION

In this paper, we propose a polynomial time optimal algorithm to solve the 1D row ordering problem for EBL stencil planning. Optimality proof is provided, and the efficiency of our algorithm is also verified by the experimental results. In the CP technology, our algorithm serves as a key subroutine for the high-level character selection and distribution problem. Those problems are proved NP-hard, but any solution of them can still benefit significantly from our solution for this 1D row ordering problem.

REFERENCES

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

[2] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on.* IEEE, 2011, pp. 1–8.

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

[4] A. Fujimur, "Beyond light: The growing importance of e-beam," in *Proc. Int. Conf. on Computer Aided Design*, 2009.

[5] A. Fujimur, "Design for e-beam: Getting the best wafers without the exploding mask costs," in *Proceedings of International Symposium on Quality Electronic Design*, 2010.

[6] K. Yoshida, T. Mitsuhashi, S. Matsushita, L. L. Chau, T. D. T. Nguyen, D. MacMillen, and A. Fujimura, "Stencil design and method for improving character density for cell projection charged particle beam lithography," Sep. 2 2009, US Patent App. 12/552,373.

[7] T. Fujino, Y. Kajiya, and M. Yoshikawa, "Character-build standard-cell layout technique for high-throughput character-projection eb lithography," in *Photomask and Next Generation Lithography Mask Technology XII.* International Society for Optics and Photonics, 2005, pp. 160–167.

[8] M. Sugihara, T. Takata, K. Nakamura, Y. Matsunaga, and K. Murakami, "A CP mask development methodology for MCC systems," in *Photomask and Next Generation Lithography Mask Technology XIII.* International Society for Optics and Photonics, 2006, pp. 62 833J–62 833J.

[9] M. Sugihara, T. Takata, K. Nakamura, R. Inanami, H. Hayashi, K. Kishimoto, T. Hasebe, Y. Kawano, Y. Matsunaga, K. Murakami et al., "Technology mapping technique for throughput enhancement of character projection equipment," in *SPIE 31st International Symposium on Advanced Lithography*. International Society for Optics and Photonics, 2006, pp. 61 510Z–61 510Z.

[10] M. Sugihara, "Optimal character-size exploration for increasing throughput of MCC lithographic systems," in *SPIE Advanced Lithography*. International Society for Optics and Photonics, 2009, p. 72710L.

[11] B. Yu, K. Yuan, J.-R. Gao, and D. Z. Pan, "E-blow: e-beam lithography overlapping aware stencil planning for MCC system," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 70.

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

[13] J. Kuang and E. F. Young, "A highly-efficient row-structure stencil planning approach for e-beam lithography with overlapped characters," in *Proceedings of the 2014 on International Symposium on Physical Design*. ACM, 2014, pp. 109–116.

[14] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.