Incorporating Cut Redistribution with Mask Assignment to Enable 1D Gridded Design *

Jian Kuang, Evangeline F. Y. Young, Bei Yu
Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, NT, Hong Kong
{jkuang,fyyoung,byu}@cse.cuhk.edu.hk

ABSTRACT

1D gridded design is one of the most promising solutions that can enable the scaling to $10nm$ technology node and beyond. Line-end cuts are needed to fabricate 1D layouts, where two techniques are available to resolve the conflicts between cuts: cut redistribution and cut mask assignment. In this paper, we consider incorporating the two techniques to enable the manufacturing of cut patterns in 1D gridded design. We first present an accurate integer linear programming (ILP) formulation that can solve the co-optimization of cut redistribution and mask assignment optimally. In addition, we propose an efficient graph-theoretic approach based on a novel integrated graph model and a longest-path-based refinement algorithm. Experimental results demonstrate that our graph-theoretic approach is orders of magnitude faster than the ILP-based method and meanwhile can obtain very comparable results. Comparing with the method that solves mask assignment and cut redistribution optimally but separately, our graph-theoretic approach that solves the two tasks simultaneously can achieve 95.0× smaller cost and 84.8× speedup on average.

1. INTRODUCTION

With the continuous scaling of transistor feature size, one dimensional (1D) gridded design (also known as unidirectional design) is widely believed to be a promising manufacturing solution for $10nm$ technology node and beyond [1–3]. The major advantages of 1D layouts over conventional two dimensional (2D) ones are lower design complexity and higher yield.

To fabricate a 1D layout, first some dense lines will be printed, and then cut masks will be used to trim off the unwanted parts. For example, given a target layout in Fig. 1(a), the 1D dense lines are first printed as shown in Fig. 1(b). Then some rectangular cuts, usually referred as *line-end cuts*, are applied to generate wires obeying the target layout (see Fig. 1(c)). The wires other than the target layout will have no electronic functionality and are called *dummy wires*. The real wires in Fig. 1(a) are located on different *tracks* and a space between two real wires on the same track is called a *gap*. Thanks to the uniformity, the dense lines are easy to print through a variety of lithography techniques, e.g., self-aligned double patterning (SADP). However, the manufacturing of cut patterns are very challenging, as two cuts that are too close to each other will result in a conflict or an error in manufacturing. For example, in Fig. 1(c), cut a conflicts with cut b while cut c conflicts with cut d.

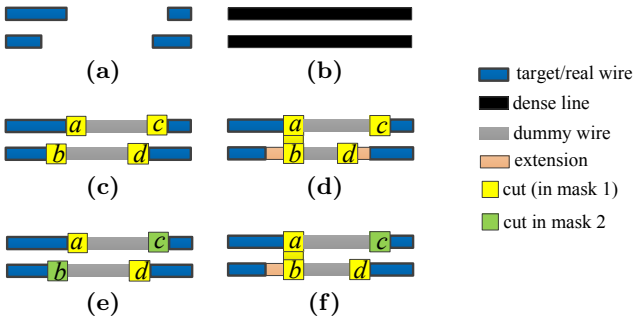

Figure 1: (a) A 1D target layout. (b) Dense lines. (c) Cuts and dummy wires. (d) The cuts are redistributed. (e) The cuts are assigned to different masks. (f) Incorporating cut redistribution with cut mask assignment.

To resolve the conflicts among cuts, two techniques are widely exploited: cut redistribution [4, 5] and mask assignment [6]. On one hand, an example of cut redistribution is illustrated in Fig. 1(d), where a conflict between two cuts can be resolved by either merging them together (e.g., cuts a and b) or locating them far away enough (e.g., cuts c and d). Note that through cut redistribution the wires will be extended, thus the timing may be affected. To limit such side effects along with wire extension, some additional constraints would be introduced, e.g., the wires on timing-critical nets are less flexible to be extended. On the other hand, through mask assignment, two conflicting cuts can be assigned to different masks as in multiple patterning lithography (MPL) [7], and manufactured by separate litho-etch processes (see Fig. 1(e)). Since relying solely on either approach may result in a large number of unresolved conflicts, in this paper we propose to incorporate cut redistribution with cut mask assignment to enable 1D grided design. Besides, we assume two cut masks are available, i.e., the mask assignment process is similar to the 2-coloring problem. An example of incorporating cut redistribution with mask assignment is shown in Fig. 1(f).

Even with cut redistribution and two cut masks, there may still be conflicts that cannot be resolved, especially in the presence of *native conflict*, i.e., a conflict that cannot be resolved by any cut redistribution nor 2-coloring. There are two ways to handle the unresolved conflicts. The first way is to minimize

*The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK14209214) and CUHK Direct Grant for Research.

ICCAD '16, November 07-10, 2016, Austin, TX, USA

 ISBN 978-1-4503-4466-1/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967048

and report the unresolved conflicts to designers for further layout modification [7]. The second way is to use complementary e-beam cuts [2], i.e., some of the cuts with unresolved conflicts will be manufactured by e-beam lithography [4, 5]. Due to the high resolution of e-beam lithography, we can assume that an e-beam cut will not conflict with any other cut, but the number of e-beam cuts should be minimized to improve throughput. In this paper, we assume the second way to handle unresolved conflicts.

There have been works studying the problem of cut redistribution for 1D design [4,5], where the problems are formulated as integer linear programming (ILP). (Note that in [5] another type of cut mask that directly removes the whole gap between wires is considered, but it will greatly increase the mask complexity.) There have also been works trying to redistribute the cuts to match with directed self-assembly (DSA) templates [8–10] or trying to incorporate MPL with DSA [11]. A recent work [12] also proposes ILP-based method to co-optimize cut mask, dummy fill and timing. However, as analyzed in Section 3, all these ILP formulations have some limitations.

In this paper, we study the problem of co-optimization of cut redistribution and mask assignment for 1D gridded design such that (i) no violation in design rules occurs; (ii) the number of e-beam cuts are minimized and (iii) the total wire extensions are minimized. We first present an accurate and optimal ILP formulation that overcomes the limitations of previous works. In addition, we propose a graph-theoretical approach based on a novel integrated graph model and a longest-path-based refinement algorithm to solve the problem efficiently and effectively.

The rest of the paper is organized as follows. Section 2 introduces some preliminaries and the problem formulation. Section 3 presents our accurate ILP formulation. Section 4 describes our graph-theoretic approach that can solve the co-optimization problem efficiently. Section 5 reports experimental results and Section 6 concludes the paper.

2. PROBLEM FORMULATION

The input to our problem is a set of n wires $\{w_1, ...w_n\}$ and $2n$ cuts $\{c_1, ...c_{2n}\}$. Variable e_i indicates whether c_i is printed using e-beam, where $1 \leq i \leq 2n$. If c_i is an e-beam cut, $e_i = 1$; otherwise, $e_i = 0$. The wires are labeled by 1 to n from the bottom to the top and from the left to the right in the layout. The tracks are labeled by 1 to the total number of tracks from the bottom to the top. We use $L(w_i)/R(w_i)$ to represent the left/right end of w_i. The cuts c_{2i-1} and c_{2i} are located at the two ends of w_i, i.e., $L(w_i)$ and $R(w_i)$, respectively. The width of a rectangular cut is W. l_i/r_i and x_{2i-1}/x_{2i} are used to represent the x-coordinates of $L(w_i)/R(w_i)$ in the input and the output, respectively. Note that in gridded design, these coordinates are discrete. Variable y_i is the label of the track on which w_i is located.

In this paper, we assume the following 1D gridded design rules as in previous works [5, 12]:

- **Rule 1**: There is an array $\mathcal{D} = \{d(0), d(1), \ldots, d(H)\}$ that defines the horizontal critical distances (i.e., safe distances) between cuts. $d(0)$ is the critical distance between two cuts located on the same track, and $d(1)$ is the critical distance between two cuts located on adjacent tracks (i.e. the difference between the labels of their tracks is 1), etc. The x-coordinate of a cut is the x-coordinate of its lower-left corner and the horizontal distance between two cuts is the difference between their x-coordinates. H is the largest difference between the track labels of two conflicting cuts. Note that such modeling of critical distance is general enough to handle critical distance measured in Euclidean distance.
- **Rule 2**: The wires can be extended but not shortened, i.e., $x_{2i-1} \leq l_i$ and $x_{2i} \geq r_i$. The total extension of a wire cannot exceed a limit for this wire, denoted as δ_i, i.e., $(x_{2i} - x_{2i-1}) - (r_i - l_i) \leq \delta_i$. Besides, the wires after extensions cannot exceed the boundaries of the layout.
- **Rule 3**: Two cuts assigned to the same mask are in conflict if (i) neither of them is an e-beam cut; (ii) they are within critical distance and (iii) they are not merged. Such a conflict is disallowed.
- **Rule 4**: Only the cuts on the same mask can be merged. There are three types of merging. The first type, as shown in Fig. 2(a), is that two cuts on the same track can be merged if they abut (e.g., cuts a and b in the figure) or overlap with each other (e.g., cuts c and d). The second type is that two cuts on adjacent tracks can be merged if they are aligned vertically (Fig. 2(b)). The third type is that two cuts on non-adjacent tracks can be merged if they are aligned vertically and they are both merged with the cuts located on the tracks in between. As a result, all these merged cuts should be vertically aligned (Fig. 2(c)).

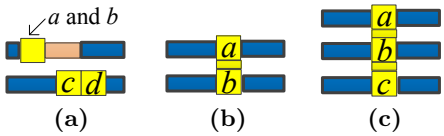

Figure 2: Different types of merging. (a) Merging on the same track. a is merged with b. c is merged with d. (b) Cuts a and b on adjacent tracks are merged. (c) Cuts a and c on non-adjacent tracks are merged. In such case, a, b and c should be vertically aligned.

We formally define the problem for co-optimization of cut redistribution and mask assignment as follows.

Problem 1. *Given a set of design rules and a layout of n wires and $2n$ cuts, decide the manufacturing method (using e-beam or not), the mask and the location of each cut, such that all the design rules are satisfied. The objective is to minimize*

$$\sum_{i=1}^{n}[(x_{2i} - x_{2i-1}) - (r_i - l_i)] + \alpha \sum_{i=1}^{2n} e_i, \tag{1}$$

where α is a variable to represent the relative importance between e-beam cuts and wire extensions (α is typically a large number).

3. AN ACCURATE ILP FORMULATION

There have been works [4, 5] solving the problem of single-mask cut redistribution for 1D design using ILP. As the ILP formulation in [5] can be solved much faster than that in [4], we will extend the formulation in [5] to simultaneously perform cut redistribution and mask assignment. Although the ILP formulation in [12] can also perform simultaneous cut redistribution and mask assignment, its limitations will be analyzed in Section 3.2.

3.1 Extensions for General Cut Redistribution

We first give some introductions to the ILP in [5]. Two gaps are called overlapping gaps if they overlap in horizontal direction. The objective of the ILP is to minimize Eq. (1). There are five sets of constraints: $\mathcal{C}1$. constraints for line end extensions (Rule 2); $\mathcal{C}2$. constraints for gaps between wires;

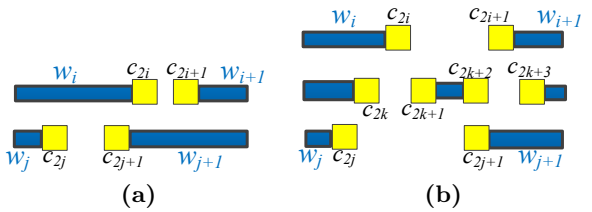

Figure 3: (a) Illustration for the constraint between non-overlapping gaps. (b) Illustration for the constraint between overlapping gaps on non-adjacent tracks.

$\mathcal{C}3$**.** constraints for non-overlapping gaps; $\mathcal{C}4$**.** constraints for overlapping gaps on adjacent tracks and $\mathcal{C}5$**.** constraints for overlapping gaps on non-adjacent tracks.

The ILP in [5] can be written as

$$\mathcal{ILCP1}: \text{min. Eq. (1)} \tag{2}$$
$$\text{s.t. } \mathcal{C}1 \sim \mathcal{C}5 \tag{2a}$$

However, the ILP in [5] has some limitations when handling $\mathcal{C}3$ and $\mathcal{C}5$, which will be analyzed in Appendix A. Note that these limitations exist for both single-mask and multiple-mask scenarios. In the following, we will extend the ILP in [5] to overcome these limitations and handle $\mathcal{C}3$ and $\mathcal{C}5$ correctly.

3.1.1 *The constraints for* $\mathcal{C}3$

As shown in Fig. 3(a), there are two non-overlapping gaps denoted as gap_i and gap_j, where gap_i is between the wires w_i and w_{i+1} and gap_j is between the wires w_j and w_{j+1}. W.l.o.g., we can assume gap_i is on the right of gap_j. We have the following constraints for $\mathcal{C}3$.

$$x_{2i} - (x_{2j+1} - W) + I(e_{2i} + e_{2j+1}) \geq d(|y_i - y_j|), \tag{3}$$
$$x_{2i} - x_{2j} + I(e_{2i} + e_{2j}) \geq d(|y_i - y_j|), \tag{4}$$
$$(x_{2i+1} - W) - x_{2j} + I(e_{2i+1} + e_{2j}) \geq d(|y_i - y_j|), \tag{5}$$
$$x_{2i+1} - x_{2j+1} + I(e_{2i+1} + e_{2j+1}) \geq d(|y_i - y_j|). \tag{6}$$

In the above equations, I means infinity. Regarding gap_i and gap_j, there are four pairs of possible conflicts between the ends of the wires and we need four equations: the conflict between $R(w_j)$ and $L(w_{i+1})$ is considered by Eq. (3), the conflict between $L(w_{j+1})$ and $L(w_{i+1})$ is considered by Eq. (4), the conflict between $L(w_{j+1})$ and $R(w_i)$ is considered by Eq. (5), and the conflict between $R(w_j)$ and $R(w_i)$ is considered by Eq. (6). In Eq. (3), x_{2j+1} is the x-coordinate of $L(w_j)$, and x_{2i} and $(x_{2j+1} - W)$ are the x-coordinates of c_{2i} and c_{2j+1}, respectively. If either of c_{2i} and c_{2j+1} is printed using e-beam, the constraint can be satisfied. Otherwise, the distance between $R(w_i)$ and $L(w_{j+1})$ must be larger than or equal to the corresponding critical distance. Eqs. (4)$\sim$(6) are similar.

3.1.2 *The constraints for* $\mathcal{C}5$

As shown in Fig. 3(b), the gap between w_i and w_{i+1} and the gap between w_j and w_{j+1} are two overlapping gaps. Again, there are four pairs of line ends that need to be considered regarding the constraint between the two gaps. For simplicity, we only describe the constraint between $R(w_i)$ and $R(w_j)$ as the others are similar. W.l.o.g., we assume $y_i - y_j = 2$. We have the following constraints for $\mathcal{C}5$.

$$x_{2i} - x_{2j} + I(e_{2i} + e_{2j} + d_{2j}^{2i} + m_{2j}^{2i}) \geq d(|y_i - y_j|), \tag{7}$$
$$x_{2j} - x_{2i} + I(e_{2i} + e_{2j} + 1 - d_{2j}^{2i} + m_{2j}^{2i}) \geq d(|y_i - y_j|), \tag{8}$$
$$x_{2i} - x_{2j} + I(1 - m_{2j}^{2i}) \geq 0, \tag{9}$$
$$x_{2i} - x_{2j} - I(1 - m_{2j}^{2i}) \leq 0, \tag{10}$$
$$m_{2j}^{2i} \leq m_{2k}^{2i} + m_{2k+1}^{2i} + m_{2k+2}^{2i} + m_{2k+3}^{2i}. \tag{11}$$

In the above equations, d_{2j}^{2i} and m_{2j}^{2i} are two binary variables. If c_{2i} is merged with c_{2j}, $m_{2j}^{2i} = 1$; otherwise, $m_{2j}^{2i} = 0$. It can be seen that if either c_{2i} or c_{2j} is printed using e-beam or the two cuts are merged, the constraints can be satisfied. Otherwise, either Eq. (7) or Eq. (8) will be activated depending on the value of d_{2j}^{2i} to ensure that the distance between c_{2i} and c_{2j} is at least the corresponding critical distance.

If the two cuts are merged, they must be vertically aligned. Eqs. (9) and (10) are used to enforce this.

Besides, as required in Rule 4, c_{2i} and c_{2j} can be merged only if they are both merged with a cut c_v located on the track between the tracks of c_{2i} and c_{2j}. In our example, there are four choices for c_v, namely c_{2k}, c_{2k+1}, c_{2k+2} and c_{2k+3}. Eq. (11) is used to make sure that c_{2i} and c_{2j} are merged with at least one of the four cuts.

Furthermore, to make sure that two cuts are merged only when neither of them is an e-beam cut, we add

$$1 - e_u \geq m_u^v \;\&\; 1 - e_v \geq m_u^v \tag{12}$$

for each variable m_u^v appeared in Eq. (11).

3.2 Extensions to Handle Simultaneous Cut Redistribution and Mask Assignment

To handle simultaneous cut redistribution and mask assignment, we add a binary variable s_i for each cut c_i to indicate the mask for c_i, and a binary variable f_i^j to indicate whether c_i is with a different color from c_j, where $f_i^j = s_i \oplus s_j$ (this can be linearized easily). To make sure that two cuts are merged only when they are in the same mask, we add

$$1 - f_i^j \geq m_i^j \tag{13}$$

for each variable m_i^j.

There is no conflict between two cuts in different masks. Thus we modify Eq. (3) as follows:

$$x_{2i} - (x_{2j+1} - W) + I(e_{2i} + e_{2j+1} + f_{2j+1}^{2i}) \geq d(|y_i - y_j|), \tag{14}$$

and similarly for other constraints.

Although the ILP formulation in [12] can also perform simultaneous cut redistribution and mask assignment, our ILP formulation has the following major advantages. First, the formulation in [12] does not differentiate overlapping gaps and non-overlapping gaps and thus may introduce some unnecessary variables and constraints, e.g., two cuts in non-overlapping gaps can never be merged but the formulation in [12] may also try to merge them. Second, the formulation in [12] only minimizes extensions. However, without incorporating the variables for unresolved conflicts or complementary e-beam cuts, an ILP may not have a solution. Third, the formulation in [12] does not have limits on the extensions of wires. Fourth, the formulation in [12] does not force the merged cuts to be in the same mask, thus, as shown Fig. 3(b), it may incorrectly align and merge c_{2i}, c_{2k} and c_{2j} even if c_{2i} and c_{2j} are in mask 1 while c_{2k} is in mask 2. Finally, the formulation in [12] also has the same problem for constraint $\mathcal{C}5$ as in [5], which will be discussed in Appendix A. By overcoming these limitations, the ILP in [12] can also be extended to our accurate ILP.

4. GRAPH-THEORETIC APPROACH

Although the accurate ILP formulation can solve Problem 1 optimally, the solving process is very time-consuming. As re-

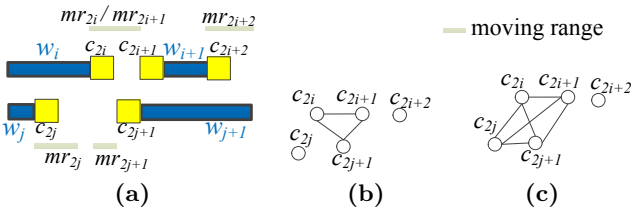

Figure 4: (a) The input layout and cuts. (b) The conflict graph G. (c) The potential conflict graph G_p.

ported in [5], the ILP formulation only performing cut redistribution takes about 13000 seconds to solve an M1 layout with 8000 tracks. After extending the formulation to Problem 1, the ILP solver must decide the mask for each cut. Thus, the running time grows exponentially with the number of cuts and may be much longer than that in [5]. In view of this, in this section we propose a novel graph-theoretic approach that can give a very comparable solution in a short time.

4.1 Potential Conflict Graph & Conflict Graph

Given a layout of wires and cuts, we can build a conflict graph G (throughout the paper, we use G to represent a conflict graph or a subgraph of a conflict graph), in which a node represents a cut and an edge between two nodes represents a conflict between the two corresponding cuts. For example, Fig. 4(b) shows the conflict graph for the cuts in Fig. 4(a). However, with cut redistribution, G is not static, meaning that the conflicts between the cuts can change dynamically. In view of this, we will build another potential conflict graph G_p before building G. G_p is similar to G, except that there is an edge between two nodes in G_p iff there is a potential conflict between the two corresponding cuts with cut redistribution. To construct G_p, we first find the possible moving range mr_i of each cut c_i. As shown in Fig. 4(a), the moving range (of the lower left corner) of c_{2i} is computed as follows. According to Rule 2, we have $r_i + \delta_i \geq x_{2i} \geq r_i$ and $x_{2i} \leq l_{i+1} - W$. For c_{2i+2}, we have an additional constraint that $x_{2i+2} \leq B_r$, where B_r is the x-coordinate of the right boundary of the given layout. The moving ranges of the other cuts can be calculated similarly. For any k and l, there is a potential conflict between c_k and c_l if c_k may conflict with c_l when they move within their moving ranges. For instance, the potential conflict graph for the cuts in Fig. 4(a) is shown in Fig. 4(c).

With G_p, we can safely split the graph into independent components and process those components separately. We can also find bridges and articulation nodes in the graph to further simplify it. Details of the graph simplification methods can be found in [13]. Applying these simplification methods on G_p is guaranteed to be safe because G_p will never change. For the same reason, we only need to build G_p once.

4.2 Overview

We use the graph simplification methods to split G_p into subgraphs, and then we will process the layouts corresponding to these subgraphs separately without losing any optimality. For each layout, we build a conflict graph G, and in the following, we mainly work on each conflict graph G. Our first task is to try to relocate the cuts to make G 2-colorable. As a result, e-beam cuts can be totally saved for this layout. If G cannot be made 2-colorable by relocating the cuts, our second task is to select some of the cuts to be printed by e-beam lithography such that the remaining subgraph is 2-colorable.

At first glance, our first task is similar to the problem of layout legalization for double patterning, i.e., modifying the layout to make the features 2-colorable. There have been some

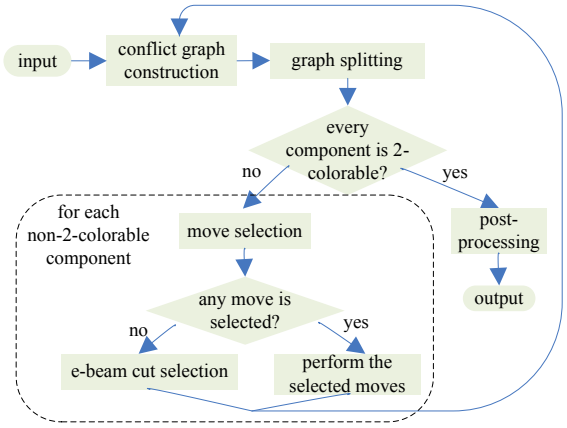

Figure 5: The flow of our graph-theoretic approach.

previous works [14–16] on this problem. In [14], a wire perturbation method is called iteratively as long as the odd cycles in the conflict graph are reduced. In [15], the conflict graph is first colored heuristically and then an LP is used to decide the locations of the features such that two features with the same color are far away enough. These methods are not applicable to our problem because of the unfixed horizontal order, the high density of the cuts, as well as the high complexity of the conflict graph in our problem. In [16], an ILP is used to decide the colors and locations of the features simultaneously, which will be very time-consuming. Besides, in the layout legalization problem, the features can only be spaced to resolve the conflicts, but in our problem the cuts can be merged. Furthermore, the existing approaches are for general 2D layouts and do not make use of the features of 1D design.

Our approach is an iterative method, whose flow can be found in Fig. 5. In each iteration, we will construct the conflict graph G, and split it into subgraphs (components) by finding independent components, bridges and articulation nodes. However, different from G_p, G is not static, and thus at the beginning of each iteration, we will repeat the conflict graph construction and splitting process. If every component is 2-colorable (which can be tested through depth-first search), we can combine these components, post-process and output the solution. Otherwise, for each non-2-colorable component, we try to select some *moves* to relocate the cuts so as to resolve some of the conflicts in that component. The definition of moves will be given later. If there are any selected moves, all of them will be performed. Otherwise, we will select some of the cuts to be printed using e-beam. After performing moves or selecting e-beam cuts for all the uncolorable components, we move on to the next iteration. The key steps of our approach, i.e., move selection, e-beam cut selection and post-processing, will be elaborated below.

4.3 Move Selection

We define the meaning of a "move" as follows. A move involves one or two cuts, and the moving directions and moving distances of these cuts. Formally, a move is $\{(c_i, \pm d_i)\}$ or $\{(c_i, \pm d_i), (c_j, \pm d_j)\}$, where c_i ad c_j are cuts, and d_i and d_j are discrete distances. We use "+" to represent moving rightwards and "−" to represent moving leftwards. The basic cost of a move is the total extensions it will cause. For example, given the cuts in Fig. 6(a) and the conflict graph in Fig. 6(b), we generate three moves as shown in Fig. 6(c), and the cost of each move is 1. At each iteration, we only allow moves to shift cuts away from their original positions to avoid moving a cut back and forth. In post-processing, a cut can be moved towards its original position to compensate for the loss

of quality.

4.3.1 Move Generation

Given a conflict edge in G, we will generate a set of moves that can resolve this conflict, under the limit on extensions. Basically, there are two types of moves that can resolve conflicts. The first type is to align two conflicting cuts or make one abut/overlap with another (so that they can be merged) and the second type is to space two conflicting cuts, e.g., among the moves in Fig. 6(c), m_2 can align b and c and m_1 can space a and b. After we generate the moves for each conflict edge separately, there may be duplicated moves because one move may solve the conflicts corresponding to multiple conflict edges at the same time, e.g., m_1 can solve the conflicts for edges (a, b) and (a, c) at the same time. Thus, after generating all the moves, we will detect and remove the duplicated ones.

Note that a move that tries to resolve one conflict may cause another new conflict, e.g., m_1 in Fig. 6(c) will cause a new conflict between a and e. If we simply forbid such moves, the cut redistribution process may get stuck because such moves may be the only possible moves for some conflicts and the newly caused conflict may be resolved by 2-coloring or moving other cuts further. Thus, our strategy is to allow such moves but increase their costs by β (which is set to 1 in our experiments) for each newly caused conflict to give preference to the moves that cause fewer new conflicts. If there are new conflicts, they will be solved in the next iteration.

4.3.2 Integrated Graph Model

In this section, we will introduce how to select moves based on an integrated graph model.

With the generated moves in the previous section, we will first build a **move constraint graph**, in which a node represents a move and an edge between two nodes means that the two corresponding moves are *incompatible*. Two moves are incompatible if (i) after applying both moves, the total extension of a wire will exceed the limit; or (ii) the two moves shift a cut in different ways; or (iii) applying both moves cannot resolve the conflicts that we intend to solve. For example, m_2 and m_3 in Fig. 6(c) are incompatible regarding the conflict between b and c because applying both of them cannot resolve the conflict. The move constraint graph for m_1, m_2 and m_3 is shown in Fig. 6(d).

Given a set $\mathcal{C}_f$ of conflict edges and a set $\mathcal{M}$ of moves, we will build a **bipartite graph** $\boldsymbol{B}_1$ between $\mathcal{C}_f$ and M as shown in Fig. 6(e). Each edge between a conflict and a move in B_1 means that the move can resolve the conflict.

A graph is 2-colorable iff it has no odd cycle. Thus, we need to select a set of edges from G to be resolved by the moves such that there is no odd cycle in the remaining subgraph. However, the problem of removing the minimum number of edges from a graph to break all the odd cycles is NP-Complete in general [17]. Enumerating all the odd cycles will also be time-consuming because the number of odd cycles grows exponentially with the number of nodes even in planar graphs [18]. Thus, in each iteration, we only resolve the odd cycles in a *cycle basis* of G. A cycle basis of a graph is a minimal set of cycles that can be combined to form every cycle in the graph using a sequence of *symmetric differences*[1] [14, 19].

A simple cycle basis of a connected graph G can be found as follows. Let T be a spanning tree of G, then each edge $\varepsilon \notin T$ combined with the path in T that connects the endpoints of ε forms a cycle in the cycle basis [19]. For instance, considering the graph G in Fig. 6(b), for the connected subgraph (without the isolated node e), a spanning tree $T = \{(c, a), (b, c), (c, d)\}$, and the only edge not in T is (a, b). Thus there is only one cycle $cl_0 = \{(c, a), (a, b), (b, c)\}$ in the cycle basis of G.

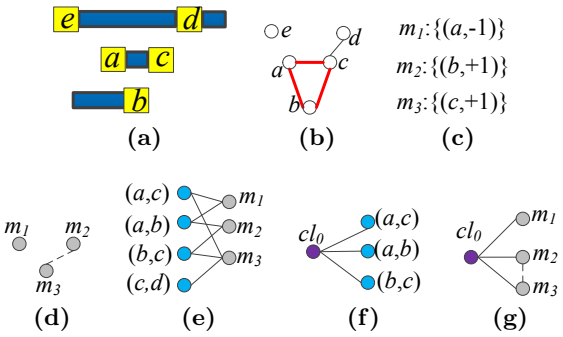

Figure 6: Illustration for move selection. Dash lines is incompatible edge between moves. cl_0 is the cycle of a, b and c. (a) Cuts. (b) Conflict graph G, and the odd cycle in its cycle basis is highlighted. (c) Moves. (d) **Move constraint graph**. (e) **Bipartite graph** $\boldsymbol{B}_1$ between $\mathcal{C}_f$ and $\mathcal{M}$. (f) **Bipartite graph** $\boldsymbol{B}_2$ between $\mathcal{C}_l$ and $\mathcal{C}_f$. (g) **The integrated graph model**.

It is easy to see that if there is no odd cycle in the cycle basis of G, then G has no odd cycles and is thus 2-colorable. On the other hand, if there is any odd cycle in the cycle basis of G, G is not 2-colorable. Therefore, in each iteration, we try to break the odd cycles in the cycle basis of G, which would be a good way to break all the odd cycles iteratively [2]. Our problem can be modeled as a **bipartite graph** $\boldsymbol{B}_2$ between the odd cycles in a cycle basis (denoted as $\mathcal{C}_l$) and the conflict edges $\mathcal{C}_f$ in these odd cycles. An example is shown in Fig. 6(f), where an edge between a cycle and a conflict edge means that resolving the conflict can break the odd cycle.

We then combine the graph models in Fig. 6(d)–6(f) together to get an **integrated graph** $\boldsymbol{G}_I$ as shown in Fig. 6(g). In Fig. 6(g), the incompitable edge between moves is the same as that in Fig. 6(d). There is an edge in G_I between $cl \in \mathcal{C}_l$ and $m \in \mathcal{M}$, iff $\exists cf \in \mathcal{C}_f$ such that an edge exists between cl and cf in B_2 and an edge exists between cf and m in B_1. The problem of move selection based on the integrated graph model can be formulated as follows.

Problem 2. *Given a graph $G_I(\mathcal{C}_l, \mathcal{M})$, select a minimum weight subset $\mathcal{M}_s$ of the nodes in $\mathcal{M}$, subject to the constraints between the nodes in $\mathcal{M}$, such that every node in $\mathcal{C}_l$ is connected to at least one node in $\mathcal{M}_s$.*

It can be seen that this problem is equivalent to the constrained set cover problem (CSCP)[3], which will be solved in the next section.

4.3.3 Solving the Constrained Set Cover Problem

The set cover problem, even without constraints, is NP-hard. To get high quality result, we formulate CSCP as an ILP and use an ILP solver to solve it optimally. The formulation

[1] The symmetric difference of two cycles is the set of edges which appear in either of the two cycles but do not appear in both of them.

[2] Notice that breaking all the odd cycles in a cycle basis of a graph may not break all the odd cycles of the graph, because the graph may have a different cycle basis after removing some edges. In our approach, the unresolved odd cycles in one iteration will be resolved in later iterations.

[3] In CSCP, $\mathcal{C}_l$ is the "universe" and $\mathcal{M}$ is the collection of "sets". The constraints are among the sets.

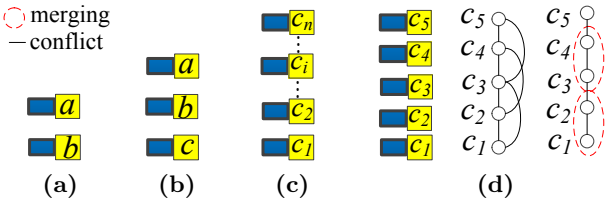

Figure 7: Vertically aligned Cuts.

is as follows:

$$\mathcal{ILP}2: \min. \quad \sum_{j=1}^{|\mathcal{M}|} b_j \cdot cost(m_j), \tag{15}$$

$$\text{s.t.} \quad \sum_{j=1}^{|\mathcal{M}|} a_{ij} \cdot b_j \geq 1, \quad \forall i \in \{1, \ldots, |\mathcal{C}_l|\}, \tag{15a}$$

$$b_i + b_j \leq 1, \ \forall c_i \text{ incompatible with } c_j, \tag{15b}$$

where a_{ij} is a binary number and b_j is a binary variable. $a_{ij} = 1$ iff there is an edge between cl_i and m_j in G_I. $b_j = 1$ iff m_j is selected.

By solving $\mathcal{ILP}2$, some moves will be selected. Note that $\mathcal{ILP}2$ can be solved much more efficiently than the ILP in Section 3 ($\mathcal{ILP}1$). There are three major reasons. First, we solve the problem iteratively as shown in Fig. 5 and in each iteration we split the conflict graph into smaller components, while such acceleration techniques cannot be applied to $\mathcal{ILP}1$. Second, the numbers of constraints are $\mathcal{O}(P)$ and $\mathcal{O}(|\mathcal{C}_l|)$ in $\mathcal{ILP}1$ and $\mathcal{ILP}2$, respectively, where P is the total number of pairs of gaps that may have conflicts. Generally speaking, $|\mathcal{C}_l| << P$. Third, the solution space of $\mathcal{ILP}1$ is very large because there are many different positions to place the cuts, while in $\mathcal{ILP}2$, a move can only be either selected or not, and the solution space is thus much smaller. The efficiency of our approach can be seen clearly from the experimental results in Section 5.

4.3.4 *Handling Vertically Aligned Cuts*

In this section, we discuss how to handle vertically aligned cuts specially when constructing the conflict graph.

For two vertically aligned cuts on adjacent tracks, such as a and b in Fig. 7(a), we will not add an edge between them when constructing the conflict graph. This is because, if a and b are finally colored differently, there is no conflict between them. On the other hand, if they are colored the same, they can always be merged and there is still no conflict.

If $H \geq 2$, for three or more consecutive and vertically aligned cuts, such as a, b and c in Fig. 7(b), the situation is more complicated. If a and c are colored the same, then b must be colored the same with a and c, as otherwise a and c cannot be merged and a conflict between them occurs. On the other hand, if a and c are colored differently, then the color of b will be the same as one of them. If we add an edge between a and c, it will force the solver to space them or to color them differently, which may result in sub-optimality. Actually, from the above analyses we can see that there is no need to add an edge between a and c as long as we can make sure that b is colored the same as at least one of a and c.

In general, consider n consecutive and vertically aligned cuts $\{c_1, ...c_n\}$ where $3 \leq n \leq H+1$, as shown in Fig. 7(c). (There is no need to consider $n > H+1$ as there will be no conflict between c_1 and c_n.) We have the following lemma , whose proof is shown in Appendix B.

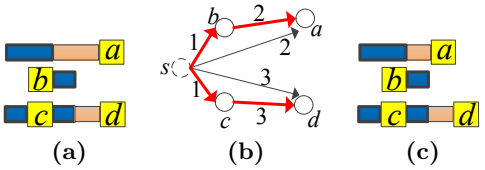

Figure 8: Illustration for the longest-path-based extension reduction method. (a) A cut redistribution and coloring solution, where the x-coordinates for a, b, c and d are 4, 1, 1 and 4, respectively. (b) The left-compaction graph and the longest paths. (c) The cut distribution after left-compaction.

Lemma 1. *There is no conflict among $c_1, ...c_n$ iff the following condition is satisfied: $\exists i$ where $2 \leq i \leq n$ such that $c_1, ...c_{i-1}$ are colored the same and $c_i, ...c_n$ are colored the same.*

According to Lemma 1, to make sure that some of the cuts are colored the same to avoid conflicts, we merge some of the nodes into one node when constructing the conflict graph. Consider constructing the conflict graph for m vertically aligned cuts. If $2 < m \leq 2H$, we will merge the nodes corresponding to the bottom $\lceil \frac{m}{2} \rceil$ cuts, and the nodes corresponding to the upper $m - \lceil \frac{m}{2} \rceil$ cuts; if $m > 2H$, we will merge the nodes in groups of H. This will make sure that among the m cuts, for any n consecutive cuts, where $3 \leq n \leq H+1$, the condition in Lemma 1 is satisfied. An example of $m = 5$ and $H = 2$ is shown in Fig. 7(d), and the merging will make sure that for any n consecutive cuts, where $3 \leq n \leq H+1$, the condition in Lemma 1 is satisfied and there is no conflict.

Note that there are many different ways to merge the nodes. Experimental results show that our merging strategy is effective to resolve the conflicts among vertically aligned cuts.

4.4 E-beam Cut Selection

For some of the components, there may not be available moves to make them 2-colorable, especially when there is native conflict. In this case, we will select some of the cuts to be printed using e-beam lithography. This operation is equivalent to deleting some nodes from the conflict graph G. We want to delete a minimum set of nodes from G such that at least one node is deleted from each odd cycle in a cycle basis of G. The problem is again formulated as a set cover problem and an ILP. The formulation is similar to Eq. (15) but without the incompatible constraints.

4.5 Post-processing

In this section, we present two post-processing methods to reduce the extensions of the wires.

4.5.1 *Longest-path-based Global Extension Reduction*

The first method to reduce wire extensions is a longest-path-based method. Given a solution of the locations and masks of the cuts, inspired by compaction in floorplanning [20], we want to compact the cuts at the right (left) ends of the wires to the left (right) as much as possible, subject to the spacing constraints between the cuts in the same mask, so as to reduce the total extensions.

For example, as shown in Fig. 8(a), where we only consider the cuts in mask 1, we construct a left-compaction graph as follows (see Fig. 8(b), where we assume that cuts on non-adjacent tracks have no conflicts). There is a node i for each cut c_i and a dummy source node s. There is a directed edge from s to each node i. If c_i is merged with some other cuts or c_i is at the left end of some wire, the cost of the edge between

Table 1: Result Comparisons

data	Single Mask [5]*			ILP Extended from [5, 12]				optCR(opt. coloring+opt. redistribution)				Our Graph-theoretic Method			
track#	eb#	ext	cost	eb#	ext	cost	time	eb#	ext	cost	time	eb#	ext	cost	time
50	14	53	14053	0	26	26	18.0	4	29	4029	0.7	0	26	26	0.1
100	24	106	24106	0	46	46	180.4	9	50	9050	1.2	0	46	46	0.2
150	36	236	36236	0	78	78	6446.9	14	91	14091	2.4	0	79	79	0.4
200	48	244	48244	OM	OM	OM	>36000	17	109	17109	2.8	0	104	104	0.5
250	59	324	59324	OM	OM	OM	>36000	19	135	19135	3.4	0	129	129	0.6
300	69	403	69403	OM	OM	OM	>36000	23	174	23174	4.7	0	164	164	0.7
1000	266	1560	267560	OM	OM	OM	>36000	69	670	69670	35.6	0	583	583	2.2
2000	515	3123	518123	OM	OM	OM	>36000	131	1380	132380	91.4	0	1230	1230	4.5
4000	1026	6447	1032447	OM	OM	OM	>36000	278	2764	280764	245.3	1	2500	3500	9.4
8000	2157	12924	2169924	OM	OM	OM	>36000	568	5740	573740	2784.9	1	5178	6178	18.8
Avg.	421.4	2542	423942	-	-	-	-	113.2	1114.2	114314.2	317.2	0.2	1003.9	1203.9	3.7
Ratio	2107.0	2.5	352.1	-	-	-	-	566.0	1.1	95.0	84.8	1	1	1	1

*The result of Single Mask is reported for reference. opt.=optimal

s and i is the x-coordinate of c_i to avoid moving it in left-compaction. Otherwise, the cost is the x-coordinate of the leftmost point in the moving range of c_i. We add a directed edge e_{ij} from i to j if (i) there is an edge between c_i and c_j in the potential conflict graph G_p; (ii) c_i has the same color as c_j; and (iii) c_j is on the right of c_i, i.e., $x_j > x_i$. The cost of e_{ij} is the required distance between c_i and c_j. With the left-compaction graph, we calculate the longest path from s to each node i, $\forall$ c_i at the right end of some wire. The longest path length from s to i means the leftmost x-coordinate that we can place c_i without any design rule violations, e.g., as shown in Fig. 8(b), the longest paths are highlighted and the longest path lengths from s to a, c and d are 3, 1 and 4, respectively. Thus, a can be moved leftwards by 1 and the extension of the corresponding wire can thus be reduced (Fig. 8(c)). Similarly, we can build the left-compaction graph for the cuts in mask 2, and the right-compaction graphs to reduce the extensions at the left ends of wires.

4.5.2 *Local Extension Reduction*

The longest-path-based method can minimize the total extensions globally, but it cannot change the colors of the cuts and the relative orders between the cuts that have potential conflicts. Thus, after calling the global extension reduction method, we will employ the following greedy extension reduction method that optimizes the extensions locally but is flexible to change the colors and orders of the cuts.

First, all e-beam cuts will be moved to their original positions as they will not cause any conflict. Then, $\forall$ c_i that is not printed using e-beam, if $x_i \neq x_i^o$ where x_i^o is the original x-coordinate of c_i, we consider the possible positions to place c_i, i.e., $\{x_i^o, x_i^o+1, ..x_i-1\}$ if c_i is at the right end of some wire, or $\{x_i^o, x_i^o-1, ..x_i+1\}$ if c_i is at the left end of some wire. At each position, we consider the two possible colors for c_i, and test whether c_i will conflict with c_j, $\forall$ c_j that has an edge with c_i in the potential conflict graph G_p. If there is no conflict, we will commit the position and coloring for c_i. Otherwise, assuming that c_i has a conflict with c_j, we will try to change the color of c_j to resolve the conflict so that we can commit the position and coloring for c_i to reduce wire extension. We consider two cases that the color of c_j can be changed. The first case is that c_i is the only cut that c_j has a conflict with. The second case is that c_i and c_j are in different independent components and the conflict edge between c_i and c_j will be a bridge between the two components, and thus we can change the colors of all the cuts in the component of c_j to resolve the conflict.

5. EXPERIMENTAL RESULTS

5.1 Experiment Setup

We implemented the proposed methods in C++, on a 2.39 GHz Linux machine with 16 CPU cores and 48 GB memory. `GUROBI` [21] is employed as our ILP solver. The benchmarks that we use are the M1 layouts used in [5] that are dense and thus need 2 masks. As in [5], α in Eq. (1) is set to 1000.

For the purpose of comparison, we implemented a method denoted as "optCR" that solves cut coloring and cut redistribution optimally but separately. There are two steps. The first step is to split a conflict graph into components and 2-color the cuts in each component optimally (using ILP) with the minimum number of unresolved conflicts, which will benefit the next step of cut redistribution. In the second step, the colors of the cuts will be imported to the ILP formulation in Section 3 such that the ILP solver only needs to decide the positions of the cuts to minimize Eq. (1).

5.2 Results

We compare the results of our graph-theoretic method, the accurate ILP extended from [5, 12], and the optCR method in Table 1. We have applied the techniques of constructing and simplifying potential conflict graph in Section 4.1 to speed up the accurate ILP-based approach. We also show the published results in [5] that uses a single mask for reference. (As different machine is used in [5], we do not show the runtime of [5].) In Table 1, "track#" means the number of tracks, "eb#" means the number of e-beam cuts, "ext" represents the total extensions of wires, "cost" is $ext + \alpha \cdot eb\#$ (i.e., Eq. (1)), "time" is the wall-clock time in seconds, and "OM" means that there is no solution because the program ran out of memory.

As shown in the table, with our graph-theoretic method, e-beam cuts can be totally saved for 8 out of 10 datasets. For the other 2 datasets, our method has also achieved the lower bound of "eb#", because there is a native conflict detected in each dataset and one native conflict requires at least one e-beam cut. Native conflict detection can be done as follows. Given two cuts and their moving ranges, it is easy to detect the case that no matter how the cuts move, they cannot be merged nor spaced. Given a set of cuts and their moving ranges, if there is an odd cycle in the conflict graph no matter how the cuts move, at least one native conflict exists. Thus we have achieved **the optimal result of** "**eb#**" for all the 10 datasets. For "ext", our results are very close to the optimal results reported by the ILP extended from [5, 12], for the datasets that can be solved by the ILP. The ILP cannot solve datasets with more than 150 tracks in a reasonable amount of time.

Comparing with the optCR method that solves coloring and redistribution optimally but separately, our method that solves the two tasks simultaneously can achieve 566.0× fewer ebeam cuts, 1.1× fewer extensions, and 95.0× smaller cost on average. For runtime, our method can solve the largest dataset within 19 seconds and is 84.8× faster than the optCR method on average, which clearly demonstrates our efficiency.

For reference, comparing with the results using a single cut mask, although using two masks will increase the mask cost, our method can obtain 2107.0× fewer ebeam cuts, 2.5× fewer extensions and 352.1× smaller cost on average, and thus the manufacturing cost can be reduced dramatically.

6. CONCLUSION

In this paper, we have proposed algorithms to co-optimize cut redistribution and mask assignment for 1D gridded design. Experiments showed that our graph-theoretic approach is very effective and efficient. As 1D gridded design is widely recognized as a promising solution to enable the scaling to $10nm$ technology node and beyond, we expect that this result can benefit the industry of circuit design and manufacturing and attract more research on the optimization for 1D gridded design.

7. REFERENCES

[1] 2013 international technology roadmap for semiconductors.
[2] M. C. Smayling. 1D design style implications for mask making and CEBL. In *Proc. SPIE*, 2013.
[3] L. Liebmann et al. The daunting complexity of scaling to 7nm without EUV: pushing DTCO to the extreme. In *Proc. SPIE*, 2015.
[4] Y. Du et al. Hybrid lithography optimization with e-beam and immersion processes for 16nm 1D gridded design. In *Proc. ASPDAC*, 2012.
[5] Y. Ding et al. Throughput optimization for SADP and e-beam based manufacturing of 1D layout. In *Proc. DAC*, 2014.
[6] W. Gillijns et al. Impact of a SADP flow on the design and process for N10/N7 Metal layers. In *Proc. SPIE*, 2015.
[7] A. B. Kahng et al. Layout decomposition for double patterning lithography. In *Proc. ICCAD*, 2008.
[8] J. Ou et al. Directed self-assembly cut mask assignment for unidirectional design. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 14(3), Jul. 2015.
[9] Z. Xiao et al. DSA template mask determination and cut redistribution for advanced 1D gridded design. In *Proc. SPIE*, 2013.
[10] Z.-W. Lin and Y.-W. Chang. Cut redistribution with directed self-assembly templates for advanced 1-D gridded layouts. In *Proc. ASPDAC*, 2016.
[11] Z.-W. Lin and Y.-W. Chang. Double-patterning aware DSA template guided cut redistribution for advanced 1-D gridded designs. In *Proc. ISPD*, 2016.
[12] K. Han et al. ILP-based co-optimization of cut mask layout, dummy fill and timing for sub-14nm BEOL technology. In *Proc. SPIE*, 2015.
[13] S.-Y. Fang et al. A novel layout decomposition algorithm for triple patterning lithography. In *Proc. DAC*, 2012.
[14] S.-Y. Fang et al. Native-conflict and stitch-aware wire perturbation for double patterning technology. *TCAD*, 31(5), May 2012.
[15] R. S. Ghaida et al. Layout decomposition and legalization for double-patterning technology. *TCAD*, 32(2), Feb. 2013.
[16] K. Yuan and D. Pan. WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography. In *Proc. ICCAD*, 2010.
[17] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Proc. STOC*, 1978.
[18] K. Buchin et al. On the number of cycles in planar graphs. In *Computing and Combinatorics*, 2007.
[19] J. T. Welch. A mechanical analysis of the cyclic structure of undirected linear graphs. *Journal of ACM*, 13(2), Apr. 1966.
[20] H. Murata et al. VLSI module placement based on rectangle-packing by the sequence-pair. *TCAD*, 15(12), Dec. 1996.
[21] Gurobi Optimizer. http://www.gurobi.com.

APPENDIX

A. LIMITATIONS OF EXISTING ILP

We first analyze the limitation for $\mathcal{C}3$. Consider the two gaps and the four pairs of line ends in Fig. 3(a). In [5], only the constraint in Eq. (3) is proposed for these two gaps and it is claimed this is sufficient. A natural thought is that as $L(w_{j+1})$ and $R(w_i)$ are the closest pair among all the four pairs of line ends, if this constraint is satisfied, the conflicts between other pairs of line ends should also be resolved. However, the problem for $\mathcal{C}3$ is that if either of c_{2i} and c_{2j+1} is printed using e-beam, the distance between $L(w_{j+1})$ and $R(w_i)$ may be smaller than the required critical distance, and so do the distances between the other three pairs of line ends.

Next we analyze the limitation for $\mathcal{C}5$. Consider the potential conflict between $R(w_i)$ and $R(w_j)$ in Fig. 3(b). The following constraints are proposed in [5].

$$\text{Eqs. (7)}\sim\text{(10)},$$
$$x_{2i} - x_{2k} + I(1 - m_{2j}^{2i}) \geq 0, \tag{16}$$
$$x_{2i} - x_{2k} - I(1 - m_{2j}^{2i}) \leq 0. \tag{17}$$

Eqs. (16) and (17) are used to make sure that c_{2i} and c_{2j} are aligned with a cut located on the track between the tracks of c_{2i} and c_{2j}, namely c_{2k}.

However, there are two problems with these constraints. The first problem is that there can be more than one cut that is possibly aligned with both c_{2i} and c_{2j}, e.g., as show in Fig. 3(b), there are four such cuts, namely c_{2k}, c_{2k+1}, c_{2k+2} and c_{2k+3}, but the above constraints only consider c_{2k}. A natural thought to solve the problem is to add the following constraints to make c_{2i} and c_{2j} vertically aligned with c_{2k+1}.

$$x_{2i} - (x_{2k+1} - W) + I(1 - m_{2j}^{2i}) \geq 0, \tag{18}$$
$$x_{2i} - (x_{2k+1} - W) - I(1 - m_{2j}^{2i}) \leq 0. \tag{19}$$

However, this does not work because adding these constraints will incorrectly force c_{2i} to align with c_{2k} and c_{2k+1} simultaneously. This problem also exists in [12]. The second problem is that c_{2k} can be an e-beam cut. In this case, c_{2i} and c_{2k} cannot be merged even if they are vertically aligned, and thus c_{2i} and c_{2j} cannot be merged through merging with c_{2k}.

B. PROOF OF LEMMA 1

PROOF. (the "if" part) Assume there exists such i. If $c_1, ...c_n$ are all colored the same, all of them can be merged together and there is thus no conflict. If $c_1, ...c_{i-1}$ are with one color and $c_i, ...c_n$ are with another color, there is no conflict between any cut in $\{c_1, ...c_{i-1}\}$ and any cut in $\{c_i, ...c_n\}$. Besides, $c_1, ...c_{i-1}$ can be merged together and $c_i, ...c_n$ can also be merged. Thus, there is no conflict.

(the "only if" part) Assume there does not exist such i. If c_1 and c_n are colored the same, there must exist $2 \leq i \leq n-1$ such that c_i is colored differently from c_1, which will result in a conflict because c_1 and c_n cannot be merged. If c_1 and c_n are colored differently, there must exist $2 \leq i < j \leq n-1$ such that c_i is colored the same as c_n while c_j is colored the same as c_1, which will result in a conflict between c_i and c_n and a conflict between c_j and c_1. □