CRMA: Incorporating Cut Redistribution With Mask Assignment to Enable the Fabrication of 1-D Gridded Design

Jian Kuang, Evangeline F. Y. Young, and Bei Yu, *Member, IEEE*

***Abstract*—1-D gridded design is one of the most promising solutions that can enable the scaling to 10 nm technology node and beyond. Line-end cuts are needed to fabricate 1-D layouts, where two techniques are available to resolve the conflicts between cuts: 1) cut redistribution and 2) cut mask assignment. In this paper, we consider incorporating the two techniques to enable the manufacturing of cut patterns in 1-D gridded design. We consider both 2-mask case (double patterning is performed on the cuts) and 3-mask case (triple patterning is performed on the cuts). 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 efficient graph-theoretic approaches based on a novel integrated graph model and a longest-path-based refinement algorithm. Experimental results demonstrate that our graph-theoretic approaches are orders of magnitude faster than the ILP-based method and meanwhile it can obtain very comparable results. For 2-mask case, 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 on average. We also extend our graph-theoretic approach to 3-mask case. Comparing with the method that reduces the 3-mask problem to 2-mask problem and solves it indirectly, our innovative approach that solves the problem directly based on a novel framework of identifying and solving 4-cliques can achieve 7.6% smaller cost on average.**

***Index Terms*—1-D design, design for manufacturability, lithography, mask assignment, multiple patterning.**

I. Introduction

ONE-DIMENSIONAL (1-D) gridded design (also known as unidirectional design) is widely believed to be a promising manufacturing solution for 10 nm technology node and beyond [2]–[4]. The major advantages of 1-D layouts over conventional 2-D ones are lower design complexity and higher yield.

Manuscript received May 21, 2017; revised August 10, 2017; accepted October 25, 2017. Date of publication November 28, 2017; date of current version September 18, 2018. This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project CUHK14209214. The preliminary version has been presented at the International Conference on Computer-Aided Design in 2016 [1]. This paper was recommended by Associate Editor I. H.-R. Jiang. *(Corresponding author: Jian Kuang.)*

The authors are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong (e-mail: jkuang@cse.cuhk.edu.hk; fyyoung@cse.cuhk.edu.hk; byu@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2778069

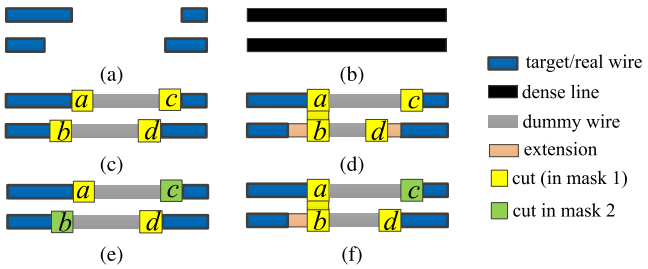

Fig. 1. (a) 1-D target layout. (b) Dense lines. (c) Cuts and dummy wires. (d) Cuts are redistributed. (e) Cuts are assigned to different masks. (f) Incorporating cut redistribution with cut mask assignment.

To fabricate a 1-D 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 1-D 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. However, the manufacturing of cut patterns is very challenging, as two cuts that are too close to each other will result in a conflict and an error in manufacturing. For example, in Fig. 1(c), cut *a* conflicts with cut *b* while cut *c* conflicts with cut *d*.

To resolve the conflicts among cuts, two techniques are widely exploited: 1) cut redistribution [5], [6] and 2) mask assignment [7]. 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) [8], [9], 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 1-D grided design. We consider two cases, namely 2-mask case (double patterning is performed on the cuts) and 3-mask case (triple patterning is performed on the cuts), i.e., the mask assignment process is similar to the 2-coloring or 3-coloring problem. An example of incorporating cut redistribution with mask assignment in 2-mask case is shown in Fig. 1(f).

Even with cut redistribution and multiple 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 coloring. There are two ways to handle the unresolved conflicts. The first way is to minimize and report the unresolved conflicts to designers for further layout modification [8]. The second way is to use complementary e-beam cuts [3], [10], i.e., some of the cuts with unresolved conflicts will be manufactured by e-beam lithography [5], [6]. 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. Note that using e-beam cut or not has very little impact on the contribution of this paper, as our approaches can be easily extended to handle unresolved conflicts in the first way above as well.

There have been works studying the problem of cut redistribution for 1-D design [5], [6], where the problems are formulated as integer linear programming (ILP). Note that in [6] another type of cut mask that directly removes the whole gap between wires is studied, but it may greatly increase the mask complexity. Besides, [11] formulates the problem as a network flow problem. However, the formulation in [11] is different from the others, which allows not only cut redistribution but also new cut insertion. There have also been works trying to redistribute the cuts to match with directed self-assembly (DSA) templates [12]–[14] or trying to incorporate MPL with DSA [15]. A recent work [16] proposes ILP-based method to co-optimize cut mask, dummy fill, and timing. Note that all the above-mentioned ILP formulations have some limitations, as shown in Section III.

In this paper, we study the problem of co-optimization of cut redistribution and mask assignment (CRMA) for 1-D gridded design such that: 1) no violation in design rules occurs; 2) the number of e-beam cuts is minimized; and 3) 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 graph-theoretical approaches based on a novel integrated graph model and a longest-path-based refinement algorithm to solve the problem efficiently and effectively. We consider both 2-mask case and 3-mask case.

The rest of this paper is organized as follows. Section II introduces some preliminaries and the problem formulation. Section III presents our accurate ILP formulation. Section IV describes our graph-theoretic approach that can solve the co-optimization problem efficiently in 2-mask case. Section V presents the extension of the graph-theoretic approach to 3-mask case. Section VI proposes native conflict identification methods. Section VII reports experimental results and Section VIII concludes this paper.

TABLE I
IMPORTANT NOTATIONS AND THEIR MEANINGS

w_i	the i-th wire
$L(w_i)/R(w_i)$	left/right end of w_i
c_{2i-1}/c_{2i}	the cut at $L(w_i)/R(w_i)$
W	the width of a cut
l_i/r_i	x-coordinate of $L(w_i)/R(w_i)$ in the input
x_{2i-1}/x_{2i}	x-coordinate of $L(w_i)/R(w_i)$ in the output
G/G_p	conflict graph/potential conflict graph
$\mathcal{C}$	the set of cuts
$\mathcal{C}_l$	a set of odd cycles
$\mathcal{C}_f$	a set of conflict edges
$\mathcal{M}$	a set of moves

II. PROBLEM FORMULATION

The input to our problem is a set of n wires $\{w_1, \ldots w_n\}$ and $2n$ cuts $\{c_1, \ldots c_{2n}\} = \mathcal{C}$. 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. Frequently used notations and their meanings are summarized in Table I.

In this paper, we assume the following 1-D gridded design rules as in previous works [6], [16].

1) *Rule 1:* There is an array $\mathcal{D} = \{d(0), d(1), \ldots, d(H)\}$ that defines the horizontal critical distances (or called 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. All coordinates should be on grid. 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.
2) *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.
3) *Rule 3:* Two cuts assigned to the same mask are in conflict if: 1) neither of them is an e-beam cut; 2) they are within critical distance; and 3) they are not merged. Such a conflict is disallowed.
4) *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 c and d in the figure) or overlap with each other (e.g., cuts a and b).

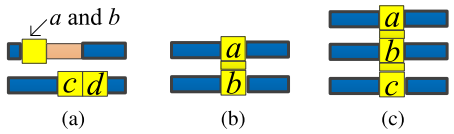

Fig. 2. Different types of merging. (a) Merging on the same track. a is merged with b. c is merged with d. (b) Cuts a and b on adjacent tracks are merged. (c) Cuts a and c on nonadjacent tracks are merged. In such case, a, b, and c should be vertically aligned.

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 nonadjacent 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)].

We formally define the problem for co-optimization of CRMA as follows.

Problem 1 (CRMA): Given a set of design rules, K masks, 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

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

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

In this paper, we consider the cases of $K=2$ and $K=3$. We denote Problem 1 as CRMA_2 when $K=2$ and CRMA_3 when $K=3$.

III. Accurate ILP Formulation

There have been works [5], [6] using ILP to solve the problem of single-mask cut redistribution for 1-D design exactly. As the ILP formulation in [6] can be solved much faster than that in [5], we will extend the formulation in [6] to simultaneously perform CRMA. Although the ILP formulation in [16] can also perform simultaneous CRMA, its limitations will be analyzed in Section III-B.

A. Extensions for General Cut Redistribution

We first give some introductions to the ILP in [6]. Two gaps are called overlapping gaps if they overlap in horizontal direction. The objective of the ILP is to minimize (1). There are five sets of constraints: $\mathcal{C}1$. constraints for line end extensions; $\mathcal{C}2$. constraints for gaps between wires; $\mathcal{C}3$. constraints for nonoverlapping gaps; $\mathcal{C}4$. constraints for overlapping gaps on adjacent tracks; and $\mathcal{C}5$. constraints for overlapping gaps on nonadjacent tracks.

Now, we explain how these constraints satisfy the design rules Rules 1–4. Rule 1 can be satisfied by simply only allowing integral coordinates in ILP. $\mathcal{C}1$ is for Rule 2. $\mathcal{C}2$–$\mathcal{C}5$ cover all situations about the relationship between two cuts, which thus make sure there is no conflict between any two cuts, obeying Rules 3 and 4.

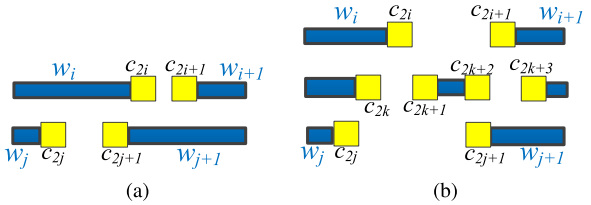

Fig. 3. Illustration for the constraint between (a) nonoverlapping gaps and (b) overlapping gaps on nonadjacent tracks.

The ILP in [6] can be written as

$$\mathcal{ILP}\mathbf{1}: \ \min \quad (1) \tag{2a}$$
$$\text{s.t.} \quad \mathcal{C}1-\mathcal{C}5. \tag{2b}$$

However, the ILP in [6] has some limitations when handling $\mathcal{C}3$ and $\mathcal{C}5$, which will be analyzed in the Appendix. Note that these limitations exist for both single-mask and multiple-mask scenarios. In the following, we will extend the ILP in [6] to overcome these limitations and handle $\mathcal{C}3$ and $\mathcal{C}5$ correctly. Details of the constraints of the original ILP can be found in [6], and are not repeated here due to space limitation.

1) Constraints for $\mathcal{C}3$: As shown in Fig. 3(a), there are two nonoverlapping 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}. Without loss of generality, we can assume gap_i is on the right of gap_j, as i and j are symmetric. We have the following constraints for $\mathcal{C}3$:

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

In the above equations, I represents a big enough number. Regarding gap_i and gap_j, there are four pairs of possible conflicts between the ends of the wires and we need four equations: 1) the conflict between $R(w_i)$ and $L(w_{j+1})$ is considered by (3); 2) the conflict between $R(w_i)$ and $R(w_j)$ is considered by (4); 3) the conflict between $L(w_{i+1})$ and $R(w_j)$ is considered by (5); and 4) the conflict between $L(w_{i+1})$ and $L(w_{j+1})$ is considered by (6). In (3), x_{2j+1} is the x-coordinate of $L(w_{j+1})$, 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. Equations (4)–(6) are similar.

2) 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. Without loss of generality, we assume $y_i-y_j=2$. We have the following constraints for $\mathcal{C}5$:

$$x_{2i}-x_{2j}+I\left(e_{2i}+e_{2j}+d_{2j}^{2i}+m_{2j}^{2i}\right)\geq d\left(|y_i-y_j|\right) \tag{7}$$
$$x_{2j}-x_{2i}+I\left(e_{2i}+e_{2j}+1-d_{2j}^{2i}+m_{2j}^{2i}\right)\geq d\left(|y_i-y_j|\right) \tag{8}$$

$$x_{2i} - x_{2j} + I\left(1 - m_{2j}^{2i}\right) \geq 0 \tag{9}$$

$$x_{2i} - x_{2j} - I\left(1 - m_{2j}^{2i}\right) \leq 0 \tag{10}$$

$$m_{2j}^{2i} \leq m_{2k}^{2i} + m_{2k+1}^{2i} + m_{2k+2}^{2i} + m_{2k+3}^{2i}. \tag{11}$$

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 (7) or (8) will be activated depending on the value of the auxiliary variable d_{2j}^{2i} to ensure that the distance between c_{2i} and c_{2j} is at least the corresponding critical distance, i.e., (7) will be activated if $d_{2j}^{2i} = 0$, while (8) will be activated if $d_{2j}^{2i} = 1$.

If the two cuts are merged, they must be vertically aligned. Equations (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}. Equation (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

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

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

B. Extensions to Handle Simultaneous Cut Redistribution and Mask Assignment

This section discusses how to handle simultaneous CRMA.

For 2-mask case (CRMA_2), 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

$$f_i^j = s_i \oplus s_j. \tag{13}$$

For 3-mask case (CRMA_3), we add two binary variable s_i^1 and s_i^2 for each cut c_i to indicate the mask for c_i : $s_i^1 = s_i^2 = 0$ means mask 1, $s_i^1 = 1$ and $s_i^2 = 0$ mean mask 2, and $s_i^1 = 0$ and $s_i^2 = 1$ mean mask 3. We thus need

$$s_i^1 + s_i^2 \leq 1 \tag{14}$$

for each i to limit the number of masks to 3. We still use a binary variable f_i^j to indicate whether c_i is with a different color from c_j, which is constrained by

$$f_{ij}^1 = s_i^1 \oplus s_j^1 \tag{15}$$

$$f_{ij}^2 = s_i^2 \oplus s_j^2 \tag{16}$$

$$f_i^j = f_{ij}^1 \vee f_{ij}^2 \tag{17}$$

where f_{ij}^1 and f_{ij}^2 are intermediate binary variables.

The above-mentioned notation "$\oplus$" means "XOR" and "$\vee$" means "OR". Both of them can be linearized easily for binary variables.

In both 2-mask and 3-mask cases, to make sure that two cuts are merged only when they are in the same mask, we add

$$1 - f_i^j \geq m_i^j \tag{18}$$

for each variable m_i^j.

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

$$x_{2i} - \left(x_{2j+1} - W\right) + I\left(e_{2i} + e_{2j+1} + f_{2j+1}^{2i}\right) \geq d\left(|y_i - y_j|\right) \tag{19}$$

and similarly for other constraints.

Although the ILP formulation in [16] can also perform simultaneous CRMA, our ILP formulation has the following major advantages. First, the formulation in [16] does not differentiate overlapping gaps and nonoverlapping gaps and thus may introduce some unnecessary variables and constraints, e.g., two cuts in nonoverlapping gaps can never be merged but the formulation in [16] may also try to merge them. Second, the formulation in [16] 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 [16] does not have limits on the extensions of wires. Fourth, the formulation in [16] does not force the merged cuts to be in the same mask, thus, as shown in 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 [16] also has the same problem for constraint $\mathcal{C}5$ as in [6], which will be discussed in the Appendix. By overcoming these limitations, the ILP in [16] can also be extended to our accurate ILP.

IV. Graph-Theoretic Approach for CRMA_2

Although the accurate ILP formulation can solve Problem 1 optimally, the solving process is very time-consuming. As reported in [6], the ILP formulation only performing cut redistribution takes about 13 000 s 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 [6]. In view of this, we propose novel graph-theoretic approaches that can give a very comparable solution in a short time. This section presents our approach for CRMA_2.

A. Potential Conflict Graph and Conflict Graph

Given a layout of wires and cuts, we can build a conflict graph G (throughout this paper, we use G to represent a conflict graph or a component 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

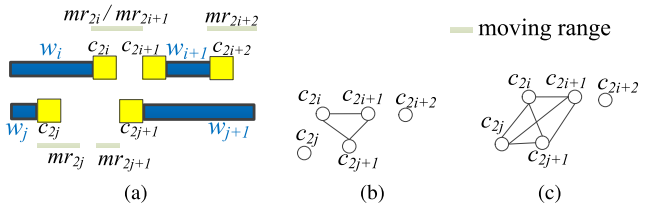

Fig. 4. (a) Input layout and cuts. (b) Conflict graph G. (c) Potential conflict graph G_p.

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 [17]. 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.

B. 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 previous works [18]–[20] on this problem. In [18], a wire perturbation method is called iteratively as long as the odd cycles in the conflict graph are reduced. In [19], 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 and the high density of the cuts, as well as the high complexity of the conflict graph in our problem. In [20], 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. Last but not least, the existing approaches are for general 2-D layouts and do not make use of the features of 1-D 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. Besides, we can recursively remove the nodes with degree

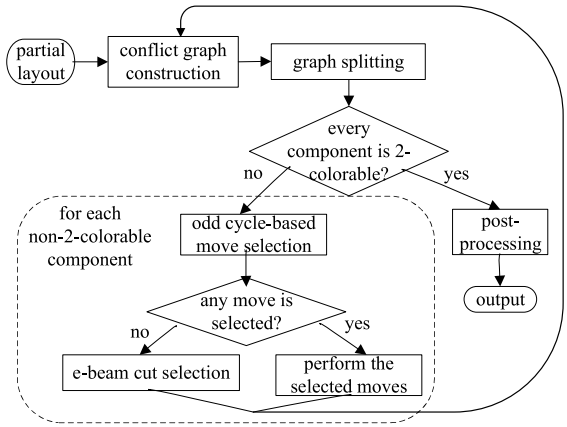

Fig. 5. Flow of our graph-theoretic approach for CRMA$_2$.

less than 2 temporarily as there is always at least one available color for these 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. The objective of move selection is to resolve all odd cycles. 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.

C. Odd Cycle-Based 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 and 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.

It is well-known that a graph is 2-colorable iff it contains no odd cycle. Thus, the key idea of odd cycle-based move selection is to select some moves to shift the cuts, so as to resolve odd cycles.

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 toward its original position to compensate for the loss of quality.

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

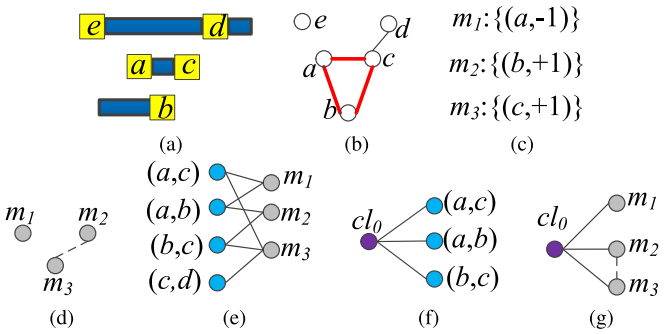

Fig. 6. Illustration for odd cycle-based move selection in CRMA$_2$. Dashed line is incompatible edge between moves. cl_0 is the cycle of a, b, and c. (a) Cuts. (b) Conflict graph G, and the odd cycle in its cycle basis is highlighted. (c) Moves. (d) Move constraint graph. (e) Bipartite graph $\boldsymbol{B}_1$ between $\mathcal{C}_f$ and $\mathcal{M}$. (f) Bipartite graph $\boldsymbol{B}_2$ between $\mathcal{C}_l$ and $\mathcal{C}_f$. (g) Integrated graph model.

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.

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: 1) after applying both moves, the total extension of a wire will exceed the limit; or 2) the two moves shift a cut in different ways; or 3) 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 $\mathcal{M}$ 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-hard in general [21]. 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 [22]. 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] [18], [23].

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 [23]. 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.

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)–(f) together to get an integrated graph $\boldsymbol{G}_I$ as shown in Fig. 6(g). In Fig. 6(g), the incompatible 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 $\mathcal{M}_s$.

In other words, Problem 2 is to select moves with the minimum cost, subject to the constraints, to resolve the odd cycles. It can be seen that this problem is equivalent to the following set cover problem but with constraints.

Problem 3 (Set Cover Problem): Given a set $\mathcal{U}$ of elements (the universe) and a collection $\mathcal{S}$ of sets whose union equals the universe and each set in $\mathcal{S}$ is with a cost, find the least-cost subcollection of $\mathcal{S}$ whose union equals the universe.

The equivalence is that, in Problem 2, $\mathcal{C}_l$ is the universe and $\mathcal{M}$ is the collection of sets, and there are additional constraints among the sets. Problem 2 is Problem 3 with constraints. We thus denote Problem 2 as the constrained set cover problem (CSCP), which will be solved in the next section.

Given a set of design rules, a cut can only conflict with a constant number of other cuts, and a move can also only

[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.

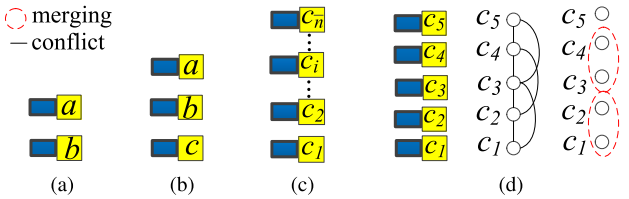

Fig. 7. Vertically aligned Cuts. (a) 2 cuts. (b) 3 cuts. (c) $\bar{n}$ cuts. (d) Example for merging vertically aligned cuts.

be incompatible with a constant number of other moves. Therefore, it is clear that $|\mathcal{C}_f|$, $|\mathcal{C}_l|$ and $|\mathcal{M}|$ are all linear to $|\mathcal{C}|$, and each step to build the integrated graph, i.e., building G_p, G, move constraint graph, B_1, B_2, and G_I, can be done in at most $\mathcal{O}(|\mathcal{C}| \cdot \log|\mathcal{C}|)$ time.

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 is as follows:

$$\mathcal{ILCP}2: \ \min \ \sum_{j=1}^{|\mathcal{M}|} b_j \cdot \text{cost}(m_j) \tag{20a}$$

$$\text{s.t.} \ \sum_{j=1}^{|\mathcal{M}|} a_{ij} \cdot b_j \geq 1, \quad \forall i \in \{1, \ldots, |\mathcal{C}_l|\} \tag{20b}$$

$$b_i + b_j \leq 1, \ \forall m_i \text{ incompatible with } m_j \tag{20c}$$

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 $\mathcal{ILCP}2$, some moves will be selected. Note that although the complexity is still exponential, $\mathcal{ILCP}2$ can be solved much more efficiently than the ILP in Section III ($\mathcal{ILCP}1$). 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 $\mathcal{ILCP}1$. Second, the numbers of constraints are $\mathcal{O}(P)$ and $\mathcal{O}(|\mathcal{C}_l|)$ in $\mathcal{ILCP}1$ and $\mathcal{ILCP}2$, 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 $\mathcal{ILCP}1$ is very large because there are many different positions to place the cuts, while in $\mathcal{ILCP}2$, 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 VII.

If $\mathcal{ILCP}2$ has no feasible solution, no move will be selected, and e-beam cut selection will be triggered (see Fig. 5).

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

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

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

($\Rightarrow$) Assume there does not exist such i. If c_1 and $c_{\bar{n}}$ are colored the same, there must exist $2 \leq i \leq \bar{n} - 1$ such that c_i is colored differently from c_1, which will result in a conflict because c_1 and $c_{\bar{n}}$ cannot be merged. If c_1 and $c_{\bar{n}}$ are colored differently, there must exist $2 \leq i < j \leq \bar{n} - 1$ such that c_i is colored the same as $c_{\bar{n}}$ while c_j is colored the same as c_1, which will result in a conflict between c_i and $c_{\bar{n}}$ and a conflict between c_j and c_1. ■

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 $\lceil (m/2) \rceil$ cuts, and the nodes corresponding to the upper $m - \lceil (m/2) \rceil$ cuts; if $m > 2H$, we will merge the nodes in groups of H. This will make sure that among the m cuts, for any $\bar{n}$ consecutive cuts, where $3 \leq \bar{n} \leq H + 1$, 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 $\bar{n}$ consecutive cuts, where $3 \leq \bar{n} \leq H + 1$, the condition in Lemma 1 is satisfied and there is no conflict.

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

D. 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

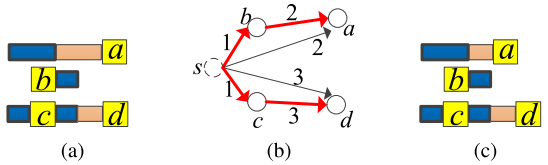

Fig. 8. Illustration for the longest-path-based extension reduction method. (a) Cut redistribution and coloring solution, where the x-coordinates for a, b, c, and d are 4, 1, 1, and 4, respectively. (b) Left-compaction graph and the longest paths. (c) Cut distribution after left-compaction.

and an ILP. The formulation is as follows:

$$\mathcal{ILP}3: \ \min \ \sum_{j=1}^{|\mathcal{C}|} e_j \tag{21a}$$

$$\text{s.t.} \ \sum_{j=1}^{|\mathcal{C}|} a_{ij} \cdot e_j \geq 1. \quad \forall i \in \{1, \ldots, |\mathcal{C}_l|\}. \tag{21b}$$

$\mathcal{ILP}3$ is similar to $\mathcal{ILP}2$ but without the incompatible constraints. We are working on a set of odd cycles $\mathcal{C}_l$ in a cycle basis of the conflict graph, and a set $\mathcal{C}$ of cuts that are candidates for e-beam cuts. e_j is a binary variable, which is 1 iff cut c_j is selected as e-beam cuts. a_{ij} is a binary number which is 1 iff c_j is a part of odd cycle cl_i. The objective is to minimize the number of e-beam cuts, and the constraints are to ensure every odd cycle has at least one cut been selected as e-beam cut.

E. Post-Processing

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

1) Longest-Path-Based Global Extension Reduction: The first method to reduce wire extensions is a longest-path-based method. The key idea is to compact the cuts to their original locations. Given a solution of the locations and masks of the cuts, inspired by compaction in floorplanning [24], 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 nonadjacent 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 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: 1) there is an edge between c_i and c_j in the potential conflict graph G_p; 2) c_i has the same color as c_j; and 3) 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 violation, 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.

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., $\{x_i^o, x_i^o + 1, \ldots, x_i - 1\}$ if c_i is at the right end of some wire, or $\{x_i^o, x_i^o - 1, \ldots, x_i + 1\}$ 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.

Obviously, both extension reduction methods above can complete in linear time, under a given set of design rules.

V. Graph-Theoretic Approach for CRMA$_3$

This section presents our graph-theoretic approaches for CRMA$_3$ that are extended from the approach for CRMA$_2$.

A. Baseline Method—Reducing CRMA$_3$ to CRMA$_2$

This section describes our baseline method to solve CRMA$_3$, which is used to be compared with the sophisticated method described in the next section.

A natural thought to solve CRMA$_3$ is to reduce it to CRMA$_2$. Given a conflict graph in CRMA$_3$, if an independent set of nodes are removed from it, the problem on remaining graph becomes CRMA$_2$. In other words, removing an independent set from a conflict graph can reduce a 3-coloring problem to a 2-coloring problem, because the nodes in the independent set can take the third color. Our baseline method is to find a maximum independent set (MIS) on the graph to make the remaining graph as small as possible. The flow is shown in Fig. 9. We first simplify the conflict graph and divide it into components similarly to Section IV-B. (The difference here is that instead of temporarily removing nodes with degree less than 2, we can temporarily remove nodes with degree less than 3 to simplify the graph.) And then, for each generated component, we find an MIS and remove the nodes in

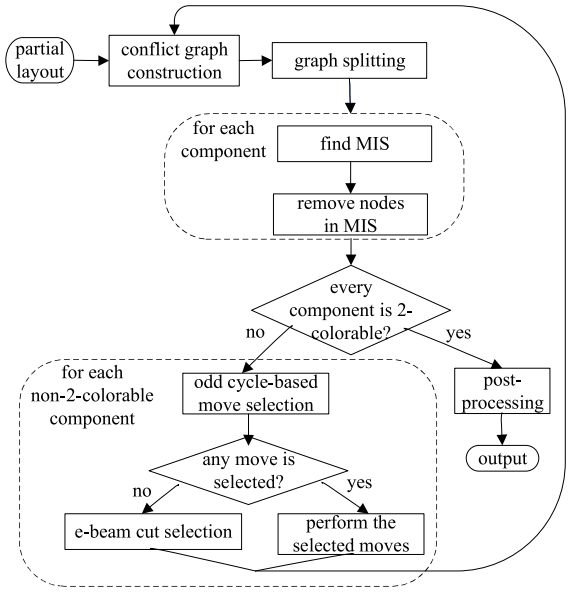

Fig. 9. Flow of the baseline method for $CRMA_3$ that reduces $CRMA_3$ to $CRMA_2$.

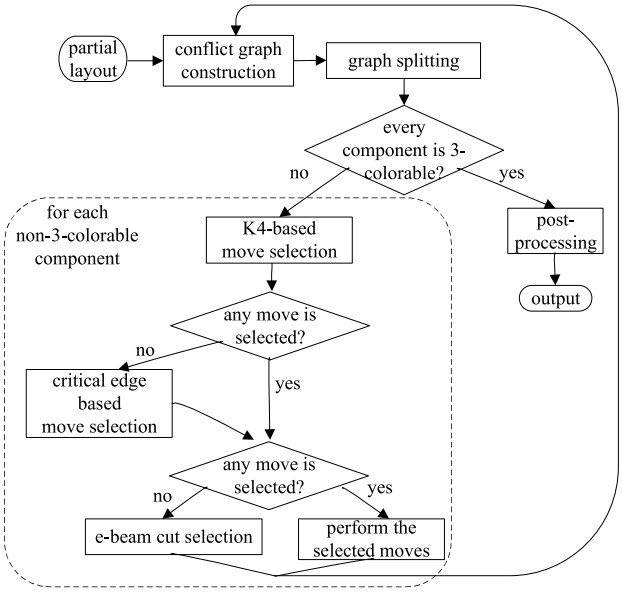

Fig. 10. Flow of our method K4solver for $CRMA_3$ that solves $CRMA_3$ directly.

Algorithm 1 MIS Finding

Input: A graph G, a number k_{init}.
Output: An MIS.

1: $k \leftarrow k_{init}$;
2: **while** kMis(G,k)=*nil* **do**
3: $k \leftarrow k/2$;
4: **end while**
5: $k_{max} \leftarrow$ `binarySearch`(k,$2k$) for the maximum k that has kMis(G,k)$\neq$*nil*;
6: return kMis(G,k_{max});

the MIS. We then solve $CRMA_2$ on the resulted components after removing the nodes in the MIS, and the remaining steps are same as the steps in Fig. 5.

It remains to describe how to find MIS in a graph (the step of "find MIS" in Fig. 9). Our method to find MIS is based on the algorithm in [25], which takes a graph and a parameter k, and returns, in polynomial time, an independent set of size at least k, or nil if such an independent set cannot be found. We denote the algorithm in [25] as kMis, and outline our MIS finding method in Algorithm 1. First, let k be a guess of the size of the MIS, k_{init}. If the kMIS algorithm cannot find a solution with size at least k, the algorithm is called again by halving k. This process continues until kMIS finds a solution with size at least k. Then the best solution must be between k and $2k$, which can be found by calling kMIS at most $\log(k)$ times by binary search. In our experiments, $k_{\text{init}} = (\#\text{Node in } G)/2$.

The kMIS algorithm takes at most $\mathcal{O}(|\mathcal{C}|^8)$ time to complete [25]. Thus, Algorithm 1 takes at most $\mathcal{O}(\log|\mathcal{C}| \cdot |\mathcal{C}|^8)$ time. Note that this is the worst case complexity. It is actually much faster in practice.

This MIS finding algorithm will be useful in the next section as well.

B. K4Solver: Solving $CRMA_3$ Directly

This section describes our method named *K4Solver* that can solve $CRMA_3$ directly.

K4Solver is extended from the approach in Section IV. The foundation of the approach in Section IV is odd cycle-based move selection, and the foundation of K4Solver is K4-based move selection. The key idea is, instead of resolving odd cycles in each iteration as shown in Fig. 5, we try to solve K4s (4-cliques) in each iteration. K4s are the key components to make a graph not 3-colorable, just as odd cycles are the key components to make a graph not 2-colorable. Containing at least one K4 is a sufficient condition for a graph to be non-3-colorable. Although it is not a necessary condition, resolving K4s can already make most graphs 3-colorable in practice.

There is a chance that some graphs are not 3-colorable even without K4s. For such graphs, we will first find *critical edges* in the graphs and then use moves to resolve the critical edges. Details of critical edge-based move selection will be elaborated later.

The flow of K4Solver is shown in Fig. 10. It is similar to the flow for $CRMA_2$ in Fig. 5 except for three key steps: 1) 3-colorability checking; 2) K4-based move selection; and 3) critical edge-based move selection. Critical edge-based move selection will be invoked only if K4-based move selection fails. Besides, e-beam selection in $CRMA_3$ is also slightly different from that in $CRMA_2$. These four steps will be elaborated in the following.

1) 3-Colorability Checking: In general, to determine the 3-colorability of a graph and to 3-color a graph are both NP-hard [9]. In K4Solver, We use a set of heuristics as shown in the flowchart in Fig. 11 to determine whether a component of a graph is 3-colorable or not, and color it if it is colorable.

The first heuristic we use is graph matching-based on the method in [26]. A graph library is first built which contains graphs that: 1) have no articulation nodes nor bridges; 2) have 4–6 nodes; and 3) have no nodes with degree less than 3.

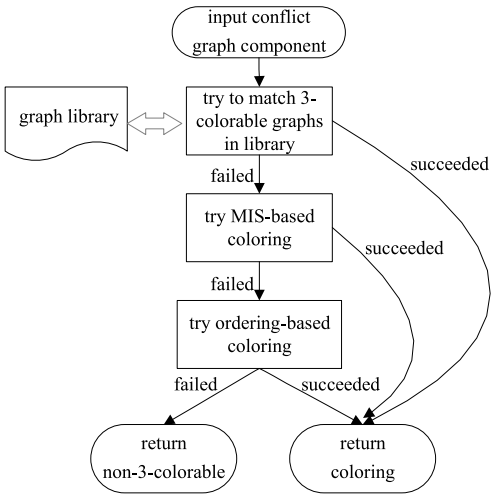

Fig. 11. Flow of 3-colorability checking.

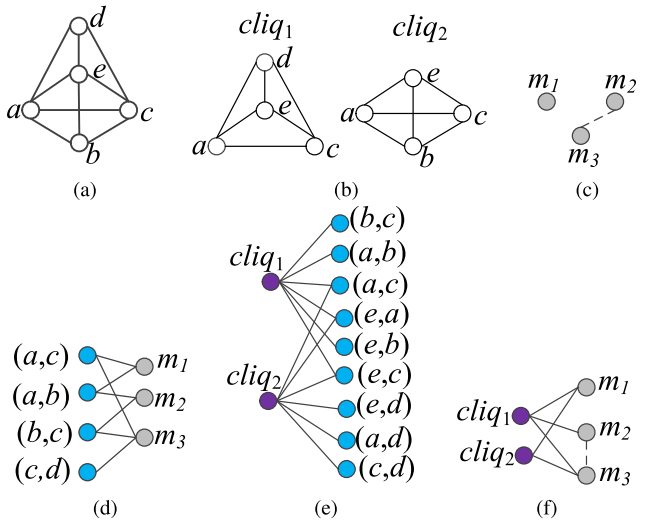

Fig. 12. Illustration for K4-based move selection in CRMA$_3$. Dashed line is incompatible edge between moves. (a) Conflict graph G. (b) Identified K4s $cliq_1$ and $cliq_2$. (c) Move constraint graph. (d) Bipartite graph $\boldsymbol{B}_1$ between $\mathcal{C}_f$ and $\mathcal{M}$. (e) Bipartite graph $\boldsymbol{B}_2$ between $\mathcal{C}_q$ and $\mathcal{C}_f$. (f) Integrated graph model. Note that (c) and (d) are assumed the same as those in Fig. 6 for simplicity.

These conditions can exclude useless graphs from the library and keep the size of the library reasonable. If the input graph component can match with a 3-colorable graph in the library, then it can be colored easily.

The graph matching method can only handle graph components with no more than 6 nodes, because it is impractical to include all of the graphs with 7 or more nodes into the library as otherwise there will be too many of them. We thus need different heuristics to handle graph components with 7 or more nodes. The second heuristic we use is MIS-based coloring, which first calls Algorithm 1 to find an MIS and then tests whether the remaining nodes are 2-colorable by depth-first search.

If both of the heuristics described above fail, we will try ordering-based coloring heuristics, i.e., color the nodes in a specific order. We consider the following orders based on [27].

1) *Largest Degree First (LDF):* Always first color the node with the largest degree.
2) *Largest Incident Degree First (LIDF):* Always first color the node with the largest number of colored neighbors.
3) *Largest Saturation Degree First (LSDF):* Always first color the node whose neighbors used the largest number of colors.

We also consider the following orders that mix the above ones.

1) *LIDF+LDF:* Always first color the node with the largest number of colored neighbors, and if multiple nodes have the same number, color the one with the LDF.
2) *LSDF+LDF:* Always first color the node whose neighbors used the largest number of colors, and if multiple nodes have the same number, color the one with the LDF.

As shown in Fig. 11, the above-described heuristics will be tried one by one, and if any of the heuristics succeed, 3-coloring of the graph component can be found. Otherwise, the graph component will be treated as not 3-colorable and move selection will be invoked as in the following sections.

The graph matching-based coloring can only handle graph with at most six nodes, which can be assumed to take constant time. The MIS-based coloring takes $\mathcal{O}(\log|\mathcal{C}| \cdot |\mathcal{C}|^8)$ time in the worst case as we have analyzed in Section V-A. The ordering-based coloring can finish in $\mathcal{O}(|\mathcal{C}|^2)$ time, as each cut can only conflict with a constant number of other cuts.

2) K4-Based Move Selection: In K4-based move selection, we first identify K4 subgraphs in the given conflict graph component and use moves to resolve the identified K4s. To identify all K4s, we enumerate all pairs of edges in the given graph component, and test whether the endpoints of the edges form a K4 (in constant time with the help of a look-up table). This can be done in $\mathcal{O}(|\mathcal{C}_f|^2) = \mathcal{O}(|\mathcal{C}|^2)$ time. For example, the conflict graph component in Fig. 12(a) has two K4s, $cliq_1$ and $cliq_2$, as shown in Fig. 12(b).

Then, we describe how to resolve the identified K4s by moves. Similar to Section IV-C2, we first build the move constraint graph, and the bipartite graph $\boldsymbol{B}_1$ between the set of conflict edges $\mathcal{C}_f$ and the set of moves $\mathcal{M}$ [see Fig. 12(c) and (d), respectively, where we assume the move constraint graph and the bipartite graph B_1 are the same as those in Fig. 6 for simplicity].

We then build bipartite graph $\boldsymbol{B}_2$ between the K4s (denoted as $\mathcal{C}_q$) and the conflict edges $\mathcal{C}_f$ in these K4s. An example is shown in Fig. 12(e), where an edge between a K4 and a conflict edge means that resolving the conflict can break the clique.

We then combine the graph models in Fig. 12(c)–(e) together to get an integrated graph $\boldsymbol{G}_I$ as shown in Fig. 12(f). In Fig. 12(f), the incompatible edge between moves is the same as that in Fig. 12(c). There is an edge in G_I between $cq \in \mathcal{C}_q$ and $m \in \mathcal{M}$, iff $\exists cf \in \mathcal{C}_f$ such that an edge exists between cq and cf in B_2 and an edge exists between cf and m in B_1. In other words, the move m can resolve the clique cq. Therefore, the problem of K4-based move selection can be reduced to Problem 2 (CSCP) on G_I and solved by $\mathcal{ILCP}_2$.

Similar to Section IV-C2, each step of building the integrated graph takes at most $\mathcal{O}(|\mathcal{C}| \cdot \log|\mathcal{C}|)$ time, except that finding all K4s needs $\mathcal{O}(|\mathcal{C}|^2)$ time as we have analyzed.

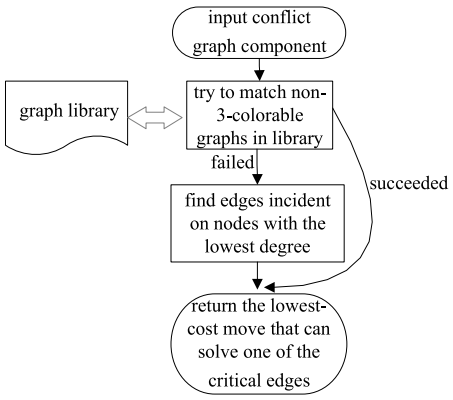

Fig. 13. Flow of critical edge-based move selection.

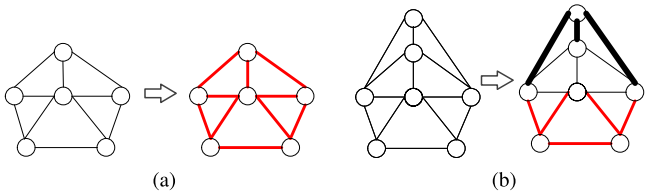

Fig. 14. Illustration for critical edges. Highlighted critical edges in a graph (a) in the library and (b) out of the library.

3) Critical Edge-Based Move Selection: In practice, there is still a small chance that K4-based move selection is unable to select any move because not all non-3-colorable graphs have a K4 as a subgraph. In such cases, we use critical edge-based move selection. Similar to [28], we define a critical edge as an edge in a non-3-colorable graph whose removal helps to make remaining subgraph 3-colorable.

As shown in Fig. 13, again, we make use of the graph library built in Section V-B1. But this time, we match the conflict graph component with non-3-colorable graphs in the library. When building the library, we can find and record the critical edges for each graph in the library that is not 3-colorable and has no K4s. Here, we only record the critical edges whose removal directly leave the remaining subgraph 3-colorable. Thus, if a graph component can match with a non-3-colorable graph in the library, we can get its critical edges immediately. An example is shown in Fig. 14(a), where all the highlighted edges are critical edges.

For graph components that cannot match any one in the library, we select the edges incident on the nodes with the lowest degree. The reason is, removing one of such edges has a higher chance to help the graph be simplified (by temporarily removing nodes with degree less than 3), and become 3-colorable after the simplification. For example, the graph in Fig. 14(b) has no K4 and cannot match any one in the library. The lowest degree in the graph is 3 and we select all edges incident on the nodes with degree 3 as critical edges (highlighted in the figure). In this particular example, removing any one of the critical edges highlighted as red will make the remaining subgraph 3-colorable.

With the selected critical edges, we build a bipartite graph B_1 between critical edges $\mathcal{C}_f$ and $\mathcal{M}$, similar to the one in Fig. 12(d). But this time our problem becomes to use the moves to resolve at least one of the critical edges. We thus select the move with the smallest cost, as shown in the flowchart in Fig. 13.

It is clear that the critical edge-based move selection can be done in $\mathcal{O}(|\mathcal{C}|)$ time.

4) E-Beam Cut Selection: It could happen that for some of the components, there are no available moves to make them 3-colorable. In such case, if K4s are identified in the component, we use an ILP to select the minimum number of cuts as e-beam cuts so that at least one cut is selected in each K4, i.e., the ILP is similar to $\mathcal{ILP}3$ but this time the constraint is for each K4 instead of each odd cycle. On the other hand, if K4s are not identified in the component, we select the cut with the highest degree as e-beam cut to help the coloring of the remaining cuts.

Fig. 15. Illustration for persistent conflict graph and native conflict. (a) Layout with one native conflict in CRMA_2. (b) persistent conflict graph G_r.

VI. Native Conflict Identification

In this section, we describe our methods to identify native conflicts, and the lower bound of the required e-beam cuts due to native conflicts. Native conflict identification has been studied in double patterning [18] and triple patterning [29], but the native conflict identification in CRMA is very different since the cuts are movable in CRMA through cut redistribution.

A native conflict in CRMA is a conflict that cannot be resolved by any cut redistribution nor coloring. Given two cuts c_1 and c_2 and their moving ranges, they can never be legally spaced if the largest distance between them within their respective moving ranges is smaller than the required critical distance. And they can never be merged if one of the following conditions is met: 1) their moving ranges do not horizontally overlap and 2) they are in nonadjacent tracks and it is not the case that a cut exists on every track between the tracks of c_1 and c_2 that can vertically align with both c_1 and c_2 (see Rule 4). For example, in Fig. 15(a), cuts a and c can never be legally spaced within the gaps they are located at. Cuts a and c can never be merged because they meet condition 1), and cuts a and b can never be merged because they meet condition 2). If a pair of cuts can never be legally spaced nor merged, we say that they have a persistent conflict.

We build a *persistent conflict graph* G_r, in which a node represents a cut and an edge between two nodes represents a persistent conflict between the two corresponding cuts. Note that G_r is static.[3] For CRMA_2, we can identify odd cycles in G_r. If there are k disjoint odd cycles, there are at least k native conflicts. This is because, any conflict in G_r cannot be resolved by cut redistribution, and any odd cycle in G_r cannot be resolved by coloring either. For example, the persistent conflict graph in Fig. 15(b) is built from the layout in Fig. 15(a), where there is one independent (i.e., disjoint

[3]In summary, G is conflict graph, G_p is potential conflict graph, and G_r is persistent conflict graph. Both G_r and G_p are static while G is dynamic and only represents the conflict at present.

TABLE II
RESULT COMPARISONS FOR $CRMA_2$

track#	Single Mask [6]*			ILP Extended from [6], [16]				optCR(opt. coloring+opt. redistribution)				Our Graph-theoretic Method				
	eb#	ext	cost	eb#	ext	cost	time	eb#	ext	cost	time	lb#	eb#	ext	cost	time
50	14	53	14053	0	26	26	18.0	4	29	4029	0.7	0	0	26	26	0.1
100	24	106	24106	0	46	46	180.4	9	50	9050	1.2	0	0	46	46	0.2
150	36	236	36236	0	78	78	6446.9	14	91	14091	2.4	0	0	79	79	0.4
200	48	244	48244	OM	OM	OM	>36000	17	109	17109	2.8	0	0	104	104	0.5
250	59	324	59324	OM	OM	OM	>36000	19	135	19135	3.4	0	0	129	129	0.6
300	69	403	69403	OM	OM	OM	>36000	23	174	23174	4.7	0	0	164	164	0.7
1000	266	1560	267560	OM	OM	OM	>36000	69	670	69670	35.6	0	0	583	583	2.2
2000	515	3123	518123	OM	OM	OM	>36000	131	1380	132380	91.4	0	0	1230	1230	4.5
4000	1026	6447	1032447	OM	OM	OM	>36000	278	2764	280764	245.3	1	1	2500	3500	9.4
8000	2157	12924	2169924	OM	OM	OM	>36000	568	5740	573740	2784.9	1	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

with others) odd cycle and thus at least one native conflict. Similarly, for $CRMA_3$, we can identify K4s in G_r. If there are k disjoint K4s, there are at least k native conflicts. Although our native conflict identification method cannot guarantee to find all native conflicts, it can find a reasonable lower bound for native conflicts. If there is at least one native conflict identified for a design, then using e-beam cut will be inevitable.

Then, we describe how to find a lower bound of e-beam cuts. For $CRMA_2$, we can apply $\mathcal{ILP}3$ on G_r, and the number of e-beam cuts in the solution will be the lower bound of e-beam cuts. For example, if we apply $\mathcal{ILP}3$ on the graph in Fig. 15(b), we can find that the lower bound of e-beam cuts is 1. For $CRMA_3$, we can apply ILP on G_r similarly, but the constraint in the ILP is for each K4 instead of each odd cycle. The lower bound of e-beam cuts will be useful to evaluate the difficulty of the benchmark and the quality of other optimization approaches.

VII. EXPERIMENTAL RESULTS

We implemented the proposed methods in C++, on a 2.39 GHz Linux machine with 16 CPU cores and 48 GB memory. GUROBI [30] is employed as our ILP solver. The benchmarks that we use are the M1 layouts used in [6] that are dense and thus need multiple masks. As in [6], α in (1) is set to 1000. [The value of α will not impact the results as long as it is big enough so that the cost of e-beam cuts dominates the cost of (1).]

A. Results for $CRMA_2$

For the purpose of comparison, we implemented a method denoted as "optCR" for $CRMA_2$ 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 III such that the ILP solver only needs to decide the positions of the cuts to minimize (1).

We compare the results of our graph-theoretic method, the accurate ILP extended from [6] and [16], and the optCR method in Table II. We have applied the techniques of constructing and simplifying potential conflict graph in Section IV-A to speed up the accurate ILP-based approach. We also show the published results in [6] that uses a single mask for reference. (The results of [11] also use a single mask but the formulation is different as it assumes new cut can be inserted.) As different machine is used in [6], we do not show the runtime of [6]. In Table II, "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., (1)], "time" is the wall-clock time in seconds, and "OM" means that there is no solution because the program ran out of memory.

TABLE III
RESULT COMPARISONS FOR $CRMA_3$

track#	ILP				Basedline Method				K4Solver				
	eb#	ext	cost	time	eb#	ext	cost	time	lb#	eb#	ext	cost	time
50	0	4	4	265.5	0	4	4	0.02	0	0	4	4	0.02
100	OM	OM	OM	>36000	0	6	6	0.03	0	0	6	6	0.03
150	OM	OM	OM	>36000	0	11	11	0.2	0	0	9	9	0.2
200	OM	OM	OM	>36000	0	12	12	0.2	0	0	10	10	0.2
250	OM	OM	OM	>36000	0	12	12	0.2	0	0	10	10	0.2
300	OM	OM	OM	>36000	0	16	16	0.2	0	0	13	13	0.2
1000	OM	OM	OM	>36000	0	53	53	0.4	0	0	49	49	0.5
2000	OM	OM	OM	>36000	0	104	104	1.9	0	0	96	96	1.8
4000	OM	OM	OM	>36000	0	228	228	6.2	0	0	213	213	5.9
8000	OM	OM	OM	>36000	0	483	483	11.5	0	0	448	448	11.9
Avg.	-	-	-	-	0.0	92.9	92.9	2.1	-	0.0	85.8	85.8	2.1
Ratio	-	-	-	-	1	1.08	1.08	1	-	1	1	1	1

As shown in the table, with our graph-theoretic method, e-beam cuts can be totally saved for 8 out of 10 datasets. For ext, our results are very close to the optimal results reported by the ILP extended from [6] and [16], 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 e-beam cuts, 1.1× fewer extensions, and 95.0× smaller cost on average. For runtime, our method can solve the largest dataset within 19 s 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 e-beam cuts, 2.5× fewer

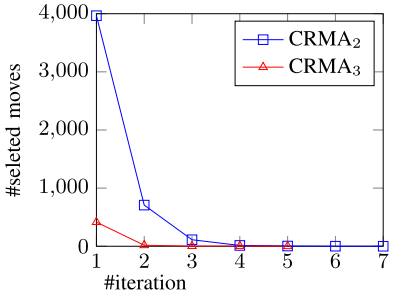

Fig. 16. Convergence of the number of selected moves for the benchmark with 8000 tracks.

extensions and 352.1× smaller cost on average, and thus the manufacturing cost can be reduced dramatically.

B. Results for CRMA3

The result comparisons for $CRMA_3$ among the accurate ILP extended from [6] and [16], the baseline method and K4solver are shown in Table III. It can be seen that e-beam cuts are totally saved for all the benchmarks with three masks. Comparing with the results of $CRMA_2$ in Table II, the wire extensions and total costs can be reduced dramatically as there is one more mask available.

As the complexity of ILP increases exponentially with the number of masks and the number of cuts, the accurate ILP can only solve the smallest case, for which the other two methods can both deliver the same result. Comparing with the baseline method that reduces $CRMA_3$ to $CRMA_2$ and solves it indirectly, the more sophisticated method, K4solver, can reduce the cost by 7.6% with similar runtime.

C. Convergence Analyses of the Graph-Theoretic Approaches

Our graph-theoretic approaches for both $CRMA_2$ and $CRMA_3$ are iterative methods. We analyze their convergence here on the benchmark with 8000 tracks. As can be seen from Fig. 16, our approaches converge very fast for both $CRMA_2$ and $CRMA_3$. Even for the largest benchmark, our approaches can finish in no more than seven iterations.

D. Identification of Native Conflicts and Lower Bounds of E-Beam Cuts

We employ the method described in Section VI to identify native conflicts and lower bounds of e-beam cuts in the used benchmarks for $CRMA_2$ and $CRMA_3$, respectively.

The columns "lb#" in Tables II and III show the results of lower bounds of "eb#" for $CRMA_2$ and $CRMA_3$, respectively. If lb# is nonzero, it means that native conflict is identified for the benchmark. It can be seen that our graph-theoretic methods have achieved the optimal result of eb# for all the ten benchmarks for both $CRMA_2$ and $CRMA_3$.

VIII. Conclusion

In this paper, we have proposed algorithms to co-optimize CRMA for 1-D gridded design. The experimental results showed that our graph-theoretic approaches for 2-mask case and 3-mask case are both very effective and efficient. As 1-D gridded design is widely recognized as a promising solution to enable the scaling to 10 nm 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 1-D gridded design.

Our future works include the following.

1) *Generalization of K4Solver:* We believe that the innovative framework of identifying and resolving K4s presented in K4Solver is not only applicable to the particular problem of $CRMA_3$ but also general enough to be extended to other coloring problems with three masks, e.g., layout decomposition in triple patterning lithograph [9], [17], [26]. One of our future works is to implement such extensions.
2) *Integration With Detail Router:* We will try to integrate the proposed cut optimization algorithms with detail router to improve the manufacturability of cut patterns starting from the routing stage.

Appendix
Limitations of Existing ILP

We First Analyze the Limitation for C3: Consider the two gaps and the four pairs of line ends in Fig. 3(a). In [6], only the constraint in (3) is proposed for these two gaps and it is claimed that 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 C5: Consider the potential conflict between $R(w_i)$ and $R(w_j)$ in Fig. 3(b). The following constraints are proposed in [6]:

$$(7)-(10)$$

$$x_{2i} - x_{2k} + I\left(1 - m_{2j}^{2i}\right) \geq 0 \tag{22}$$

$$x_{2i} - x_{2k} - I\left(1 - m_{2j}^{2i}\right) \leq 0. \tag{23}$$

Equations (22) and (23) 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 shown 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}:

$$x_{2i} - (x_{2k+1} - W) + I\left(1 - m_{2j}^{2i}\right) \geq 0 \tag{24}$$

$$x_{2i} - (x_{2k+1} - W) - I\left(1 - m_{2j}^{2i}\right) \leq 0. \tag{25}$$

However, this does not work either because adding these constraints will incorrectly force c_{2i} to align with c_{2k} and c_{2k+1} simultaneously. This problem also exists in [16]. 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}.

REFERENCES

[1] J. Kuang, E. F. Y. Young, and B. Yu, "Incorporating cut redistribution with mask assignment to enable 1D gridded design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Austin, TX, USA, Nov. 2016, pp. 1–8.

[2] *2013 International Technology Roadmap for Semiconductors*. Accessed: Nov. 12, 2017. [Online]. Available: http://www.itrs2.net

[3] M. C. Smayling, "1D design style implications for mask making and CEBL," in *Proc. SPIE Photomask Technol.*, vol. 8880. Monterey, CA, USA, Sep. 2013, Art. no. 888012.

[4] L. Liebmann, A. Chu, and P. Gutwin, "The daunting complexity of scaling to 7nm without EUV: Pushing DTCO to the extreme," in *Proc. SPIE Design Process Technol. Co Optim. Manufacturability IX*, vol. 9427. San Jose, CA, USA, Feb. 2015, Art. no. 942702.

[5] Y. Du, H. Zhang, M. D. F. Wong, and K.-Y. Chao, "Hybrid lithography optimization with e-beam and immersion processes for 16nm 1D gridded design," in *Proc. IEEE Asia South Pac. Design Autom. Conf.*, Sydney, NSW, Australia, Jan. 2012, pp. 707–712.

[6] Y. Ding, C. Chu, and W.-K. Mak, "Throughput optimization for SADP and e-beam based manufacturing of 1D layout," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[7] W. Gillijns *et al.*, "Impact of a SADP flow on the design and process for N10/N7 metal layers," in *Proc. SPIE Design Process Technol. Co Optim. Manufacturability IX*, vol. 9427. San Jose, CA, USA, Feb. 2015, Art. no. 942709.

[8] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 939–952, Jun. 2010.

[9] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 3, pp. 433–446, Mar. 2015.

[10] J. Kuang and E. F. Y. Young, "Optimization for multiple patterning lithography with cutting process and beyond," in *Proc. IEEE Design Autom. Test Europe Conf.*, Dresden, Germany, Mar. 2016, pp. 43–48.

[11] Y. Zhang *et al.*, "Network flow based cut redistribution and insertion for advanced 1D layout design," in *Proc. IEEE Asia South Pac. Design Autom. Conf.*, Chiba, Japan, 2017, pp. 360–365.

[12] J. Ou, B. Yu, J.-R. Gao, and D. Z. Pan, "Directed self-assembly cut mask assignment for unidirectional design," *J. Micro Nanolithography MEMS MOEMS*, vol. 14, no. 3, pp. 1–8, Jul. 2015.

[13] Z. Xiao, Y. Du, M. D. F. Wong, and H. Zhang, "DSA template mask determination and cut redistribution for advanced 1D gridded design," in *Proc. SPIE Photomask Technol.*, vol. 8880. Monterey, CA, USA, Sep. 2013, Art. no. 888017.

[14] Z.-W. Lin and Y.-W. Chang, "Cut redistribution with directed self-assembly templates for advanced 1-D gridded layouts," in *Proc. IEEE Asia South Pac. Design Autom. Conf.*, Macau, China, Jan. 2016, pp. 89–94.

[15] Z.-W. Lin and Y.-W. Chang, "Double-patterning aware DSA template guided cut redistribution for advanced 1-D gridded designs," in *Proc. ACM Int. Symp. Phys. Design*, Santa Rosa, CA, USA, Apr. 2016, pp. 47–54.

[16] K. Han, A. B. Kahng, H. Lee, and L. Wang, "ILP-based co-optimization of cut mask layout, dummy fill and timing for sub-14nm BEOL technology," in *Proc. SPIE Photomask Technol.*, vol. 9635. Monterey, CA, USA, Sep. 2015, Art. no. 96350E.

[17] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 3, pp. 397–408, Mar. 2014.

[18] S.-Y. Fang, S.-Y. Chen, and Y.-W. Chang, "Native-conflict and stitch-aware wire perturbation for double patterning technology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 5, pp. 703–716, May 2012.

[19] R. S. Ghaida *et al.*, "Layout decomposition and legalization for double-patterning technology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 2, pp. 202–215, Feb. 2013.

[20] K. Yuan and D. Z. Pan, "WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, Nov. 2010, pp. 32–38.

[21] M. Yannakakis, "Node-and edge-deletion NP-complete problems," in *Proc. ACM Symp. Theory Comput.*, San Diego, CA, USA, May 1978, pp. 253–264.

[22] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel, "On the number of cycles in planar graphs," in *Proc. Int. Conf. Comput. Combinatorics*, Banff, AB, Canada, Jul. 2007, pp. 97–107.

[23] J. T. Welch, "A mechanical analysis of the cyclic structure of undirected linear graphs," *J. ACM*, vol. 13, no. 2, pp. 205–210, Apr. 1966.

[24] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.

[25] A. Dharwadker. *The Independent Set Algorithm*. Accessed: Nov. 12, 2017. [Online]. Available: http://www.dharwadker.org/independent_set/

[26] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. ACM/IEEE Design Autom. Conf.*, Austin, TX, USA, Jun. 2013, pp. 1–6.

[27] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *Proc. ACM Symp. Parallelism Algorithms Archit.*, Prague, Czech Republic, Jun. 2014, pp. 166–177.

[28] J. Kuang, J. Ye, and E. F. Y. Young, "Simultaneous template optimization and mask assignment for DSA with multiple patterning," in *Proc. IEEE Asia South Pac. Design Autom. Conf.*, Macau, China, Jan. 2016, pp. 75–82.

[29] J. Kuang, W.-K. Chow, and E. F. Y. Young, "Triple patterning lithography aware optimization and detailed placement algorithms for standard cell-based designs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 4, pp. 1319–1332, Apr. 2016.

[30] *Gurobi Optimizer*. Accessed: Nov. 12, 2017. [Online]. Available: http://www.gurobi.com

Jian Kuang received the B.E. degree from Sun Yat-sen University, Guangzhou, China, in 2012 and the Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2016.

He is currently with Cadence Design Systems, San Jose, CA, USA. His current research interests include VLSI computer-aided design, physical design automation, and design for manufacturability.

Evangeline F. Y. Young received the B.Sc. and M.Phil. degrees in computer science from the Chinese University of Hong Kong (CUHK), Hong Kong, and the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 1999.

She is currently a Professor with the Department of Computer Science and Engineering, CUHK. She is actively researching on floorplanning, placement, routing, design for manufacturability, and algorithmic designs. Her current research interests include algorithms and computer-aided design of VLSI circuits.

Bei Yu (S'11–M'14) received the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu was a recipient of four Best Paper Awards at International Symposium on Physical Design 2017, the SPIE Advanced Lithography Conference 2016, the International Conference on Computer Aided Design 2013, and the Asia and South Pacific Design Automation Conference 2012, and four ICCAD/ISPD contest awards. He has served in the editorial boards of *Integration,* the *VLSI Journal* and *IET Cyber-Physical Systems: Theory & Applications*.