Journal of
Micro/Nanolithography,
MEMS, and MOEMS

Nanolithography.SPIEDigitalLibrary.org

Triple/quadruple patterning layout decomposition via linear programming and iterative rounding

Yibo Lin
Xiaoqing Xu
Bei Yu
Ross Baldick
David Z. Pan

SPIE.

Yibo Lin, Xiaoqing Xu, Bei Yu, Ross Baldick, David Z. Pan, "Triple/quadruple patterning layout decomposition via linear programming and iterative rounding," *J. Micro/Nanolith. MEMS MOEMS* **16**(2), 023507 (2017), doi: 10.1117/1.JMM.16.2.023507.

Triple/quadruple patterning layout decomposition via linear programming and iterative rounding

Yibo Lin,[a,*] Xiaoqing Xu,[a] Bei Yu,[b] Ross Baldick,[a] and David Z. Pan[a]
[a]University of Texas at Austin, Electrical and Computer Engineering Department, Austin, Texas, United States
[b]Chinese University of Hong Kong, Department of Computer Science and Engineering, New Territories, Hong Kong

Abstract. As the feature size of the semiconductor technology scales down to 10 nm and beyond, multiple patterning lithography (MPL) has become one of the most practical candidates for lithography, along with other emerging technologies, such as extreme ultraviolet lithography (EUVL), e-beam lithography (EBL), and directed self-assembly. Due to the delay of EUVL and EBL, triple and even quadruple patterning is considered to be used for lower metal and contact layers with tight pitches. In the process of MPL, layout decomposition is the key design stage, where a layout is split into various parts and each part is manufactured through a separate mask. For metal layers, stitching may be allowed to resolve conflicts, whereas it is forbidden for contact and via layers. We focus on the application of layout decomposition where stitching is not allowed, such as for contact and via layers. We propose a linear programming (LP) and iterative rounding solving technique to reduce the number of nonintegers in the LP relaxation problem. Experimental results show that the proposed algorithms can provide high quality decomposition solutions efficiently while introducing as few conflicts as possible. [DOI: 10.1117/1.JMM.16.2.023507]

Keywords: multiple patterning lithography; layout decomposition; linear programming.

Paper 17031P received Mar. 23, 2017; accepted for publication May 18, 2017; published online Jun. 14, 2017.

1 Introduction

Triple patterning lithography (TPL) and quadruple patterning lithography (QPL) are promising techniques to enhance lithography resolution when the feature size of semiconductor technology scales down to 10 nm and beyond. While it is true that there are other techniques, such as extreme ultraviolet lithography (EUVL), e-beam lithography (EBL), and directed self-assembly (DSA), the merits and demerits of these techniques result in various choices according to different applications.[1]

A typical process of TPL consists of litho-etch-litho-etch-litho-etch steps, which need three exposure/etching steps. In order to manufacture with TPL technology, it is necessary to split a single layer into three masks so that the features on each mask are far away enough to meet the resolution requirement of the optical system. This process is called "layout decomposition." Similarly for QPL, one layer is decomposed into four masks for manufacturing.

The condition for splitting features is usually related to the distance between them. For any feature pair, if two features are very close to each other, they should be split into different masks; otherwise, a "conflict" is introduced and it is not possible to manufacture them. In layout decomposition, each feature can be viewed as a vertex in a graph. If two features are too close to be manufactured in the same mask, a conflict edge is introduced to connect corresponding vertices. Vertices that share the same conflict edge must be assigned to different masks, or in other words, labeled with different colors. Then, the layout decomposition problem can be formulated into a graph coloring problem, as shown in Fig. 1. That is, TPL can be formulated to three-coloring and QPL can be formulated to four-coloring. The minimum distance to insert a conflict edge between two features is defined as "coloring distance."

Graph coloring is known as a nondeterministic polynomial time (NP) complete problem for a color number larger than two.[2] A layout may include subgraphs, such as four-clique (K4) structure that is not three-colorable. In Fig. 1(c), vertices a, b, c, and d form a K4 structure, so at least four masks are needed. The main objective of layout decomposition is to minimize the number of conflicts. For metal layers, stitching may be allowed to resolve conflicts; i.e., a feature can be split into two parts with different colors. But stitches are forbidden for contact and via layers. Even for metal layers, some fabs do not allow stitch insertion as it degrades the manufacturing yield. While layout decomposition is still different from traditional graph coloring, the NP completeness still holds.

Various algorithms have been proposed for the MPL layout decomposition problem, including integer linear programming (ILP), semidefinite programming (SDP), and other heuristic approaches.[3–10] Although ILP can solve the problem optimally, it suffers from an exponential runtime. SDP and other heuristic approaches are introduced to speedup the decomposition process with tradeoffs between runtime and solution quality. In order to improve the feature uniformity on each mask, density balance is also introduced as a secondary optimization target during decomposition.[11,12] Li et al.[13] formulate the TPL layout decomposition to 0-1 program and solve by the branch-and-bound method. Jiang and Chang[14] propose an exact set cover formulation to MPL layout decomposition with complex coloring rules. For row-based layout structures, even faster approaches have been proposed with the guarantee of optimality.[15–18] For a single standard cell, Yu et al.[19] proposed search-based methods to

*Address all correspondence to: Yibo Lin, E-mail: yibolin@utexas.edu

1932-5150/2017/$25.00 © 2017 SPIE

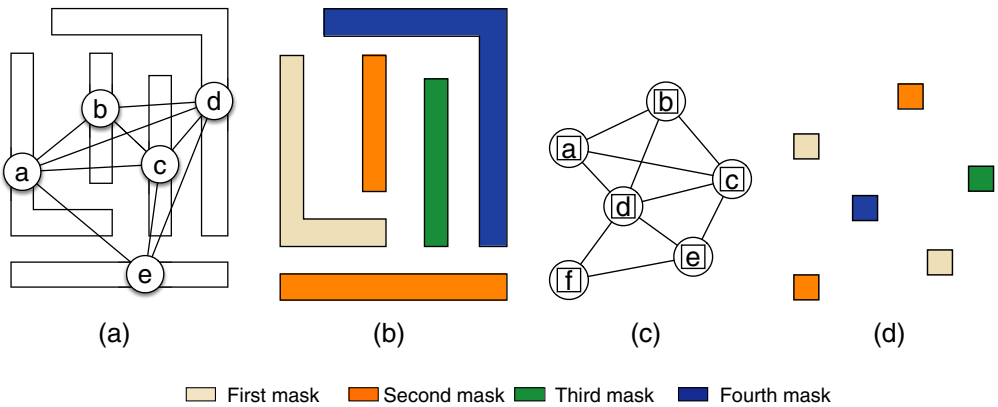

Fig. 1 Multiple patterning layout decomposition: (a) metal layer conflict graph, (b) corresponding decomposition, (c) contact layer conflict graph, and (d) corresponding decomposition.

enumerate all possible coloring solutions. Besides, Yu et al.[20] developed an incremental framework to accelerate conventional layout decomposition flow. Recently, Guo et al.[21] studied the lithography impact of different decomposition solutions based on lithography models.

In this paper, we focus on the application of layout decomposition where stitching is not allowed such as for contact and via layers. Given a layout with either rectangles or polygons where stitch insertion is forbidden, our goal is to provide high quality decomposition results efficiently while introducing as few conflicts as possible. Our major contributions are listed as follows:

1. We develop a linear programming (LP) and iterative rounding (IR) scheme to solve layout decomposition efficiently with high performance.
2. We propose an odd cycle-based technique to prune native noninteger solutions in the feasible set of LP, which can effectively reduce noninteger solutions in LP.
3. Experimental results show that our algorithm gets 3.3× speedup for TPL and 4.7× speedup for QPL compared with the state-of-the-art SDP-based layout decomposer with less 1% degradation in final conflict numbers.

The rest of the paper is organized as follows. In Sec. 2, we discuss the preliminary and problem formulation. In Sec. 3, we explain the algorithms, such as integer LP formulation, LP relaxation and our IR scheme. The experimental result is presented in Sec. 4 and we conclude this paper in Sec. 6.

2 Preliminaries and Problem Formulation

2.1 *Preliminaries*

Given a layout with rectangular or polygon shapes, we construct decomposition graph where each feature is denoted by a vertex in the graph. An edge is inserted if two corresponding features have a distance smaller than a minimum coloring distance. In the decomposition graph, if two vertices connected by a conflict edge are assigned to same color, a conflict is generated. The main target during layout decomposition is to minimize the number of conflicts. Figure 2(a) shows an example of conflict, where nodes c and d are assigned to the same mask, which results in a conflict highlighted by a red line.

In addition, the density of features at each mask should also be considered to reduce lithography hotspots and improve critical dimension uniformity.[12] Figures 2(b) and 2(c) compare two coloring solutions. The former one is unbalanced because most nodes are assigned to the first and second mask, and the third mask has only one node. The latter has more uniform distribution of nodes on each mask. Therefore, during the decomposition process, it is necessary to maintain density uniformity in each mask as well as the density ratios along different colors.

2.2 *Problem Formulation*

Given a layout with features of rectangular or polygon shapes, decomposition graph is constructed. The layout decomposition problem assigns features to different colors such that the total number of conflicts is minimized and

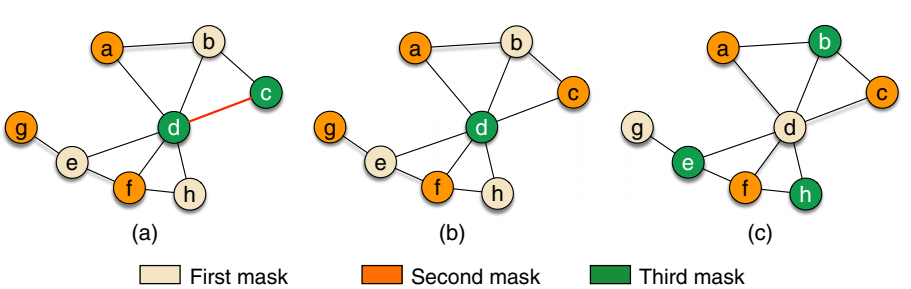

Fig. 2 Example of (a) conflict decomposition, (b) unbalanced decomposition, and (c) balanced decomposition in triple patterning layout decomposition.

meanwhile the coloring density at each mask is balanced. The coloring density balancing is defined as the difference between the most frequently used color and the least frequently used color.[3]

3 Algorithms

In this section, we will go over the overall flow of our algorithm. Then, we introduce our ILP formulation of layout decomposition, which is slightly different from conventional decomposition and explain its corresponding LP relaxation with the IR scheme in detail.

3.1 *Overall Flow*

The overall flow of our framework is shown in the left part of Fig. 3. It first constructs conflict graph from the layout and performs graph simplification, which will generate a set of simplified components. Each simplified component will be fed to kernel coloring algorithm based on linear programming and iterative rounding (LPIR). As some vertices are removed from the original graph to generate simplified components during graph simplification, it is necessary to recover them during postcolor assignment. In the end, the final coloring results are produced. The detailed flow for LPIR will be explained in Sec. 3.3.

3.2 *Integer Linear Programming Formulation*

Given a conflict graph $G(V, E)$, it is necessary to introduce two binary variables for each node to represent three/four colors in the TPL/QPL layout decomposition problem. The ILP formulation is shown in Eq. (1). For each conflict edge in the conflict edge set E_c, the situation of identical colors on both vertices is forbidden by constraints of Eqs. (3)–(6); e.g., the constraint of Eq. (3) requires that x_{i1}, x_{i2}, x_{j1}, and x_{j2} cannot be zero at the same time, which would otherwise result in conflict at edge e_{ij}, and so forth for the other constraints. The constraint of Eq. (2) is only used to eliminate the fourth color in TPL layout decomposition, so it is not needed in the QPL decomposition problem. Different from the ILP formulation in Ref. 3, additional stitch edge variables are not introduced since stitch insertion is not allowed. There are no additional conflict variables either because we handle the minimization of conflicts in the LP relaxation. Instead of minimizing the total cost from conflicts, the target of our ILP formulation is to seek a feasible color assignment to the variables while optimizing the changeable objective function.

$$\min \ \text{obj}, \tag{1}$$

$$\text{s.t. } x_{i1} + x_{i2} \le 1, \tag{2}$$

$$x_{i1} + x_{i2} + x_{j1} + x_{j2} \ge 1, \quad \forall\, e_{ij} \in E_c, \tag{3}$$

$$x_{i1} + \bar{x}_{i2} + x_{j1} + \bar{x}_{j2} \ge 1, \quad \forall\, e_{ij} \in E_c, \tag{4}$$

$$\bar{x}_{i1} + x_{i2} + \bar{x}_{j1} + x_{j2} \ge 1, \quad \forall\, e_{ij} \in E_c, \tag{5}$$

$$\bar{x}_{i1} + \bar{x}_{i2} + \bar{x}_{j1} + \bar{x}_{j2} \ge 1, \quad \forall\, e_{ij} \in E_c, \tag{6}$$

$$\bar{x}_{i1} = 1 - x_{i1}, \quad \forall\, i \in V, \tag{7}$$

$$\bar{x}_{i2} = 1 - x_{i2}, \quad \forall\, i \in V, \tag{8}$$

$$x_{i1}, \quad x_{i2} \in \{0,1\}, \quad \forall\, i \in V. \tag{9}$$

3.3 *Linear Programming and Iterative Rounding Flow*

Although the proposed ILP formulation is able to find optimal color assignment when there exists conflict free solution, it has an exponential runtime and is not capable of minimizing total conflicts if there is at least one conflict in the optimal solution due to the infeasibility. With LP relaxation of the problem, it is possible to avoid the infeasibility issue and find a solution with few conflicts. The relaxation from ILP to LP will result in noninteger solutions, so it is critical to find a proper rounding scheme for quality guarantee.

The LPIR flow for our coloring framework is demonstrated in the right part of Fig. 3. The framework starts

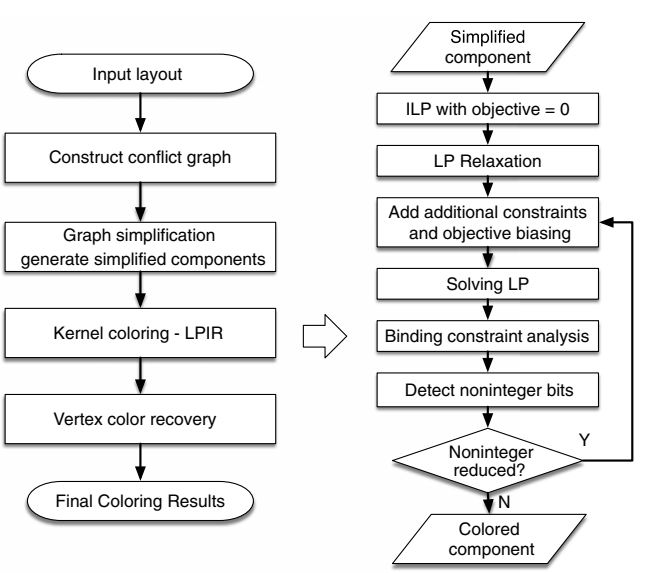

Fig. 3 Overall flow for our coloring framework.

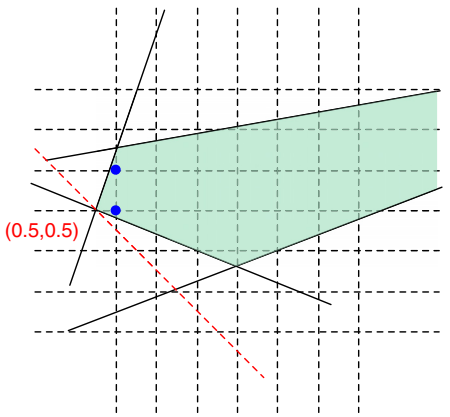

Fig. 4 The polyhedron for feasible LP solutions. The shaded region denotes the feasible space for LP. The dashed red line denotes the objective function with optimal value. The grids consisting of dashed black lines are possible solutions with integer bits.

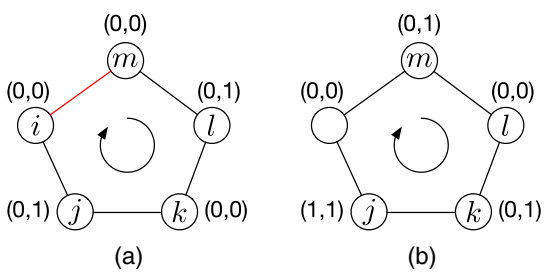

Fig. 5 One possible odd cycle in the conflict graph: (a) a coloring conflict from due to identical first bit values of nodes and (b) resolved conflict.

with the LP with an objective of zero. To deal with noninteger in the solution from LP, we identify some native noninteger solutions in the feasible set resulted from odd cycles. Then, additional constraints are introduced to prune these native noninteger solutions. During each iteration, objective function is changed from the original LP formulation to push noninteger to integers. In particular, these

Algorithm 1 Density Aware IVR Vertex Color Recovery.

Require: A stack S containing uncolored vertices.

Ensure: Assign colors with balanced density.

1. Define available color set C for TPL/QPL and available color set C_v for vertex v;
2. Define distance $d_{c,v}$ for vertex v as the distance with the closest vertex with color c;
3. Define best color bc_v for vertex v and bd_v as the corresponding distance;
4. **while** $S \neq \emptyset$ **do**
5. $v \leftarrow S.pop()$;
6. **for** $c \in C$ **do**
7. Compute $d_{c,v}$;
8. **end for**
9. Compute C_v for vertex v;
10. $bc_v \leftarrow -1$, $bd_v \leftarrow -\infty$;
11. **for** $c \in C_v$ **do**
12. **if** $d_{c,v} > bd_v$ **then**
13. $bd_v \leftarrow d_{c,v}$, $bc_v \leftarrow c$;
14. **end if**
15. **end for**
16. Assign color bc_v to vertex v;
17. **end while**

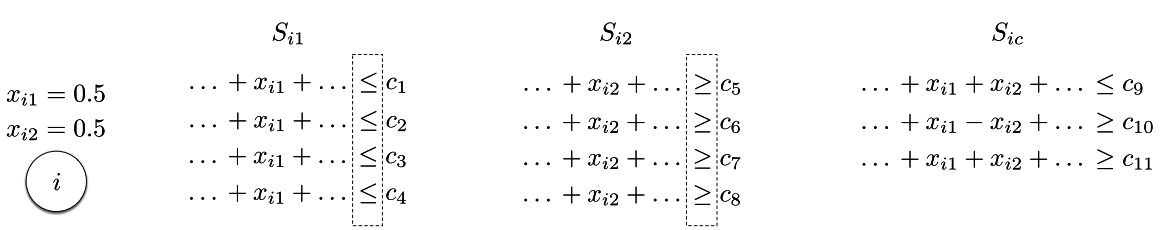

Fig. 6 An example of binding constraints analysis.

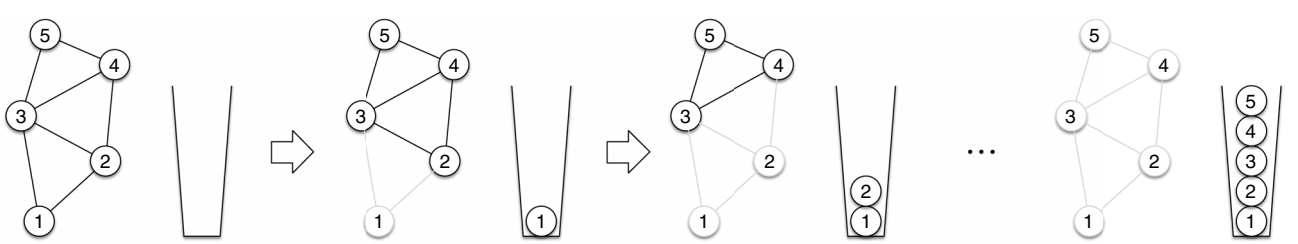

Fig. 7 An example of IVR in TPL.

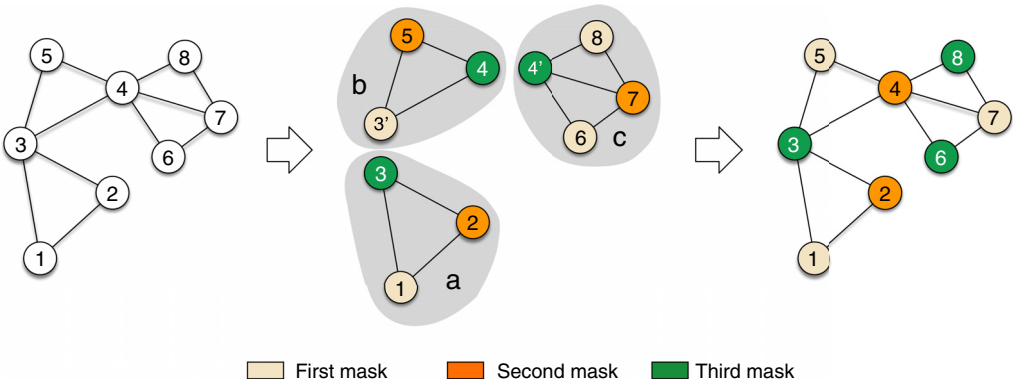

Fig. 8 An example of biconnected BCE and color recovery.

additional constraints and objective function change will not break the feasibility of possible coloring assignment. We continue the LPIR iterations until the number of nonintegers fail to reduce, and we round the solutions of variables to nearest integers. Then, a simple greedy refinement is applied to further reduce conflicts. For each edge, we try to reduce the conflict cost by enumerating the color assignment of its two vertices. The refinement continues until no vertex changes its coloring solution.

3.4 *Linear Programming and Iterative Rounding*

The ILP formulation in Eq. (1) is relaxed to LP by replacing the constraint of Eq. (9) with $0 \leq x_{i1} \leq 1$ and $0 \leq x_{i2} \leq 1$.

Table 1 Comparison on runtime and performance for TPL.

	ILP[3]		SDP[3]		MIS[5]		LPIR	
Circuit	cn#	CPU (s)	cn#	CPU (s)	cn#	CPU (s)	cn#	CPU (s)
C432	4	0.211	4	0.273	4	0.034	4	0.024
C499	0	0.174	0	0.071	0	0.033	0	0.025
C880	7	0.347	7	0.078	7	0.034	7	0.031
C1355	3	0.242	3	0.185	3	0.041	3	0.028
C1908	1	0.189	1	0.148	1	0.050	1	0.050
C2670	6	0.504	6	0.186	6	0.066	6	0.051
C3540	9	0.814	9	0.250	9	0.080	9	0.066
C5315	9	0.871	9	0.361	9	0.125	9	0.089
C6288	205	10.988	205	0.386	205	0.237	205	0.144
C7552	22	1.831	22	0.509	22	0.158	22	0.141
S1488	2	0.379	2	0.132	2	0.048	2	0.033
S38417	95	28.784	95	1.897	96	0.888	95	0.579
S35932	157	82.615	157	5.642	158	2.463	157	1.637
S38584	230	81.172	230	5.299	231	2.555	231	1.612
S15850	212	71.115	212	4.384	212	2.328	212	1.531
Average	64	18.682	64	1.320	64	0.609	64	0.403
Ratio	0.999	46.388	0.999	3.278	1.002	1.513	1.000	1.000

The critical issue from LP relaxation is that it may introduce many nonintegers in the solution, because a typical LP algorithm like simplex walks through the boundaries of the polyhedron space of feasible set and search for best solutions. It is very likely that the solutions at the boundaries of the polyhedron space contain nonintegers. For instance, with the constraint of Eq. (7), a trivial feasible solution is $x_{i1} = 0.5$, $x_{i2} = 0.5$, $\forall\, i \in V$. As shown in Fig. 4, the feasible space for the LP is denoted as the shaded region. The dashed red line denotes the objective function with optimal value. The grids consisting of dashed black lines are possible solutions with integer bits. We can see that the optimal solution from LP is (0.5,0.5). Efficient techniques are needed to push the LP solution to those blue dots in the feasible region with integer solutions while being close to the optimal point.

3.4.1 *Odd cycle constraints*

It is known that an odd cycle in a graph needs at least three colors. For the odd cycle example shown in Fig. 5, if the first bits of the vertices are equal, e.g., $x_{i1} = x_{j1} = x_{k1} = x_{l1} = x_{m1} = 0$, then it is not possible to obtain a solution without conflicts by adjusting x_{i2}, x_{j2}, x_{k2}, x_{l2}, x_{m2}. The LP relaxation will produce all 0.5 solutions for x_{i2}, x_{j2}, x_{k2}, x_{l2}, x_{m2} to satisfy constraints of Eqs. (3)–(6). These solutions are native noninteger solutions in the feasible set, which should be pruned. To avoid such kind of situations, the first bits of the vertices should not be equal. Figure 5(b) shows that as long as $x_{i1}, x_{j1}, x_{k1}, x_{l1}, x_{m1}$ are not all equal, it is very easy to find a coloring solution without any conflicts. We can avoid the situation of equality of the first bits by adding the following constraints, which forbid the cases of all zeros and all ones:

$$x_{i1} + x_{j1} + x_{k1} + x_{l1} + x_{m1} \geq 1, \tag{10}$$

$$(1 - x_{i1}) + (1 - x_{j1}) + (1 - x_{k1}) + (1 - x_{l1}) + (1 - x_{m1}) \geq 1. \tag{11}$$

It must be noted that although constraints of Eqs. (10) and (11) disallow the first bits to be identical, it helps to resolve the potential conflicts and nonintegers for the second bits. Similar technique can be applied to the second bits.

For a general odd cycle C, we have the following constraints:

$$\sum_{i \in C} x_{i1} \geq 1, \tag{12}$$

Table 2 Comparison on runtime and performance for QPL.

	ILP[9]		SDP[9]		MIS[5]		LPIR	
Circuit	cn#	CPU (s)	cn#	CPU (s)	cn#	CPU (s)	cn#	CPU (s)
C432	2	0.237	2	0.137	2	0.022	2	0.018
C499	5	0.296	5	0.078	5	0.043	5	0.032
C880	1	0.083	1	0.074	1	0.035	1	0.028
C1355	4	0.292	4	0.098	4	0.048	4	0.041
C1908	4	0.389	4	0.184	4	0.057	4	0.052
C2670	6	0.464	6	0.161	6	0.073	6	0.061
C3540	3	0.337	3	0.196	3	0.077	3	0.069
C5315	14	1.053	14	0.272	14	0.111	14	0.094
C6288	9	0.789	9	0.261	9	0.109	9	0.103
C7552	15	1.343	15	0.400	15	0.162	15	0.152
S1488	6	0.381	6	0.103	6	0.066	6	0.063
S38417	567	262.293	567	4.004	567	1.434	569	1.200
S35932	N/A	>3600	1792	18.994	1792	4.740	1810	3.761
S38584	N/A	>3600	1691	14.614	1691	3.710	1707	2.881
S15850	N/A	>3600	1500	12.054	1502	3.048	1505	2.405
Average	N/A	>737.8	375	3.442	375	0.916	377	0.731
Ratio	N/A	>1009.8	0.993	4.711	0.993	1.253	1.000	1.000

$$\sum_{i \in C}(1 - x_{i1}) \geq 1, \tag{13}$$

$$\sum_{i \in C} x_{i2} \geq 1, \tag{14}$$

$$\sum_{i \in C}(1 - x_{i2}) \geq 1, \tag{15}$$

where constraints of Eqs. (12) and (13) forbid the possibility of the first bits to be all zeros or all ones; constraints of Eqs. (14) and (15) forbid the same thing to the second bits. These constraints prune invalid solutions without losing the feasibility of the LP problem.

3.4.2 *Objective function biasing*

To eliminate the noninteger results in an LP solution, one heuristic is to push the corresponding variables to 0 or 1 by adjusting the objective function. For example, if x_{i1} turns out to be 0.6, it indicates that x_{i1} has the tendency to 1; hence, we add $(1 - x_{i1})$ to the objective function so that x_{i1} tends to be pushed to 1 during the next iteration. It can be generalized to the following rules:

$$\text{obj} = \begin{cases} \text{obj} + (1 - x_i), & \text{if } x_i > 0.5, \\ \text{obj} + (x_i), & \text{if } x_i < 0.5. \end{cases} \tag{16}$$

3.4.3 *Binding constraints analysis*

One drawback for the objective function biasing technique is that it cannot handle the noninteger value 0.5. Therefore, we propose a method to round those vertices with coloring solution $(x_{i1}, x_{i2}) = (0.5, 0.5)$ pairwisely by analyzing the related binding constraints. For a constraint in LP, if the inequality turns out to be equality according to the LP solution, we call it "binding", and this constraint is called "binding constraint."

Figure 6 shows an example of constraints for a vertex whose solution is (0.5,0.5). Let S_{i1} be the set of constraints only related to x_{i1}, S_{i2} be the set of constraints only related to x_{i2}, and the set of shared constraints is denoted by S_{ic}. For simplicity during illustration, assume each constraint is formatted in such a way that all variables are on the left side of the inequality operator and only constants are on the right side. At the same time, the coefficients for x_{i1} should remain positive.

If all constraints in S_{i1} share the same kind of operators such as "$\leq$" then these constraints will not be violated if x_{i1} is pushed from 0.5 to 0; similarly, if all operators are "$\geq$," then x_{i1} can be pushed from 0.5 to 1. The condition also holds for x_{i2} by checking all constraints in S_{i2}. With the analysis above, we can generate a candidate rounded solution for (x_{i1}, x_{i2}). The solution will not be accepted unless the rounded solution also satisfies all constraints in S_{ic}. For the example in Fig. 6, we can generate a candidate rounded solution (0,1) and then check if constraints in S_{ic} are satisfied as well. If true, (x_{i1}, x_{i2}) are rounded to (0,1). This technique will not affect the feasibility of the LP.

3.4.4 *Anchoring highest degree vertex*

During color assignment, one vertex of the graph can be assigned any arbitrary color, which has the effect of reducing the solution space without eliminating any optimal solutions and can improve performance. The degree of vertices in the graph varies from vertex to vertex, and the selection of precolored vertex leads to different coloring results. As a high-degree vertex has a large set of neighbors, the solution space will be largely reduced if its color is predetermined. Therefore, when constructing the mathematical formulation, we anchor the color of the vertex with highest degree.

3.5 *Graph Simplification*

Since the conflict graph constructed from initial layout is very large, we perform graph simplification to reduce the problem sizes. As the conflict graph is usually not strongly connected, independent components are extracted. Then, for each independent component, two more steps are further adopted to simplify it: iterative vertex removal (IVR)[3] and biconnected component extraction (BCE).[5] It shall be noted that if a simplification technique modifies the original graph, it needs a corresponding approach to recover the colors of vertices at the proper time. Since the two simplification methods we adopt will either remove vertices from the graph or divide graphs into components, we will also explain its recovery approach. In our experiment, IVR is performed

Table 3 Density variation.

Circuit	LPIR for TPL	LPIR for QPL
C432	0.022	0.044
C499	0.007	0.026
C880	0.001	0.005
C1355	0.001	0.014
C1908	0.001	0.014
C2670	0.003	0.010
C3540	0.001	0.009
C5315	0.006	0.003
C6288	0.000	0.006
C7552	0.001	0.012
S1488	0.008	0.016
S38417	0.001	0.041
S35932	0.000	0.065
S38584	0.000	0.055
S15850	0.000	0.055
Average	0.004	0.025

before BCE, so during the recovery process, the recovery algorithm for IVR is executed after that of BCE.

3.5.1 *Iterative vertex removal and density aware recovery*

In the graph coloring problem, if the degree of a vertex is smaller than the number of colors n, we can always remove it temporarily and assign color later, because its neighboring vertices in the graph will take, at most, $n-1$ colors. There will always be available colors left for this vertex. When a vertex is removed, some other vertices may turn out to be removable, so this procedure can be performed iteratively, which is shown in Fig. 7. In each iteration, the removed vertices are pushed into a stack, which is used to maintain the proper order during the vertex color recovery.

When assigning color to removed vertices during vertex color recovery, it is necessary to keep the popping order of the stack. For the example of Fig. 7, we will assign color in an order of 5, 4, 3, 2, 1. During the recovery process, a vertex may have multiple available colors; e.g., if vertex 5 is assigned to color 1, then vertex 4 will have two candidate colors, i.e., 2 and 3, in TPL. The choices of colors in the recovery stage play an important role in color density balancing. Therefore, we design an algorithm to consider color density during recovery, as shown in Algorithm 1. The basic idea is to compute the priority for each candidate color c of vertex v based on the distance $d_{c,v}$ with closest vertex of the same color c. The larger $d_{c,v}$ is, the higher priority the color c has. Eventually, the candidate color with largest $d_{c,v}$ will be chosen. The main loop from line 4 to line 17 in Algorithm 1 iterates though the vertices with the order defined by the stack. For each candidate color of a vertex, the distance $d_{c,v}$ is computed in line 7. The best color is computed from line 10 to line 15.

3.5.2 *Biconnected component extraction and color recovery*

In graph theory, a biconnected component is defined as a maximal biconnected subgraph. In TPL and QPL, we can divide a graph into biconnected components so that each component can be solved independently. Figure 8 shows an example of BCE. In the figure, the graph is split into three components a, b, and c. Vertex 3 is shared by components a and b; vertex 4 is shared by components b and c. These components can be colored independently and reunited later.

Table 4 Comparison on number of nonintegers for odd cycle constraints in TPL.

			w. OCC				w.o. OCC			
Circuit	LP1NI	LP1HI	LP2NI	LP2HI	LPENI	LPEHI	LP2NI	LP2HI	LPENI	LPEHI
C432	18	18	18	18	18	18	18	18	18	18
C499	12	12	18	18	18	18	12	12	12	12
C880	28	28	28	28	28	28	28	28	28	28
C1355	18	18	16	16	16	16	18	18	18	18
C1908	12	12	12	12	12	12	12	12	12	12
C2670	42	42	42	42	42	42	42	42	42	42
C3540	58	58	48	48	48	48	58	58	58	58
C5315	60	60	60	60	60	60	60	60	60	60
C6288	860	860	858	792	858	792	860	860	860	860
C7552	142	142	120	114	120	114	142	142	142	142
S1488	32	32	10	10	10	10	32	32	32	32
S38417	2730	2730	1510	1482	1510	1482	2730	2730	2730	2730
S35932	8330	8330	4339	4256	4339	4256	8330	8330	8330	8330
S38584	8073	8069	4281	4221	4281	4221	8075	8067	8075	8067
S15850	6918	6918	3756	3708	3756	3708	6918	6918	6918	6918
Average	1822	1822	1008	988	1008	988	1822	1822	1822	1822
Ratio	1.000	1.000	0.553	0.542	0.553	0.542	1.000	1.000	1.000	1.000

Since each component is processed independently, it is likely to result in the condition that the colors of shared vertices in different components are different like vertex 3 in components a and b. Therefore, color rotation is necessary during the process of recovery. The color assignments of component b should be rotated in such a way that vertex $3'$ in component b has identical color to vertex 3 in component a. As the coloring solution of component b is changed, vertex 4 and vertex $4'$ no longer remain the same color, so component c should follow component b and rotate its coloring solution as well.

For a more general procedure of color rotation during the recovery, we can construct an undirected acyclic graph (UAG) in which each biconnected component is a vertex and two vertices are connected if the corresponding components share a vertex in the original graph. The color rotation for biconnected components can be solved by applying depth first search to the UAG.

4 Experimental Results

Our algorithms were implemented in C++ and tested on an 8-Core 3.40 GHz Linux server with 32 GB RAM. The same benchmarks from Ref. 3 are used. Gurobi[22] is used as the ILP and LP solver. The minimum coloring distance for TPL is set to 120 nm and that for QPL is set to 160 nm. We compare our algorithm with the state-of-the-art ILP and SDP algorithms from Yu et al.[3] in Table 1. We also implemented the maximum independent set (MIS) based algorithm from Fang et al.[5] to compare with our results, as the original implementation from Fang et al.[5] focuses on stitch insertion. The basic idea of the MIS-based algorithm searches for best independent set that minimizes the total edge weights in the residual graph (the graph after removing the independent set) for each color. Once all the three or four colors are assigned, the rest of the vertices are greedily assigned colors for minimum conflicts.

In Tables 1 and 2, our algorithm is shown as LPIR. Conflict number is denoted by "cn#" and runtime is denoted by "CPU" in seconds. As stitch insertion is not allowed, the stitch number is always zero and thus not shown in the table. We can see that the LPIR algorithm is able to achieve minimum conflicts for almost all benchmarks in TPL with 46× speedup to ILP and 3.3× speedup to SDP. In QPL, the speedup from LPIR is even more impressive, which achieves 1000× speedup to ILP and 4.7× speedup to SDP, while it only produces around 0.7% more conflicts than SDP. The

Table 5 Comparison on number of nonintegers for odd cycle constraints in QPL.

			w. OCC				w.o. OCC			
Circuit	LP1NI	LP1HI	LP2NI	LP2HI	LPENI	LPEHI	LP2NI	LP2HI	LPENI	LPEHI
C432	24	16	20	8	20	8	24	16	24	16
C499	20	20	20	10	20	10	20	20	20	20
C880	6	6	7	0	7	0	6	6	6	6
C1355	16	16	16	8	16	8	16	16	16	16
C1908	26	26	25	19	25	19	26	26	26	26
C2670	24	24	24	12	24	12	24	24	24	24
C3540	14	14	14	10	14	10	14	14	14	14
C5315	62	62	62	36	62	36	62	62	62	62
C6288	46	46	45	22	45	22	46	46	46	46
C7552	62	62	60	32	60	32	62	62	62	62
S1488	24	24	24	12	24	12	24	24	24	24
S38417	2707	2469	2616	1370	2614	1370	2677	2465	2677	2465
S35932	9747	8449	9253	4900	9264	4901	9493	8283	9491	8281
S38584	8400	7578	8096	4273	8099	4276	8283	7459	8283	7459
S15850	7261	6575	7033	3689	7033	3670	7160	6502	7160	6502
Average	1896	1692	1821	960	1822	959	1862	1668	1862	1668
Ratio	1.000	1.000	0.960	0.567	0.961	0.567	0.982	0.986	0.982	0.986

Table 6 Comparison of conflicts on anchoring highest degree vertex.

	cn# for TPL		cn# for QPL	
Circuit	w. Anchoring	w.o. Anchoring	w. Anchoring	w.o. Anchoring
C432	4	4	2	2
C499	0	0	5	5
C880	7	7	1	1
C1355	3	3	4	4
C1908	1	1	4	4
C2670	6	6	6	6
C3540	9	9	3	3
C5315	9	9	14	14
C6288	205	205	9	9
C7552	22	22	15	15
S1488	2	2	6	6
S38417	95	95	569	569
S35932	157	157	1810	1817
S38584	231	230	1707	1711
S15850	212	212	1505	1509
Average	64	64	377	378

small degradation in conflicts is reasonable because designers need to manually fix conflicts by modifying the layout anyway. Compared with MIS-based approach,[5] we have slightly smaller conflicts for TPL and slightly more conflicts for QPL. The speedup is around 50% for TPL and 25% for QPL.

We also study the density balancing of the experimental results, which is handled during the vertex recovery of IVR in Sec. 3.5.1. We adopt the metric of density variation in Ref. 12 to evaluate our density uniformity as in the following equation:

$$\sigma = \frac{d_{\max}}{d_{\min}} - 1, \tag{17}$$

where $d_{\max}$ is the maximum color density of all colors and $d_{\min}$ is the minimum color density. In an ideal case, σ should approach zero; i.e., $d_{\max}$ is equal to $d_{\min}$. Table 3 shows the density variation of LPIR for TPL and QPL. We can see that in TPL, the average density variation is only 0.4% and most benchmarks have a variation approaching zero. For QPL, the average density variation is 2.5%, which is acceptable.

4.1 *Effectiveness of Odd Cycle Constraints*

We discuss the empirical results of noninteger removal from odd cycle constraints here. We compare the number of nonintegers and half-integers for odd cycle constraints in Tables 4 and 5. The columns "LP1NI" and "LP1HI" denote the number of nonintegers and half-integers, respectively, after first iteration of LP without any odd cycle constraint or objective biasing. The columns "LP2NI" and "LP2HI" denote the number of nonintegers and half-integers, respectively, after second iteration of LP. The columns "LPENI" and "LPEHI" denote the number of nonintegers and half-integers, respectively, after last iteration of LP. The column "w. OCC" denotes that odd cycle constraints are applied, whereas the column "w.o. OCC" denotes that odd cycle constraints are not applied. As each design will be decomposed to components after graph simplification, we sum up all the number of nonintegers and half-integers for all components. For TPL, odd cycle constraints in general remove more than 40% nonintegers on average compared with both the results from first iteration of LP and that without odd cycle constraints. For QPL, it achieves around 2% fewer nonintegers and more than 40% fewer half-integers, though in some corner cases, it ends up with slightly more nonintegers like that in benchmark C880 for QPL. We also observe that the LPIR generally finishes in two to three iterations for each component, so the numbers of nonintegers and half-integers in the end of LPIR are very close to that of the second iteration.

4.2 *Effectiveness of Anchoring Highest Degree Vertex*

Anchoring highest degree vertex in Sec. 3.4.4 can reduce the solution space during LP solving; the empirical results in Table 6 also show that it can achieve slightly fewer number of conflicts. Considering that the solution space of LP generally contains many more noninteger feasible solutions than integer feasible solutions, when we prune the solution space by anchoring a vertex, more noninteger solutions are likely to be removed than integer solutions, which ends up with higher probability of achieving better solution quality.

5 Discussion

In this section, we show a potential way to extend the framework to handle stitch insertion and discuss whether it is suitable to such an application, since in applications like metal layers, stitch insertion is allowed to resolve conflicts.

Inspired by Ref. 23, we try to integrate constraints and objective for stitch edges into the ILP formulation in Eq. (1). Suppose the conflict graph is constructed with both conflict edges and stitch edges and let E_s denote the set of stitch edges. We introduce stitch variable s_{ij} for each stitch edge and adjust the ILP formulation in Eq. (1) as follows:

$$\min \text{ obj} + w_s \sum_{e_{ij} \in E_s} s_{ij}, \tag{18}$$

such that Eqs. (2) to (9),

$$x_{i1} - x_{j1} \le s_{ij}, \quad \forall\, e_{ij} \in E_s, \tag{19}$$

$$x_{j1} - x_{i1} \le s_{ij}, \quad \forall\, e_{ij} \in E_s, \tag{20}$$

$$x_{i2} - x_{j2} \leq s_{ij}, \quad \forall\, e_{ij} \in E_s, \tag{21}$$

$$x_{j2} - x_{i2} \leq s_{ij}, \quad \forall\, e_{ij} \in E_s, \tag{22}$$

where w_s denotes the weight for stitch edges, which is set to 0.1 in the experiment and the term "obj" still denotes the changeable objective in Eq. (1). Constraint Eqs. (19)–(22) guarantee that if either (x_{i1}, x_{j1}) are the same or (x_{i2}, x_{j2}) are the same, s_{ij} has to be no smaller than 1. Together with the additional summation of s_{ij} in the objective, stitches are minimized in Eq. (6). In the LPIR scheme, techniques like odd cycle constraints (Sec. 3.4.1) and objective function biasing (Sec. 3.4.2) are adjusted to work on conflict edges only, whereas other techniques like binding constraints analysis (Sec. 3.4.3) are compatible to all edges.

Table 7 shows the comparison of stitches and conflicts when allowing stitch insertion. Column "sn#" denotes the number of stitches and column "cn#" denotes the number of conflicts. We observe that LPIR results in more conflicts than both SDP and ILP, even though the average amount of stitches is smaller than that of SDP. Actually, the observation also indicates that LPIR fails to differentiate the importance of conflicts from that of stitches, while taking them as equal importance instead, especially in benchmarks, such as "C6288," "S38417," etc. As a consequence, it often trades number of conflicts for number of stitches and produces solutions with poor quality. Although the weight w_s for stitches is adjustable, the solution quality does not improve too much by adjusting it. In addition, our study shows that the additional constraints for stitches [Eqs. (19)–(22)] result in many more nonintegers, especially half-integers, in the LP solutions, which deteriorates the number of conflicts after IR. The reason probably lies in the fact that the polyhedron space is changed to contain more nonintegers at the boundaries when the additional constraints for stitches are incorporated. Therefore, LPIR scheme is suitable to the situation without stitch insertion while it fails to address the importance of conflicts when stitching is allowed. The application of this scheme is focused on vias and contacts where stitching is disabled.

6 Conclusion

In this paper, we propose a new and effective algorithm for a layout decomposition problem for TPL and QPL. By utilizing the features in the polyhedron space of the feasible set, we approximate the ILP formulation with IR to the LP relaxation. Several techniques are proposed to shrink the polyhedron space without losing the feasibility, such as odd cycle constraints, binding constraint analysis, and vertex anchoring. Experiment results show the effectiveness and efficiency

Table 7 Comparison on performance with stitch insertion.

	ILP for TPL3		SDP for TPL3		LPIR for TPL		ILP for QPL9		SDP for QPL9		LPIR for QPL	
Circuit	sn#	cn#	sn#	cn#	sn#	cn#	cn#	cn#	sn#	cn#	sn#	cn#
C432	4	0	4	0	8	0	0	2	0	2	0	2
C499	0	0	0	0	0	0	3	1	4	1	4	1
C880	3	0	7	0	5	0	0	1	0	1	1	2
C1355	2	0	3	0	2	0	1	0	4	0	1	0
C1908	0	0	1	0	0	0	2	2	3	2	2	2
C2670	2	0	6	0	4	0	2	0	6	0	3	0
C3540	4	1	9	1	4	1	1	1	3	1	1	1
C5315	5	0	9	0	5	0	6	1	13	1	7	2
C6288	111	0	213	1	139	19	0	9	0	9	3	9
C7552	10	0	26	0	9	1	9	2	13	2	6	12
S1488	1	0	3	0	1	0	0	0	6	0	0	0
S38417	20	27	57	20	28	35	66	20	551	20	50	66
S35932	42	77	60	46	86	102	257	46	1745	50	259	261
S38584	45	81	131	36	78	99	N/A	N/A	1653	41	173	183
S15850	57	57	119	34	75	73	N/A	N/A	1462	42	185	148
Average	20	16	43	9	30	22	27	7	364	11	46	46

of the algorithm compared with ILP-, SDP-, and MIS-based approaches.

Acknowledgments

This work is supported in part by National Science Foundation (NSF), Semiconductor Research Corporation (SRC), and Chinese University of Hong Kong (CUHK) Direct Grant for Research.

References

1. D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nanolithography," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**(10), 1453–1472 (2013).
2. R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of computer computations*, pp. 85–103, Springer US (1972).
3. B. Yu et al., "Layout decomposition for triple patterning lithography," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **34**, 433–446 (2015).
4. R. S. Ghaida et al., "A novel methodology for triple/multiple-patterning layout decomposition," *Proc. SPIE* **8327**, 83270M (2012).
5. S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **33**, 397–408 (2014).
6. K. Lucas et al., "Implications of triple patterning for 14 nm node design and patterning," *Proc. SPIE* **8327**, 832703 (2012).
7. J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 1–6 (2013).
8. Y. Zhang et al., "Layout decomposition with pairwise coloring for multiple patterning lithography," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 170–177 (2013).
9. B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *ACM/IEEE Design Automation Conf. (DAC)*, Vol. 53, pp. 1–6 (2014).
10. B. Yu et al., "Design for manufacturability and reliability in extreme-scaling VLSI," *Sci. China Inf. Sci.* **59**, 061406 (2016).
11. Z. Chen, H. Yao, and Y. Cai, "SUALD: spacing uniformity-aware layout decomposition in triple patterning lithography," in *IEEE Int. Symp. on Quality Electronic Design (ISQED)*, pp. 566–571 (2013).
12. B. Yu et al., "A high-performance triple patterning layout decomposer with balanced density," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 163–169 (2013).
13. X. Li, Z. Zhu, and W. Zhu, "Discrete relaxation based layout decomposition for triple patterning lithography," *IEEE Trans. Comput.* **66**(2), 1 (2016).
14. I. H.-R. Jiang and H.-Y. Chang, "Multiple patterning layout decomposition considering complex coloring rules and density balancing," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* PP(99), 1 (2017).
15. H. Tian et al., "A polynomial time triple patterning algorithm for cell based row-structure layout," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 57–64 (2012).
16. H. Tian et al., "Constrained pattern assignment for standard cell based triple patterning lithography," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 178–185 (2013).
17. H. Tian et al., "An efficient linear time triple patterning solver," in *IEEE/ACM Asia and South Pacific Design Automation Conf. (ASPDAC)*, pp. 208–213 (2015).
18. H.-A. Chien et al., "A cell-based row-structure layout decomposer for triple patterning lithography," in *ACM Int. Symp. on Physical Design (ISPD)*, pp. 67–74 (2015).
19. B. Yu et al., "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **34**, 726–739 (2015).
20. B. Yu, G. Garreton, and D. Z. Pan, "Layout compliance for triple patterning lithography: an iterative approach," *Proc. SPIE* **9235**, 923504 (2014).
21. D. Guo et al., "Model-based multiple patterning layout decomposition," *Proc. SPIE* **9635**, 963522 (2015).
22. Gurobi Optimization Inc., "Gurobi optimizer reference manual," Gurobi Optimizer, http://www.gurobi.com (2014).
23. B. Yu et al., "Layout decomposition for triple patterning lithography," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 1–8 (2011).

Yibo Lin received his BS degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013. He is currently pursuing his PhD at the Department of Electrical and Computer Engineering, University of Texas at Austin. His research interests include physical design and design for manufacturability. He has received Franco Cerrina Memorial Best Student Paper Award at SPIE Advanced Lithography Conference 2016. He has interned at IMEC, Cadence, and Oracle.

Xiaoqing Xu is currently a senior research engineer at the ARM Research Group, Austin, Texas. His research interests include robust standard cell design, design for manufacturability and physical design. His research has been recognized with numerous awards including Golden Medal at ACM Student Research Competition at ICCAD 2016, SPIE BACUS fellowship in 2016, Best in Session Award at SRC TECHCON 2015, William J. McCalla Best Paper Award at ICCAD 2013.

Bei Yu is currently an assistant professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served in the editorial boards of integration, the VLSI Journal, and IET Cyber-Physical Systems: Theory and Applications. He has received four Best Paper Awards at ISPD 2017, SPIE Advanced Lithography Conference 2016, ICCAD 2013, ASPDAC 2012, EDAA Outstanding Dissertation Award in 2014, and SPIE Scholarship in 2013.

Ross Baldick received his BSc and BE degrees in electrical engineering from the University of Sydney, Australia, in 1983 and 1985 and his MS and PhD degrees from the University of California, Berkeley, in 1988 and 1990, respectively. He is a professor and leland barclay fellow in the Department of Electrical and Computer Engineering at the University of Texas at Austin. He is a fellow of the IEEE and the recipient of the 2015 IEEE PES Outstanding Power Engineering Educator Award.

David Z. Pan is Engineering Foundation Professor at ECE Department, The University of Texas at Austin. His research interests include cross-layer design for manufacturability, reliability, security, and CAD for emerging technologies. He has published over 280 technical papers, and graduated over 20 PhDs. He has received numerous awards, including SRC Technical Excellence Award and many Best Paper Awards at premier venues. He is a fellow of IEEE and SPIE.