Triple/Quadruple Patterning Layout Decomposition via Novel Linear Programming and Iterative Rounding

Yibo Lin[†], Xiaoqing Xu[†], Bei Yu[‡], Ross Baldick[†] and David Z. Pan[†]
[†]ECE Dept., University of Texas at Austin, Austin, TX USA
[‡]CSE Dept., The Chinese University of Hong Kong, NT, Hong Kong
Email[†]: {yibolin, xiaoqingxu}@cerc.utexas.edu, {baldick, dpan}@ece.utexas.edu
Email[‡]: byu@cse.cuhk.edu.hk

ABSTRACT

As feature size of the semiconductor technology scales down to $10nm$ 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 (DSA). Due to the delay of EUVL and EBL, triple and even quadruple patterning are 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, while it is forbidden for contact and via layers.

In this paper, 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 and iterative rounding (LPIR) solving technique to reduce the number of non-integers 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.

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 $10nm$ 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 (LELELE) 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 3-coloring and QPL can be formulated to 4-coloring. The minimum distance to insert a conflict edge between two features is defined as *coloring distance*.

Graph coloring is known as an NP-complete problem for a color number larger than two.[2] A layout may include sub-graphs such as 4-clique (K4) structure that is not 3-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,

Design-Process-Technology Co-optimization for Manufacturability X,
edited by Luigi Capodieci and Jason Cain Proc. of SPIE Vol. 9781, 97810M
© 2016 SPIE · CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2218628

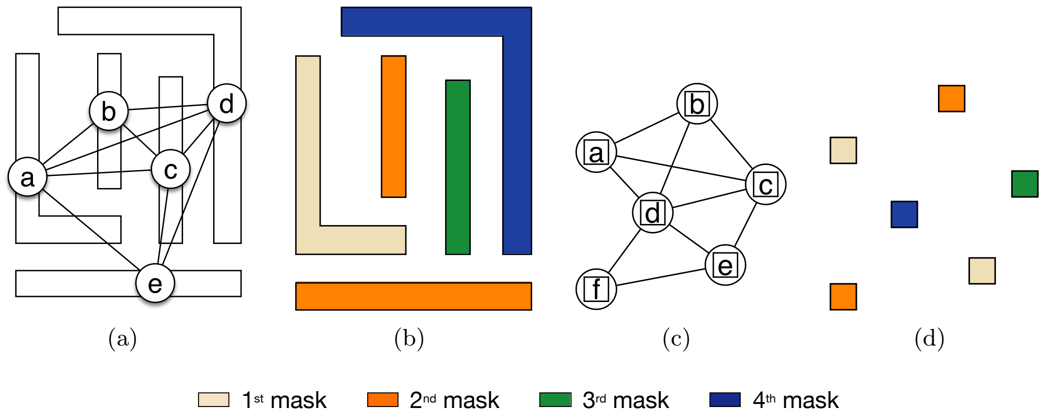

Figure 1. Multiple patterning layout decomposition: (a) metal layer conflict graph, (b) corresponding decomposition, (c) contact layer conflict graph, and (d) corresponding decomposition.

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.[3–9] Although ILP can solve the problem optimally, it suffers from exponential runtime. SDP and other heuristic approaches are introduced to speedup the decomposition process with trade-offs 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.[10, 11] For row based layout structures, even faster approaches have been proposed with the guarantee of optimality.[12–15] For a single standard cell, Yu et al. proposed search based methods to enumerate all possible coloring solutions.[16] Besides, Yu et al. developed an incremental framework to accelerate conventional layout decomposition flow.[17] Recently, Guo et al. studied the lithography impact of different decomposition solutions based on lithography models.[18]

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 novel linear programming (LP) and iterative rounding scheme to solve layout decomposition efficiently with high performance.
2. We propose an odd cycle based technique to prune native non-integer solutions in the feasible set of LP, which can effectively reduce non-integer solutions in LP.
3. Experimental results show that our algorithm gets 1.8x speedup for TPL and 2.6x speedup for QPL compared with the state-of-the-art layout decomposer with less 2% degradation in final conflict numbers.

The rest of the paper is organized as follows. In Section 2, we discuss the preliminary and problem formulation. In Section 3, we explain the algorithms such as integer linear programming formulation, linear programming relaxation and our iterative rounding scheme. The experimental result is presented in Section 4 and we conclude this paper in Section 5.

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

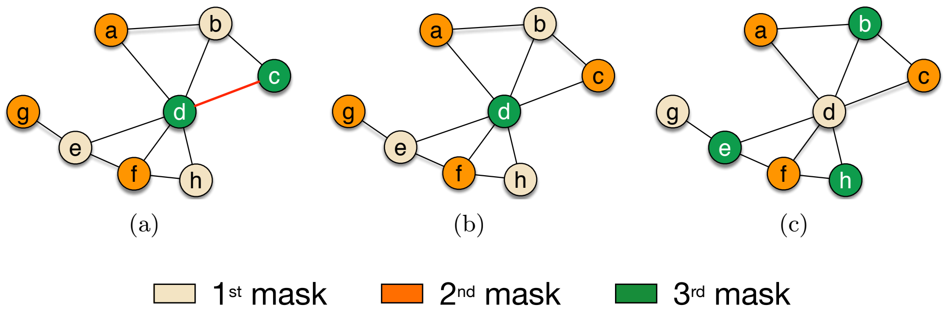

Figure 2. Example of (a) conflict and (b) unbalanced decomposition (c) balanced decomposition in triple patterning layout decomposition.

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. Fig. 2(a) shows an example of conflict where node 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 CD uniformity.[11] Fig. 2(b) and Fig. 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 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 iterative rounding scheme in details.

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 post color assignment. In the end, the final coloring results are produced. The detailed flow for LPIR will be explained in Section 3.3.

3.2 ILP 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 Formulation (1). For each conflict edge in the edge set E_c, the situation of identical colors on both vertices is forbidden by Constraints (1c) to (1f); e.g. Constraint (1c) 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. Constraint (1b) 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,[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

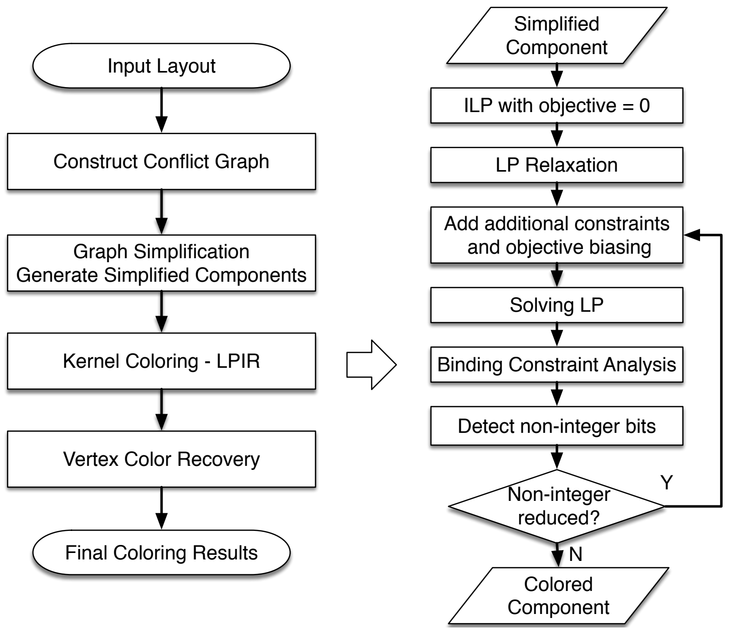

Figure 3. Overall flow for our coloring framework.

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.

$$\begin{aligned}
\min \quad & \text{obj} && \text{(1a)}\\
\text{s.t.} \quad & x_{i1} + x_{i2} \le 1, && \text{(1b)}\\
& x_{i1} + x_{i2} + x_{j1} + x_{j2} \ge 1, \quad \forall e_{ij} \in E_c, && \text{(1c)}\\
& x_{i1} + \bar{x}_{i2} + x_{j1} + \bar{x}_{j2} \ge 1, \quad \forall e_{ij} \in E_c, && \text{(1d)}\\
& \bar{x}_{i1} + x_{i2} + \bar{x}_{j1} + x_{j2} \ge 1, \quad \forall e_{ij} \in E_c, && \text{(1e)}\\
& \bar{x}_{i1} + \bar{x}_{i2} + \bar{x}_{j1} + \bar{x}_{j2} \ge 1, \quad \forall e_{ij} \in E_c, && \text{(1f)}\\
& \bar{x}_{i1} = 1 - x_{i1}, \quad \forall i \in V, && \text{(1g)}\\
& \bar{x}_{i2} = 1 - x_{i2}, \quad \forall i \in V, && \text{(1h)}\\
& x_{i1}, x_{i2} \in \{0, 1\}, \quad \forall i \in V. && \text{(1i)}
\end{aligned}$$

3.3 LPIR Flow

Although the ILP formulation is able to find optimal color assignment when there exists conflict free solution, it has 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 non-integer 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 with the LP with an objective of zero. To deal with non-integers in the solution from LP, we identify some native non-integer solutions in the feasible set resulted from odd cycles. Then additional constraints are introduced to prune these native non-integer solutions. During each iteration, objective function is changed from the original LP formulation to push non-integer to integers. In particular, these additional constraints and objective function change will not break the feasibility of possible coloring assignment. We continue the LPIR iterations until no further improvement.

3.4 Linear Programming and Iterative Rounding

The ILP formulation in (1) is relaxed to LP by replacing Constraint (1i) with $0 \leq x_{i1} \leq 1$ and $0 \leq x_{i2} \leq 1$. The critical issue from LP relaxation is that it may introduce many non-integers 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 non-integers. For instance, with the constraint (1g), 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 light green region. The dash 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.

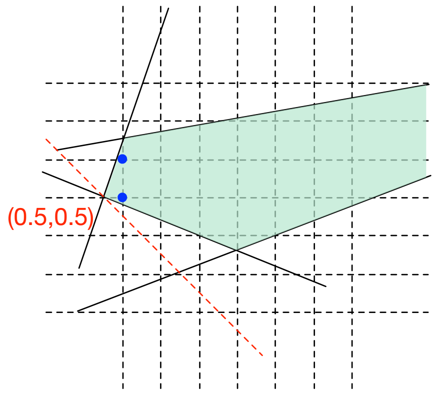

Figure 4. The polyhedron for feasible linear programming solutions.

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. $x_{i1} = x_{j1} = x_{k1} = x_{l1} = x_{m1} = 0$, 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 (1c) to (1f). These solutions are native non-integer 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. Just like Fig. 5(b) shows, 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 conflict. We can avoid the situation of equality of the first bits by adding following constraints, which forbids the cases of all zeros and all ones:

$$x_{i1} + x_{j1} + x_{k1} + x_{l1} + x_{m1} \geq 1, \tag{2a}$$

$$(1 - x_{i1}) + (1 - x_{j1}) + (1 - x_{k1}) + (1 - x_{l1}) + (1 - x_{m1}) \geq 1. \tag{2b}$$

It must be noted that although Constraints (2a) and (2b) disallow the first bits to be identical, it helps to resolve the potential conflicts and non-integers for the second bits. Similar technique can be applied to the second bits.

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

$$\sum_{i \in C} x_{i1} \geq 1, \tag{3a}$$

$$\sum_{i \in C} (1 - x_{i1}) \geq 1, \tag{3b}$$

$$\sum_{i \in C} x_{i2} \geq 1, \tag{3c}$$

$$\sum_{i \in C} (1 - x_{i2}) \geq 1, \tag{3d}$$

where Constraint (3a) and (3b) forbid the possibility of the first bits to be all zeros or all ones; Constraint (3c) and (3d) forbid the same thing to the second bits. These constraints prune invalid solutions without losing the feasibility of the LP problem.

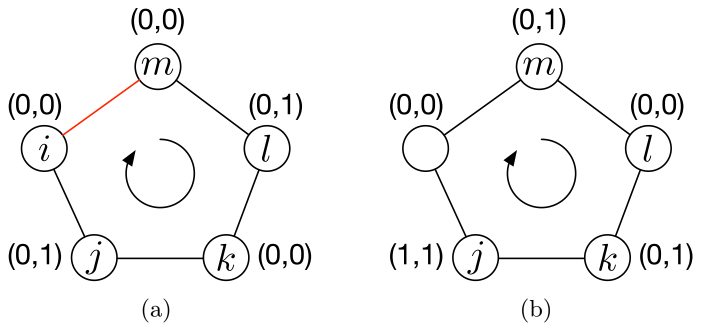

Figure 5. One possible odd cycle in the conflict graph (a) a coloring conflict from due to identical first bit values of nodes (b) resolved conflict.

3.4.2 Objective Function Biasing

To eliminate the non-integer 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 following rules:

$$\text{obj} = \begin{cases} \text{obj} + (1 - x_i), & \text{if } x_i > 0.5, \\ \text{obj} + (x_i), & \text{if } x_i < 0.5. \end{cases} \tag{4}$$

3.4.3 Binding Constraints Analysis

One drawback for the objective function biasing technique cannot handle non-integers like 0.5. Therefore, we propose a method to round those vertex 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*.

Fig. 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 coloring solution can actually rotate to generate other coloring solutions. To reduce the solution space, it will not do harm to the optimality if one vertex of the graph is pre-colored. The degree of vertices in the graph varies from vertex to vertex, and the selection of pre-colored 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 pre-determined. Therefore, when constructing the mathematic formulation, we anchor the color of the vertex with highest degree.

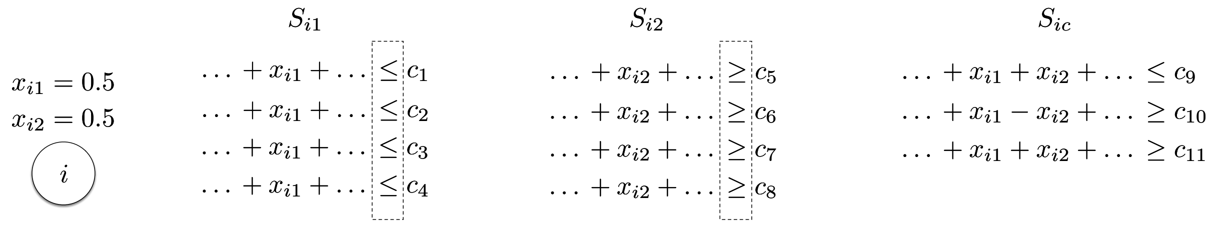
S_{i1}
S_{i2}
S_{ic}
$x_{i1} = 0.5$
$x_{i2} = 0.5$
i
$\ldots + x_{i1} + \ldots \le c_1$
$\ldots + x_{i1} + \ldots \le c_2$
$\ldots + x_{i1} + \ldots \le c_3$
$\ldots + x_{i1} + \ldots \le c_4$
$\ldots + x_{i2} + \ldots \ge c_5$
$\ldots + x_{i2} + \ldots \ge c_6$
$\ldots + x_{i2} + \ldots \ge c_7$
$\ldots + x_{i2} + \ldots \ge c_8$
$\ldots + x_{i1} + x_{i2} + \ldots \le c_9$
$\ldots + x_{i1} - x_{i2} + \ldots \ge c_{10}$
$\ldots + x_{i1} + x_{i2} + \ldots \ge c_{11}$

Figure 6. An example of binding constraints analysis.

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 vertices at 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 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.

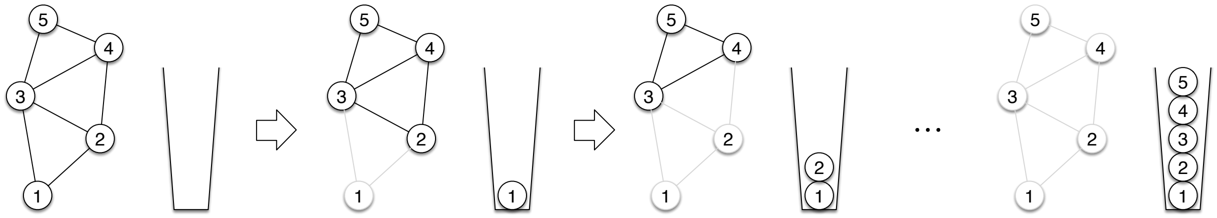

Figure 7. An example of iterative vertex removal in TPL.

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 a 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, shown in Alg. 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 Alg. 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. Fig. 8 shows an example of biconnected component extraction. In the figure, the graph is split into three components a,

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
```

b, and *c*. Vertex 3 is shared by component *a* and *b*; vertex 4 is shared by component *b* and *c*. These components can be colored independently and reunited later.

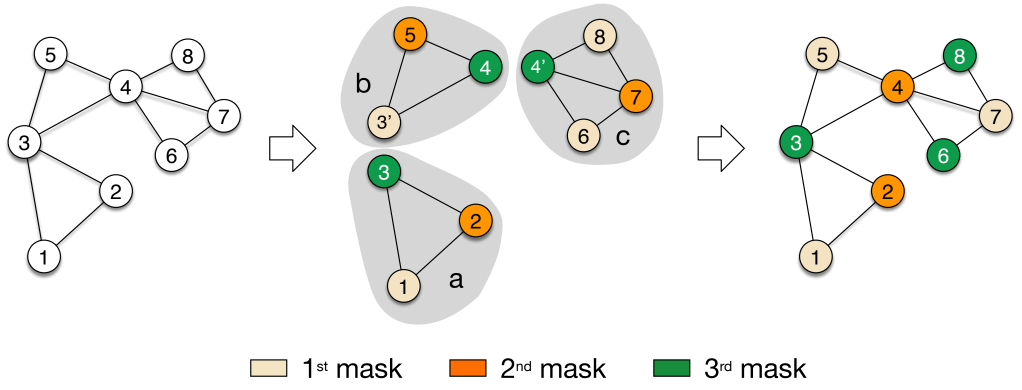

Figure 8. An example of biconnected component extraction and color recovery.

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 component *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 (DFS) 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`[19] is used as the ILP and LP solver. The minimum coloring distance for TPL is set to $120nm$ and that for QPL is set to $160nm$. We compare our algorithm with the state-of-the-art ILP and SDP algorithms from Ref. 3 in Table 1.

Table 1. Comparison on Runtime and Performance

Circuit	ILP for TPL[3]		SDP for TPL[3]		LPIR for TPL		ILP for QPL[9]		SDP for QPL[9]		LPIR for QPL	
	cn#	CPU(s)	cn#	CPU(s)	cn#	CPU(s)	cn#	CPU(s)	cn#	CPU(s)	cn#	CPU(s)
C432	4	0.211	4	0.273	4	0.026	2	0.237	2	0.137	2	0.027
C499	0	0.174	0	0.071	0	0.035	5	0.296	5	0.078	5	0.046
C880	7	0.347	7	0.078	7	0.039	1	0.083	1	0.074	1	0.043
C1355	3	0.242	3	0.185	3	0.053	4	0.292	4	0.098	4	0.062
C1908	1	0.189	1	0.148	1	0.077	4	0.389	4	0.184	4	0.085
C2670	6	0.504	6	0.186	6	0.11	6	0.464	6	0.161	6	0.129
C3540	9	0.814	9	0.25	9	0.142	3	0.337	3	0.196	3	0.151
C5315	9	0.871	9	0.361	9	0.199	14	1.053	14	0.272	14	0.219
C6288	205	10.988	205	0.386	205	0.263	9	0.789	9	0.261	9	0.224
C7552	22	1.831	22	0.509	22	0.29	15	1.343	15	0.4	15	0.384
S1488	2	0.379	2	0.132	2	0.065	6	0.381	6	0.103	6	0.082
S38417	95	28.784	95	1.897	95	1.069	567	262.293	567	4.004	573	1.965
S35932	157	82.615	157	5.642	157	2.765	N/A	>3600	1792	18.994	1854	5.79
S38584	230	81.172	230	5.299	230	2.817	N/A	>3600	1691	14.614	1727	4.781
S15850	212	71.115	212	4.384	212	2.662	N/A	>3600	1500	12.054	1521	4.419
avg.	64	18.682	64	1.320	64	0.707	N/A	>737.8	376	3.442	383	1.227

In Table 1, 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 all benchmarks in TPL with 26x speedup to ILP and 1.8x speedup to SDP. In QPL, the speedup from LPIR is even more impressive, which achieves 601x speedup to ILP and 2.8x speedup to SDP, while it only produces around 2% more conflicts than SDP. The small degradation in conflicts is reasonable because designers need to manually fix conflicts by modifying the layout anyway.

We also study the density balancing of the experimental results, which is handled during the vertex recovery of IVR in Section 3.5.1. We adopt the metric of density variation in[11] to evaluate our density uniformity as in following equation:

$$\sigma = \frac{d_{max}}{d_{min}} - 1, \tag{5}$$

where d_{max} is the maximum color density of all colors, and d_{min} is the minimum color density. In ideal case, σ should approach zero; i.e. d_{max} is equal to d_{min}. Table 2 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 3%, which is acceptable. The outlier of the variation in QPL lies in benchmark C432 which only contains 1109 vertices. Although there are only 25 more vertices assigned to the first mask than that to the fourth mask, the variation is large due to the small value of total vertices, because small benchmarks are more sensitive to small density variation.

5. CONCLUSION

In this paper, we propose a new and effective algorithm for 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 iterative rounding 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 of the algorithm compared with both ILP and SDP.

Table 2. Density Variation

Circuit	LPIR for TPL	LPIR for QPL
C432	0.021	0.093
C499	0.020	0.025
C880	0.001	0.011
C1355	0	0.013
C1908	0.001	0.028
C2670	0.003	0.015
C3540	0	0.008
C5315	0.005	0.001
C6288	0	0.007
C7552	0	0.008
S1488	0.004	0.016
S38417	0	0.043
S35932	0	0.067
S38584	0	0.054
S15850	0	0.055
avg.	0.004	0.030

Acknowledgment

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] Pan, D. Z., Yu, B., and Gao, J.-R., "Design for manufacturing with emerging nanolithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **32**(10), 1453–1472 (2013).

[2] Karp, R. M., [*Reducibility among combinatorial problems*], Springer (1972).

[3] Yu, B., Yuan, K., Ding, D., and Pan, D. Z., "Layout decomposition for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **34**, 433–446 (March 2015).

[4] Ghaida, R. S., Agarwal, K. B., Liebmann, L. W., Nassif, S. R., and Gupta, P., "A novel methodology for triple/multiple-patterning layout decomposition," in [*Proceedings of SPIE*], **8327** (2012).

[5] Fang, S.-Y., Chang, Y.-W., and Chen, W.-Y., "A novel layout decomposition algorithm for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **33**, 397–408 (March 2014).

[6] Lucas, K., Cork, C., Yu, B., Luk-Pat, G., Painter, B., and Pan, D. Z., "Implications of triple patterning for 14 nm node design and patterning," in [*Proceedings of SPIE*], **8327** (2012).

[7] Kuang, J. and Young, E. F., "An efficient layout decomposition approach for triple patterning lithography," in [*ACM/IEEE Design Automation Conference (DAC)*], 69:1–69:6 (2013).

[8] Zhang, Y., Luk, W.-S., Zhou, H., Yan, C., and Zeng, X., "Layout decomposition with pairwise coloring for multiple patterning lithography," in [*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*], 170–177 (2013).

[9] Yu, B. and Pan, D. Z., "Layout decomposition for quadruple patterning lithography and beyond," in [*ACM/IEEE Design Automation Conference (DAC)*], 53:1–53:6 (2014).

[10] Chen, Z., Yao, H., and Cai, Y., "SUALD: Spacing uniformity-aware layout decomposition in triple patterning lithography.," in [*IEEE International Symposium on Quality Electronic Design (ISQED)*], 566–571 (2013).

[11] Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., and Pan, D. Z., "A high-performance triple patterning layout decomposer with balanced density," in [*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*], 163–169 (2013).

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

[13] Tian, H., Du, Y., Zhang, H., Xiao, Z., and Wong, M. D. F., "Constrained pattern assignment for standard cell based triple patterning lithography," in [*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*], 178–185 (2013).

[14] Tian, H., Zhang, H., Xiao, Z., and Wong, M. D. F., "An efficient linear time triple patterning solver," in [*IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*], 208–213 (2015).

[15] Chien, H.-A., Han, S.-Y., Chen, Y.-H., and Wang, T.-C., "A cell-based row-structure layout decomposer for triple patterning lithography," in [*ACM International Symposium on Physical Design (ISPD)*], 67–74 (2015).

[16] Yu, B., Xu, X., Gao, J.-R., Lin, Y., Li, Z., Alpert, C., and Pan, D. Z., "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **34**, 726–739 (May 2015).

[17] Yu, B., Garreton, G., and Pan, D. Z., "Layout compliance for triple patterning lithography: an iterative approach," in [*Proceedings of SPIE*], **9235** (2014).

[18] Guo, D., Tian, H., Du, Y., and Wong, M. D., "Model-based multiple patterning layout decomposition," in [*Proceedings of SPIE*], **9635** (2015).

[19] Gurobi Optimization Inc., "Gurobi optimizer reference manual." `http://www.gurobi.com` (2014).