OpenMPL: An Open Source Layout Decomposer
(Invited Paper)

Wei Li*, Yuzhe Ma*, Qi Sun*, Yibo Lin†, Iris Hui-Ru Jiang‡, Bei Yu*, David Z. Pan§
*The Chinese University of Hong Kong, †Peking University
‡National Taiwan University, §University of Texas at Austin
Email: {wli,byu}@cse.cuhk.edu.hk, yibolin@pku.edu.cn

***Abstract*—Multiple patterning lithography has been widely adopted in advanced technology nodes of VLSI manufacturing. As a key step in the design flow, multiple patterning layout decomposition (MPLD) is critical to design closure. Due to the $\mathcal{NP}$-hardness of the general decomposition problem, various efficient algorithms have been proposed with high quality solutions. However, with increasingly complicated design flow and peripheral processing steps, developing a high-quality layout decomposer becomes more and more difficult, slowing down the further advancement in this field. This paper presents OpenMPL [1], an open-source layout decomposition framework, with well-separated peripheral processing and the core solving steps. We demonstrate the flexibility of the framework with efficient implementations of various state-of-the-art algorithms, which enable us to reproduce most of the recent results on widely-recognized benchmarks. We believe OpenMPL can pave the road for developing layout decomposition engines and stimulate further researches on this problem.**

I. Introduction

Multiple patterning layout decomposition (MPLD) has been adopted to improve the lithography resolution. The key idea of MPLD is to assign features that are close to each other to different masks, such that within each mask, the features are far away enough to be printed with existing lithography techniques. MPLD can be divided into double patterning layout decomposition (DPLD), triple patterning layout decomposition (TPLD) and quadruple patterning layout decomposition (QPLD) according to the number of masks. This problem is difficult since it is a variation of the graph coloring problem, which is $\mathcal{NP}$-hard for $k \geq 3$, where k is the number of masks.

Fig. 1 is an example of TPLD, where three colors represent corresponding masks. Different from classical graph coloring problem, the layout decomposition problem has several unique characteristics. 1) Stitch: a polygon feature is allowed to be split into multiple overlapping segments to resolve coloring conflicts, as shown in the dashed edge of Fig. 1(b). 2) Special patterns: circuit layout follows some kinds of patterns due to the design styles, e.g., the alternative power and ground lines that may help to simplify the graph. 3) Complex rules: besides the widely-adopted spacing constraint for the same color, there are also other rules like the different color spacing constraints [2] related to the ordering of masks, and the pre-colored constraints where the colors of some features are pre-determined before decomposition. These characteristics make the problem special and require customized algorithms to solve it effectively and efficiently.

To achieve high efficiency and meanwhile maintain high solution quality, a variety of decomposition algorithms have been proposed. These algorithms can be roughly categorized into three types: mathematical programming and relaxation, graph-theoretical approaches and search-based approaches [3], [4]. Mathematical programming is to solve MPLD problem by formulating it into a mathematical programming model, such as integer linear programming (ILP) for DPLD [5]–[7] and TPLD [8]–[10]. Due to the $\mathcal{NP}$-hardness, relaxation techniques such as semidefinite programming (SDP) [8], linear programming (LP) [11] and a discrete relaxation method [12] are also proposed based on ILP. Another category is to directly perform color assignment based on a set of graph-theoretical algorithms, e.g., the maximal independent set (MIS) [13], the shortest-path [14], [15], and the fixed-parameter tractable (FPT) [16] algorithms. Search-based algorithms follow a divide-and-conquer principle with each sub-graph containing less than 20 nodes. Then a search procedure is applied to find the optimal solutions for small sub-graphs [8], [13], [17]–[21]. Besides the researches on single layout decomposition stage, recent work [22] pioneers a new direction which integrates layout decomposition and mask optimization seamlessly, achieving compelling results from a global view of the solution space.

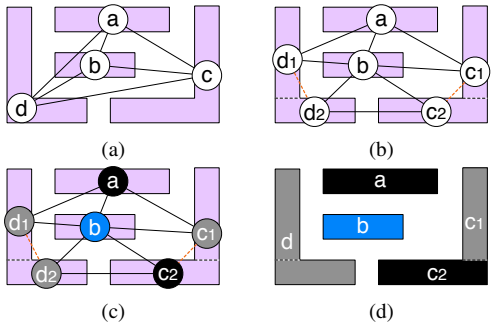

Fig. 1 An example of TPLD with stitches. (a) The constructed layout graph without stitch candidates insertion, which is actually a 4-clique and therefore not 3-colorable; (b) The constructed layout graph with stitch candidates insertion. Two stitch candidates are introduced and the original 4-clique is dismissed; (c) Coloring on the layout graph with stitch candidates insertion. No conflict in the final coloring result with only one stitch inserted; (d) The final decomposed layout with three masks (each color corresponds to one mask), the cost is 0.1 following Equation (1).

Besides the innovations to the core algorithms, many graph simplification techniques have been developed to reduce the problem size, such as independent component computation (ICC) [8], hide small degrees [8], [18], biconnected component analysis [6], [7], sub-K4 structure merging for TPLD [11].

To reduce the repeated effort in reimplementation of the whole decomposition framework and lower the bar of the research on MPLD, in this paper, we present a general and open-source framework, OpenMPL, as an open platform for developing MPLD algorithms. We carefully design the software architectures and APIs to decouple the innovations on the core optimization steps. For example, one can focus on developing novel graph simplification or decomposition techniques without worrying about the peripheral processing issues as the platform provides clean and well-defined APIs for kernel optimization engines. We also provide efficient implementations of widely-adopted graph simplification techniques and state-of-the-art layout decomposition algorithms which have been introduced above. We believe that this open platform paves the road for the development of MPLD engines and will stimulate more researches in the near future, eventually contributing to better manufacturability in advanced technology nodes.

978-1-7281-0735-6/19/$31.00 ©2019 IEEE

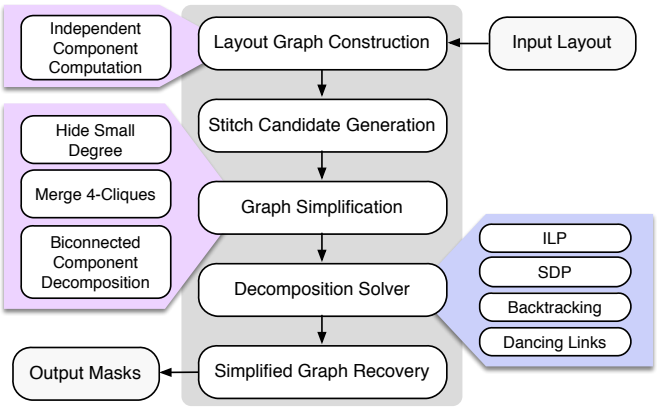

Fig. 2 The workflow of OpenMPL.

II. The OpenMPL Framework

OpenMPL is a general framework offering various decomposition algorithms and different simplification techniques. These methods are well embedded into the framework with unified interfaces. The framework operation flow is depicted in Fig. 2. Technical details are as follows.

A. Problem Formulation

Given an input layout specified by features in polygonal shapes, the layout can be translated into an undirected layout graph $G = (V, E)$, where every node $v_i \in V$ corresponds to one feature in layout and each edge $e_{ij} \in E$ is used to characterize the relationships between features. Considering conflict and stitch relationships, E is composed of this two kinds of edges, denoted by $E = \{CE \cup SE\}$, where SE is the set of stitch edges and CE is the set of conflict edges. The MPLD problem can be formulated as below:

$$\min_{\boldsymbol{x}} \quad \sum c_{ij} + \alpha \sum s_{ij}, \tag{1a}$$

$$\text{s.t.} \quad c_{ij} = (x_i == x_j), \qquad \forall e_{ij} \in CE, \tag{1b}$$

$$s_{ij} = (x_i \neq x_j), \qquad \forall e_{ij} \in SE, \tag{1c}$$

$$x_i \in \{0, 1, \ldots, k\}, \qquad \forall x_i \in \boldsymbol{x}, \tag{1d}$$

where x_i is a variable for the k available colors of the pattern v_i, c_{ij} is a binary variable representing conflict edge $e_{ij} \in CE$, s_{ij} stands for stitch edge $e_{ij} \in SE$, α is a user-defined parameter and is set as 0.1 by default in our framework to assign relative importance between the conflict cost and the stitch cost. If two nodes, x_i and x_j, within the minimal coloring distance are assigned the same color (i.e. $x_i == x_j$), then $c_{ij} = 1$. On the contrary, $s_{ij} = 1$ when two nodes connected by stitch edge are assigned different color (i.e. $x_i \neq x_j$). The objective function is to minimize weighted summation of the conflict number and the stitch number.

B. Design Principles

OpenMPL is designed for end-users, developers and researchers as a general platform for MPLD algorithms. Therefore, we emphasize usability, efficiency, and extensibility during development. The core design principles are highlighted as follows.

- Decoupled design stages. The implementation clearly separates different optimization stages in Fig. 2 such that the interdependency between them is minimized. In this way, developers can focus on verifying individual stages without worrying about cross-stage impacts.
- Graphs for communications among kernel stages. After layout graph construction, the graph simplification, decomposition solver, and the simplified graph recovery stages use pure graphs as input/output, without involving mask data. This design leads to well-defined and highly separatable core algorithms, making the framework highly extensible.
- Efficiency and generality for different mask data. As a mask layer can be a contact layer or a metal layer, the processing efficiency varies significantly for different types of layers. We design a general mask database with separate processing routines for contact layers and metal polygon layers for efficiency enabled by C++ polymorphism, since contact layers can be processed in a much simpler way.

C. Workflow and Functionalities

The workflow of OpenMPL is shown in Fig. 2. Firstly, one chip layout information (GDS) file is loaded and transformed into a layout graph, which is represented by a vector of rectangle pointers, where the rectangles are defined in Boost. Secondly, a stitch insertion process [23] is executed to generate a decomposed graph with stitches after simple ICC simplification. Then the decomposed graph is simplified by several simplification techniques, where some of them are implemented in third-party library Limbo [24]. After simplification, a coloring solver is called for each component in the decomposed graph to solve the component coloring problem. Finally, our framework recovers nodes removed in the simplification step and assigns legal color for each removed node. In the following sub-sections, we are going to introduce all of the functionalities in two crucial procedures of OpenMPL, layout simplification and decomposition.

Layout graph simplification techniques can be used to reduce the graph size and therefore reduce the computational complexity. We only need to deal with the smaller graph without affecting the final result. All of the four simplification techniques mentioned in Section I are supported in our framework, including `ICC`, `HIDE_SMALL_DEGREE`, `BICONNECTED_COMPONENT` and `MERGE_SUBK4`.

`ICC` is proposed based on the fact that there are many isolated clusters in a real layout such that `ICC` can break down the layout graph into several independent components. `HIDE_SMALL_DEGREE` temporarily removes the nodes whose degree is less than the number of color options in an iterative manner. `BICONNECTED_COMPONENT` simplifies the layout graph by removing all the bridge vertices. `MERGE_SUBK4` detects and merges specific structures whose number of edges is exactly one less than four-clique structures and thus is only applicable for TPLD. Different simplification techniques require different recovery methods. However, those nodes which are shared among different components may be assigned different colors after recovery. To tackle this, color rotation is implemented in our framework.

Graph color assignment is the most crucial step in the flow, which impacts the final coloring results directly. In graph color assignment, simplified graph is provided and each vertex in the graph should be assigned one color by specified algorithm. Users of our framework can specify which algorithm is adopted and the number of colors available. OpenMPL has supported all of the commonly-used algorithms in layout decomposition. The methodologies are briefly introduced in the following context.

- **Integer Linear Programming**: Solving Problem (1) is non-trivial. A widely used method to solve this problem is integer linear programming [5]–[8], which converts this problem into linear programming by binary encoding of vertex colors and replacing the color constraints with a set of inequality constraints. ILP can be easily extended to solve different coloring problems, including TPLD and QPLD. Our framework is based on the theory proposed in [8]. We use Gurobi [25], Lemon [26] and CBC [27] as the ILP solvers.
- **Semidefinite Programming**: The discrete integer programming

solving process of Equation (1) is $\mathcal{NP}$-hard, thus it may suffer from run-time overhead for practical designs. As shown in [8], [28], [29], the color assignment can be formulated as a vector programming and then relaxed and solved by semi definite programming, which can be solved in polynomial time. Given the solutions of SDP, a mapping process is used to map the solutions to coloring results. CSDP [30] is used as the SDP solver.

- **Backtracking**: Backtracking is a DFS fashion algorithm that is used to find solutions with constraints in the whole solution space. We also provide backtracking routine in OpenMPL. Though backtracking is widely used, its runtime performance is unsatisfactory for complicated graphs. Therefore we use it as a sub-solver to solve color assignment problem in simple sub-graphs .
- **Dancing Links**: Different from original dancing links and algorithm X, dancing links based algorithm for MPLD problem [20] concludes conflicts earlier by traversal in BFS manner, treating this problem as an exact cover problem. In OpenMPL, this solver is developed by ourselves instead of third-party libraries. We follow the statement in [20] and build a classical dancing links data structure in our implementation so that there are still much room for optimization.

OpenMPL also supports decomposition algorithms like MIS, LP, etc, which cannot solve the graph containing stitch edges while work well on stitch-free graphs. Due to page limit, we leave the details on the tool release page [1].

D. Additional Features

Some additional features are also supported for better usability, efficiency and extensibility. (1) OpenMPL supports **multi-threading** operation by OpenMP [31] and users can specify the number of threads. Graph components are solved in parallel and layout decomposition algorithms also support multi-threading computations. (2) We can identify all the possible positions of stitches through pattern projections [8] in **stitch insertion**, which is one of the most critical steps to parse a layout. One example of stitch is shown in Fig. 1. There are lots of candidate positions for stitch insertion, and we only choose some stitches from those candidates. One thing should be noted is that all the coloring algorithms provided in the framework share the identical stitch candidate generation procedure, which results in identical graph simplification results regardless of coloring algorithms. (3) In practice, a pattern in the layout may be a polygon or rectangle. Consequently, the storage may vary from case to case. OpenMPL provides a **shape-friendly** system considering this case and users can specify the shape, `POLYGON` or `RECTANGLE`, to guarantee the performance to avoid unnecessary calculations. For polygonal inputs, to simplify the storage structure design and save space, OpenMPL firstly decomposes the polygons to rectangles. After reading the whole input file, DFS is utilized to find connected components and re-union rectangles into polygons. For rectangle circuits, we directly store these patterns without further operations.

III. Experimental Results

We implement OpenMPL in C++ and use Boost [32] as the basic graphics library. All of the experiments are tested on an Intel Core 2.9 GHz Linux machine. We adapt scaled down and modified ISCAS benchmarks from [8] to conduct experiments, which are widely used in previous works. The minimum coloring spacing is set to $120nm$ for the first ten cases and as $100nm$ for the last five cases, as in [8], [13], [20]. The thread number is 8 and the graph simplification level is 3 (`ICC`, `HIDE_SMALL_DEGREE`, `BICONNECTED_COMPONENT`). Fig. 16 shows the decomposition result for case `C432` by dancing links based algorithm, which can be obtained in 0.008 seconds.

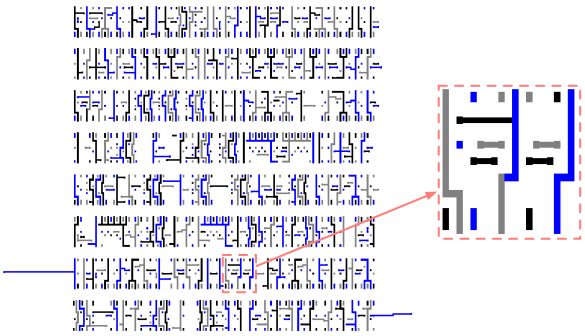

Fig. 3 Case `C432` decomposition result.

In this section, we only focus on the results of different decomposition algorithms with stitch insertion on TPLD problem due to page limit, which is more difficult to obtain optimal results compared with DPLD and stitch-free problems. More detailed results and discussions can be found in [1].

Firstly, we discuss the effectiveness of different decomposition algorithms implemented in our framework. Specially, MIS and LP are not listed since they cannot solve the problem with stitch edges. We compare the conflict and stitch number in coloring results and compute the corresponding cost following Equation (1), where parameter α is set to 0.1, thus the decomposition cost is calculated by $\text{cn\#} + 0.1 \cdot \text{st\#}$.

TABLE I compares the performance of implemented algorithms, where "ILP", "SDP", "Dancing links" and "Backtracking" represent OpenMPL with corresponding algorithms respectively. Columns "st#" and "cn#" denote the stitch number and the conflict number in the final coloring result, while the column "cost" is the decomposition cost.

For the cases whose runtime are more than 3600 seconds, we directly terminate the computations and do not measure the effectiveness. According to the results shown in the table, ILP-based algorithm achieves the best results no matter in the conflict number or the decomposition cost. Also, the backtracking algorithm can achieve same results with ILP-based algorithm in the cases which can be solved within 3600 seconds by backtracking algorithm. This is because all of the algorithms use the same simplification graphs as input and both ILP-based algorithm and backtracking can search for the optimal solution of Equation (1) in the search space. The result of SDP-based algorithm is close to the optimal solution, where the average decomposition cost is increased by 21.7%. The reason is that some stitches which can be applied to solve conflicts are ignored by SDP-based algorithm, such that the average stitch number of SDP-based algorithm is reduced by 8.4%. Another approaching but not optimal algorithm is the dancing links based algorithm, where the decomposition cost is increased by 5.4%. The reason is that we only insert exactly one stitch candidate in each polygon feature for speedup while there are some features whose stitches are more than one, which is ignored by our current dancing links implementation.

We also measure and compare the runtimes of different decomposition algorithms. The runtime result can be also found in TABLE I, where the column "time(s)" is the real time of decomposition in seconds instead of CPU time considering that we use multiple threads in our experiments. According to the results, backtracking-based algorithm faces a serious performance bottleneck when the input circuit becomes larger and even fails to finish the decomposition procedure within 3600 seconds on `S35932` and `S38584` circuits.

Besides the worst backtracking algorithm, the other three algorithms also show an obvious differences in runtime, where the dancing links based algorithm outperforms all of the other algorithms in most cases

TABLE I Decomposition Cost Comparison

Circuit	ILP				SDP				Dancing links				Backtracking			
	time(s)	st#	cn#	cost	time(s)	st#	cn#	cost	time(s)	st#	cn#	cost	time(s)	st#	cn#	cost
C432	0.045	4	0	0.4	0.021	4	0	0.4	0.008	4	0	0.4	0.023	4	0	0.4
C499	0.047	0	0	0	0.016	0	0	0	0.014	0	0	0	0.012	0	0	0
C880	0.053	7	0	0.7	0.018	7	0	0.7	0.008	7	0	0.7	0.010	7	0	0.7
C1355	0.050	3	0	0.3	0.022	2	1	1.2	0.013	3	0	0.3	0.035	3	0	0.3
C1908	0.039	1	0	0.1	0.010	1	0	0.1	0.008	1	0	0.1	0.018	1	0	0.1
C2670	0.064	6	0	0.6	0.025	6	0	0.6	0.016	6	0	0.6	0.024	6	0	0.6
C3540	0.067	8	1	1.8	0.035	8	1	1.8	0.016	8	1	1.8	1.981	8	1	1.8
C5315	0.074	9	0	0.9	0.029	9	0	0.9	0.012	10	0	1	0.034	9	0	0.9
C6288	0.679	205	1	21.5	0.137	200	6	26	0.082	203	5	25.3	125.562	205	1	21.5
C7552	0.105	21	1	3.1	0.049	20	2	4	0.040	22	1	3.2	0.143	21	1	3.1
S1488	0.184	2	0	0.2	0.020	2	0	0.2	0.005	2	0	0.2	0.013	2	0	0.2
S38417	0.848	55	19	24.5	0.327	50	24	29	0.248	56	20	25.6	2.388	55	19	24.5
S35932	2.756	40	44	48	0.910	24	60	62.4	0.513	45	46	50.5	>3600	NA	NA	NA
S38584	2.183	111	42	53.1	0.908	99	54	63.9	0.475	110	44	55	>3600	NA	NA	NA
S15850	2.148	98	34	43.8	0.852	90	42	51	0.470	102	35	45.2	2340.87	98	34	43.8
average	0.623	38.000	9.467	13.267	0.225	34.800	12.667	16.147	0.129	38.600	10.133	13.993	>647.32	NA	NA	NA
ratio	1.000	1.000	1.000	1.000	0.361	0.916	1.338	1.217	0.206	1.016	1.070	1.054	>1039.04	NA	NA	NA

and reduces the average runtime to 20.6% compared with the ILP-based algorithm. SDP-based algorithm is also faster than ILP-based algorithm where the average runtime is reduced by 63.9% but it is still almost twice as much as dancing links based algorithm.

IV. Conclusion and Future Work

OpenMPL is a general framework for multiple patterning layout decomposition problem and provides unified interfaces for layout decomposition algorithms and graph simplification speed-up techniques. Multi-threading, stitch insertion and shape-free feature are also supported in our framework. All of these features and some variables such as minimal distances are customizable and users can switch between these options freely. This version of OpenMPL has implemented most state-of-the-art algorithms and the results demonstrate the effectiveness and the efficiency. However, there are still much room for OpenMPL to improve. In the future we plan to integrate post-refinement, optimize our stitch insertion phase and develop acceleration techniques for the dancing links based algorithm.

V. Acknowledgment

This work is partially supported by Cadence Design Systems, The Research Grants Council of Hong Kong SAR (No. CUHK24209017), Taiwan MOST under Grant No. 106-2628-E-002-019-MY3, and US NSF under award No. 1718570.

References

[1] "OpenMPL," https://github.com/limbo018/OpenMPL.
[2] H.-Y. Chen, Y.-T. Hou, K. Yu-Hsiang, K.-H. Hsieh, R.-G. Liu, and L.-C. Lu, "Layout optimization for integrated circuit design," 2017, US Patent.
[3] D. Z. Pan, L. Liebmann, B. Yu, X. Xu, and Y. Lin, "Pushing multiple patterning in sub-10nm: Are we ready?" in *Proc. DAC*, 2015, pp. 197:1–197:6.
[4] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: A systematic review," in *Proc. VLSI-SoC*, 2017, pp. 1–6.
[5] Y. Xu and C. Chu, "GREMA: graph reduction based efficient mask assignment for double patterning technology," in *Proc. ICCAD*, 2009, pp. 601–606.
[6] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE TCAD*, vol. 29, pp. 939–952, June 2010.
[7] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *IEEE TCAD*, vol. 29, no. 2, pp. 185–196, Feb. 2010.
[8] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 3, pp. 433–446, March 2015.
[9] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, and D. Z. Pan, "A high-performance triple patterning layout decomposer with balanced density," in *Proc. ICCAD*, 2013, pp. 163–169.
[10] B. Yu, S. Roy, J.-R. Gao, and D. Z. Pan, "Triple patterning lithography layout decomposition using end-cutting," *JM3*, vol. 14, no. 1, pp. 011 002–011 002, 2015.
[11] Y. Lin, X. Xu, B. Yu, R. Baldick, and D. Z. Pan, "Triple/quadruple patterning layout decomposition via linear programming and iterative rounding," *JM3*, vol. 16, no. 2, 2017.
[12] X. Li, Z. Zhu, and W. Zhu, "Discrete relaxation method for triple patterning lithography layout decomposition," *IEEE TC*, vol. 66, no. 2, pp. 285–298, 2017.
[13] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," *IEEE TCAD*, vol. 33, no. 3, pp. 397–408, March 2014.
[14] H.-A. Chien, S.-Y. Han, Y.-H. Chen, and T.-C. Wang, "A cell-based row-structure layout decomposer for triple patterning lithography," in *Proc. ISPD*, 2015, pp. 67–74.
[15] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. D. F. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *Proc. ICCAD*, 2012, pp. 57–64.
[16] J. Kuang and E. F. Y. Young, "Fixed-parameter tractable algorithms for optimal layout decomposition and beyond," in *Proc. DAC*, 2017, pp. 61:1–61:6.
[17] B. Yu, G. Garreton, and D. Z. Pan, "Layout compliance for triple patterning lithography: an iterative approach," in *Proc. SPIE*, vol. 9235, 2014.
[18] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. DAC*, 2013, pp. 69:1–69:6.
[19] Y. Zhang, W.-S. Luk, H. Zhou, C. Yan, and X. Zeng, "Layout decomposition with pairwise coloring for multiple patterning lithography," in *Proc. ICCAD*, 2013, pp. 170–177.
[20] H.-Y. Chang and I. H.-R. Jiang, "Multiple patterning layout decomposition considering complex coloring rules," in *Proc. DAC*, 2016, pp. 40:1–40:6.
[21] W. Fang, S. Arikati, E. Cilingir, M. A. Hug, P. De Bisschop, J. Mailfert, K. Lucas, and W. Gao, "A fast triple-patterning solution with fix guidance," in *Proc. SPIE*, vol. 9053, 2014.
[22] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *Proc. ICCAD*, 2017, pp. 81–88.
[23] B. Yu, X. Xu, J.-R. Gao, Y. Lin, Z. Li, C. Alpert, and D. Z. Pan, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE TCAD*, vol. 34, no. 5, pp. 726–739, May 2015.
[24] "Limbo," http://yibolin.com/Limbo/docs/html/index.html.
[25] Gurobi Optimization Inc., "Gurobi optimizer reference manual," http://www.gurobi.com, 2016.
[26] "LEMON," http://lemon.cs.elte.hu/trac/lemon.
[27] "CBC," http://www.coin-or.org/projects/Cbc.xml.
[28] T. Matsui, Y. Kohira, C. Kodama, and A. Takahashi, "Positive semidefinite relaxation and approximation algorithm for triple patterning lithography," in *Proc. ISAAC*, 2014, pp. 365–375.
[29] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *Proc. DAC*, 2014, pp. 53:1–53:6.
[30] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, pp. 613–623, 1999.
[31] "OpenMP," http://www.openmp.org/.
[32] "Boost C++ Library," http://www.boost.org.