Layout Decomposition via Boolean Satisfiability

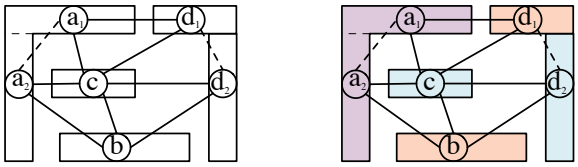
Hongduo Liu[1], Peiyu Liao[1], Mengchuan Zou[2], Bowen Pang[2] Xijun Li[2]
Mingxuan Yuan[2] Tsung-Yi Ho[1], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]Huawei Noah's Ark Lab

***Abstract*—Multiple patterning lithography (MPL) has been introduced in the integrated circuits manufacturing industry to enhance feature density as the technology node advances. A crucial step of MPL is assigning layout features to different masks, namely layout decomposition. Exact algorithms like integer linear programming (ILP) can solve layout decomposition to optimality but lacks scalability for very dense patterns. Approximation algorithms (e.g., linear programming, semi-definite programming) and heuristics (e.g., Exact-Cover) are capable of handling large cases but can only get inferior solutions. In this paper, we propose a new exact algorithm that tackles layout decomposition by solving a series of boolean satisfiability instances. Our algorithm can preserve optimality and achieve more than $4\times$ speedup compared to ILP. In addition, we provide an approximation algorithm by reformulating the layout decomposition to a bilevel optimization problem. Experiments show that our approximation algorithm can attain higher solution quality compared to SDP and heuristics within faster convergence.**

I. Introduction

Multiple patterning lithography (MPL) is a technique to overcome the lithographic limitations for manufacturing integrated circuits (ICs), and it has been explored to enable chipmakers to image designs at 20nm and below. MPL requires the decomposition of one layout onto multiple masks for better manufacturability. Specifically, the decomposition method should avoid assigning two features within a given distance to the same mask, which is considered a conflict or an assignment failure. Moreover, a conflict may be fixed by splitting a pattern into two sub-features with a stitch. As stitches can also introduce yield loss, we want to minimize both conflict and stitch during layout decomposition. Generally, the layout decomposition problem can be seen as a graph coloring problem, and Fig. 1 shows an example. Fig. 1(a) illustrates how a layout is transformed into a graph, namely a decomposition graph. The left-most and right-most features are divided into two sub-features by a stitch. As we can see, the graph has two kinds of edges: stitch edge and conflict edge. Stitch edges denoted by dashed lines connect two sub-features that are divided from the same layout pattern, and conflict edges denoted by real lines connect two sub-features that should be assigned to different masks. Fig. 1(b) depicts the decomposition results obtained by graph coloring, where sub-features in the same color are assigned to the same mask. The stitch edge between sub-feature d_1 and d_2 is activated, while the stitch edge between a_1 and a_2 is not. Besides, the decomposition introduces no conflict because the patterns connected by conflict edges belong to different masks.

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14208021).



(a) (b)

Figure 1 An example of transforming layout decomposition to graph coloring. (a) The graph representation of a layout; (b) Decomposition results can be obtained by graph coloring.

Double patterning layout decomposition (DPLD), which decomposes a layout onto two masks, can be solved in polynomial time [1]. Nevertheless, double patterning lithography is inadequate to ensure resolution as the minimum feature size keeps decreasing. A natural solution is to increase the number of exposure from two to three. Thus, triple pattering lithography layout decomposition (TPLD) is proposed to handle extremely dense and complex layouts. TPLD has been proven to be NP-hard [2], and this paper focuses on this scenario. Generally, layout decomposition algorithms can be categorized into mathematical programming and heuristic methods. Mathematical programming approaches [2]–[4] like ILP, SDP, and LP optimize an objective function subject to a set of constraints. Among these approaches, ILP can guarantee the optimality of the composition but suffers from unacceptable runtime overhead when facing dense layouts. SDP and LP show great scalability, but the solution quality is rather poor. Heuristic methods [5]–[7] expose similar problems as these relaxation methods. The drawbacks of existing methods motivate us to find a more efficient exact algorithm and a tighter approximation algorithm.

Reviewing the ILP formulation of layout decomposition problems, we find that all the decision variables are binary. Modern generic ILP solvers can handle general optimization problems with discrete variables. However, they may not fully exploit the boolean nature of decision variables and the special structures of the constraints for our layout decomposition problem, thus leaving the performance on the table. Instead of formulating layout decomposition into an integer program, we can see it as a sequence of satisfiable problems with optimality bound translated to an SAT clause. As a result, we can take advantage of the power of SAT solvers, which excel at handling pure boolean formulations.

Recall that the objective of the layout decomposition problem is to minimize both the conflict and stitch costs. To minimize conflict costs, we may increase the use of stitches. On the other

hand, optimizing the number of stitches may incur a higher conflict cost. These can be seen as two optimization problems nested with each other. Therefore, we can also treat layout decomposition as a bilevel optimization problem. The bilevel formulation enables us to decouple conflict minimization and stitch minimization, which are much easier to solve due to their particular structures.

Our contributions are summarized as follows:

- We propose a new exact algorithm to handle layout decomposition by solving a sequence of SAT problems. The algorithm can leverage the boolean nature of the decision variables in the cost optimization problem, thus being more efficient to solve.
- We provide a deep analysis of our SAT-based layout decomposer with an ILP-based decomposer. This analysis explains why our SAT-based method runs faster than the ILP-based method.
- We provide a new angle to see the layout decomposition problem as a bilevel optimization problem instead of a single-level one. Based on the bilevel reformulation, we propose an effective algorithm to solve the bilevel optimization problem in a hierarchical way.
- Experiments show that our new SAT-based exact algorithm can preserve optimality and achieve significant runtime speedup compared to ILP. Evaluations on dense layouts demonstrate that our approximation algorithm can get better decomposition results than SDP and heuristics in less time.

II. PRELIMINARIES

A. ILP Formulation for Triple Pattering Lithography Layout Decomposition

Given a layout composed of polygon features, we can construct an undirected graph $G = (V, CE \cup SE)$, where each node $v_i \in V$ represents a sub-feature (we also call a feature without stitch as a sub-feature), each edge $e_{ij} \in CE$ signalizes the conflict relationship between sub-feature i and j, and each edge $e_{ij} \in SE$ signalizes the stitch relationship between sub-feature i and j. Here V, CE, and SE are set of vertices, conflict edges, and stitch edges, respectively. The layout decomposition can be formulated as an integer linear program [8] as shown in program (1), where all the decision variables are binary.

Constraint (1b) applies for all sub-features. We use two binary variables x_{i1} and x_{i2} to encode the color of vertex i, and the color of vertex i can be either 00, 01, or 10. Color 11 is not allowed because we only consider triple patterning lithography. Constraints (1c) - (1f) model the conflict relationship between two features p_m and p_n. C_{mn} indicates whether there is a conflict between them. If there exists a sub-feature $r_i \in p_m$ and a sub-feature $r_j \in p_n$ connected by conflict edge e_{ij} are assigned to the same color, then $C_{ij} = 1$. Otherwise, $C_{ij} = 0$. Constraints (1g) - (1j) characterize the stitch relationship between two sub-features connected by stitch edge $s_{ij} \in SE$. Suppose two sub-features from the same feature are assigned to different colors, then the corresponding stitch variable s_{ij} is activated. The objective function (1a) is a weighted sum of

$$\min \sum_{r_i \in p_m, r_j \in p_n, c_{ij} \in CE} C_{mn} + \alpha \sum_{s_{ij} \in SE} s_{ij}, \tag{1a}$$

$$\text{s.t. } x_{i1} + x_{i2} \leq 1, \tag{1b}$$

$$x_{i1} + x_{i2} + x_{j1} + x_{j2} + C_{mn} \geq 1, \tag{1c}$$

$$x_{i1} - x_{i2} + x_{j1} - x_{j2} - C_{mn} \leq 1, \tag{1d}$$

$$-x_{i1} + x_{i2} - x_{j1} + x_{j2} - C_{mn} \leq 1, \tag{1e}$$

$$x_{i1} + x_{i2} + x_{j1} + x_{j2} - C_{mn} \leq 3, \tag{1f}$$

$$x_{i1} - x_{j1} + s_{ij} \geq 0, \tag{1g}$$

$$x_{i1} - x_{j1} - s_{ij} \leq 0, \tag{1h}$$

$$x_{i2} - x_{j2} + s_{ij} \geq 0, \tag{1i}$$

$$x_{i2} - x_{j2} - s_{ij} \leq 0. \tag{1j}$$

conflict cost $\sum C_{mn}$ and stitch cost $\sum s_{ij}$, where α controls the relative importance of the two costs. After the formulation is constructed, we can use an ILP solver to get an optimal coloring scheme.

B. Satisfiable Problems

A propositional logic formula is said to be in Conjunctive Normal Form (CNF) if it is a conjunction ("and") of disjunctions ("ors") of literals. A literal is either a boolean variable x or its negation $\neg x$. For example, $(p \vee q) \wedge (\neg q \vee \neg q)$ is a CNF, where $p, q, \neg p \neg q$ are all literals. The disjunctions $(p \vee q)$ and $(\neg q \vee \neg q)$ are also called clauses. The satisfiability (SAT) problem is to find a satisfying assignment to the boolean variables such that the CNF formula yields true. In other words, each clause should have at least one literal that is true under the assignment. If a satisfying assignment exists, the problem is said to be satisfied. Otherwise, the problem is unsatisfiable.

Boolean satisfiability is known to be NP-complete. No algorithm can solve all SAT problems in polynomial time. Despite this, scalable SAT solvers have been developed to solve SAT instances with tens of thousands of variables and millions of constraints effectively. One early breakthrough of SAT solving is Davis–Putnam–Logemann–Loveland (DPLL) [9], [10] algorithm. It assigns values to variables first and then backtracks when a contradiction is detected. Another milestone algorithm is the conflict-driven clausing learning (CDCL) algorithm, which augments DPLL with conflict analysis, clause learning, and backjumping. CDCL has been the core algorithm of most modern SAT solvers.

C. Bilevel Optimization

A bilevel optimization problem reads

$$\min_{x \in X, y} \; F(x, y) \tag{2a}$$

$$\text{s.t. } G(x, y) \geq 0, \tag{2b}$$

$$y \in S(x), \tag{2c}$$

where $S(x)$ is the set of optimal solutions of the x-parameterized problem

$$\min_{y \in Y} \; f(x, y) \tag{3a}$$

$$\text{s.t. } g(x, y) \geq 0. \tag{3b}$$

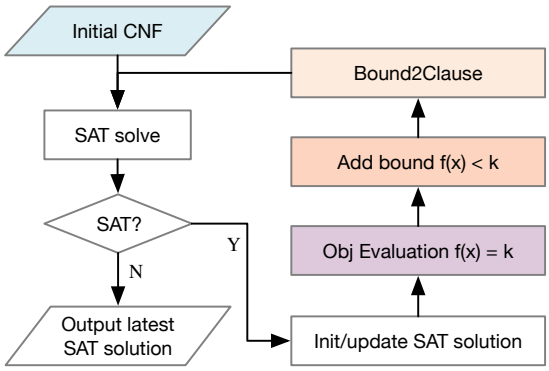


Figure 2 The flow of our SAT-based exact algorithm.

Problem 2 is called the upper-level (or the leader's) problem, and problem 3 is called the lower-level (or the follower's) problem, which is parameterized by x. Moreover, the variables $x \in \mathbb{R}^{n_x}$ are the upper-level variables and $y \in \mathbb{R}^{n_y}$ are lower-level variables. $F, f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ are objective functions and $G : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^p$, $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^q$ are used to describe the upper-level and lower-level constraints. The sets $X \subseteq \mathbb{R}^{n_x}$ and $Y \subseteq \mathbb{R}^{n_y}$ may denote additional constraints like integrality.

A bilevel optimization problem can be solved through single-level reduction. One classical solution is to replace the lower-level optimization problem with its Karush-Kuhn-Tucker (KKT) conditions if the lower-level problem is convex and sufficiently regular [11]. Other methods include descent methods [12], penalty function methods [13], genetic algorithms [14], etc.

III. Optimal Layout Decomposition through SAT

This section shows how to translate layout decomposition into instances of SAT problems based on the original ILP formulation to exploit the binary nature of decision variables better.

A. The Whole Flow

The whole flow of our SAT-based method is shown in Fig. 2. To begin with, we translate all the constraints in problem 1 into CNF clauses. Then, we will solve the SAT problem without considering the objective function, and we can get an initial satisfying assignment of all decision variables. Consequently, we can get an upper bound of the objective value by evaluating the objective function using the satisfiable solution. In the next step, We need to introduce the optimality bound into SAT solving to find a better solution. If x is the satisfying assignment and $f(x) = k$ is the evaluation result on objective function $f(\cdot)$, the optimality bound $f(x) < k$ should be added to the next SAT instance. However, an SAT solver only accepts a CNF as input. Thus, we need to convert the optimality bound into a CNF clause. If the new SAT instance after adding optimality bound is satisfiable, then we have found a better solution. Otherwise, we have demonstrated that k is the optimal value, and the latest satisfiable solution gives the optimal solution. Fig. 3 gives a concrete example of our SAT-based decomposer. The initial CNF gives a coloring solution that has two conflicts. After solving some SAT instances, we can get a coloring scheme without any conflict.

B. The Construction of Initial Clauses

The decision variables in problem 1 can only take value from $\{0, 1\}$. Unlike constraints that admit unrestricted integer variables, we can convert them to CNF clauses. The easiest one to convert is a cardinality constraint with the right-hand side equal to 1. More specifically, constraint $x_1 + x_2 + \ldots + x_k \geq 1$ is equal to a CNF clause $(x_1 \vee x_2 \vee \ldots x_k)$. Constraint $x_1 + x_2 + \ldots + x_k \geq 1$ is satisfied iff at least one of the variables is set to 1. Equivalently, $(x_1 \vee x_2 \vee \ldots x_k)$ is true iff at least one of the literal is true. Fortunately, all the constraints posed in problem 1 belong to this kind of constraint after a few trivial reformulations. For example, inequality (1b) can be transformed into a CNF clause through the following steps:

- Let the $\leq$ be $\geq$ by multiplying -1 on both sides of the inequality. We have $-x_{i1} - x_{j1} \geq -1$.
- Replace x_{i1}, x_{j1} by $-(1 - \overline{x_{i1}}), -(1 - \overline{x_{j1}})$ respectively. We can get $-(1 - \overline{x_{i1}}) - (1 - \overline{x_{j1}}) \geq -1$. Here $\bar{x}$ is the negation of x, and it is easy to see $\bar{\bar{x}} = x$.
- Reorganize the terms we have $\overline{x_{i1}} + \overline{x_{j1}} \geq 1$, which can be represented by a CNF clause $(\overline{x_{i1}} \vee \overline{x_{j1}})$.

Similarly, other constraints can be transformed into a set of CNF clauses. Then, problem 1 can be rewritten as

$$\min \sum_{r_i \in p_m, r_j \in p_n, c_{ij} \in CE} C_{mn} + \alpha \sum_{s_{ij} \in SE} s_{ij}, \tag{4a}$$

$$\text{s.t. } (\overline{x_{i1}} \vee \overline{x_{i2}}), \tag{4b}$$

$$\wedge (x_{i1} \vee x_{i2} \vee x_{j1} \vee x_{j2} \vee C_{mn}), \tag{4c}$$

$$\wedge (\overline{x_{i1}} \vee x_{i2} \vee \overline{x_{j1}} \vee x_{j2} \vee C_{mn}), \tag{4d}$$

$$\wedge (x_{i1} \vee \overline{x_{i2}} \vee x_{j1} \vee \overline{x_{j2}} \vee C_{mn}), \tag{4e}$$

$$\wedge (\overline{x_{i1}} \vee \overline{x_{i2}} \vee \overline{x_{j1}} \vee \overline{x_{j2}} \vee C_{mn}), \tag{4f}$$

$$\wedge (x_{i1} \vee \overline{x_{j1}} \vee s_{ij}), \tag{4g}$$

$$\wedge (\overline{x_{i1}} \vee x_{j1} \vee s_{ij}), \tag{4h}$$

$$\wedge (x_{i2} \vee \overline{x_{j2}} \vee s_{ij}), \tag{4i}$$

$$\wedge (\overline{x_{i2}} \vee x_{j2} \vee s_{ij}). \tag{4j}$$

We observe that the constraints' nice properties enable us to seamlessly translate them into CNF clauses without introducing additional variables. Firstly, we can avoid tedious and time-consuming transformations. Also, the size of the derived CNF stays tractable. As the initial CNF is always satisfiable, we can find an optimal solution through our algorithm, as shown in the following theorem.

Theorem 1. *The algorithm can always find an optimal solution.*

Proof. Let all x_{i1}, x_{i2} be false, and set all C_{mn}, s_{ij} be true, the initial CNF is satisfied, which means we can at least find a solution. Then, an optimal solution can be found by strengthening the optimality bound until the derived SAT problem is unsatisfiable. □

C. Optimality Bound to CNF Clause

In the previous subsection, we have shown how to translate the initial constraints into CNF clauses. To allow the SAT solver

to be capable of optimization, we also need to translate the optimality bound to a CNF clause. This is more subtle than what we have done to the original constraints. In some literature, linear constraints over binary variables are also called pseudo-boolean (PB) constraints. They can be translated into clauses that an SAT solver can handle. [15] proposes three different techniques to finish the translation. Firstly, they convert each constraint into a single-output circuit and then translate all the circuits to clauses by a variation of the Tseitin transformation [16]. The circuit can be a BDD, a network of adders, or a network of sorters. We can apply these techniques to our framework without further modifications. Before the transformation, we should make the coefficients of the objective function integral. When α is set to 0.1, we can use an alternative objective function $10\sum C_{mn} + \sum s_{ij}$.

IV. Approximate Layout Decomposition Approach

A. Reformulation

The layout decomposition problem can also be formulated as a bilevel optimization problem. The upper-level optimization problem is given by

$$\begin{aligned} \min_{C,s} \quad & \sum_{r_i \in p_m, r_j \in p_n, c_{ij} \in CE} C_{mn} + \alpha \sum_{s_{ij} \in SE} s_{ij}, \\ \text{s.t.} \quad & \text{constraint (1b) – constraint (1f)}, \\ & s \in S(C), \end{aligned}$$

where $S(C)$ is the set of optimal solutions of the C-parameterized problem

$$\min_{s} \quad \sum_{s_{ij} \in SE} s_{ij}, \tag{5a}$$

$$\text{s.t.} \quad \text{constraint (1b) – constraint (1j)}. \tag{5b}$$

We want to minimize the weighted sum of conflict and stitch costs in the upper-level optimization problem. $s \in S(C)$ indicates that for every assignment of conflict variables, the stitch cost should be optimal.

B. Approximation Algorithm

One way to solve the bilevel optimization problem is through single-level reduction. In the layout decomposition problem, the reduced single-level problem is shown exactly as problem 1, which can preserve optimality. However, neither the ILP-based and SAT-based algorithms can get an optimal solution for very large and dense layouts in an acceptable time. We need an approximation algorithm to obtain good results quickly for these cases. Another approach to solving a bilevel optimization problem is nested optimization, which solves the lower-level optimization problem corresponding to every upper-level member. Once the upper-level variables are fixed, we can solve the lower-level optimization to optimality. Then, the assignments of upper-level variables and the derived lower-level solution give an upper bound to the bilevel optimization. Theoretically, it can converge to the optimal solution by enumerating every feasible upper-level variable and solving the lower-level problem accordingly.

One critical problem is generating good candidates for the upper-level variables C_{mn}. This is not trivial. Firstly, some assignments of the upper-level variables can make the upper-level problem infeasible, i.e., there is no coloring scheme that can satisfy all the upper-level constraints. Besides, if we need multiple iterations to converge, we may need to solve the lower-level problem many times, which can be time-consuming. For these reasons, we get the assignments of upper-level variables by solving the upper-level problem ignoring the lower-level variables. In the layout decomposition problem, it can be seen as only considering conflict minimization. Once all conflict variables C_{mn} are fixed, we can perform stitch cost minimization. To be more specific, the first optimization problem we need to solve is

$$\min_{C} \quad \sum_{r_i \in p_m, r_j \in p_n, c_{ij} \in CE} C_{mn}, \tag{6a}$$

$$\text{s.t.} \quad \text{constraint (1b) – constraint (1f)}. \tag{6b}$$

In our approximation algorithm, we also use the SAT-based framework to solve upper- and lower-level problems. Once all the C_{mn} have been fixed, the initial CNF for the lower-level problem can be greatly simplified. For example, (4c) can be rewritten as $(x_{i1} \vee x_{i2} \vee x_{j1} \vee x_{j2})$ if $C_{mn} = \texttt{false}$ is derived in the first-stage optimization. If the assignment $C_{mn} = \texttt{true}$ is obtained, (4c) can be deleted from the CNF because the clause has already been satisfied. The same rule can also be applied to (4c)-(4e) to simplify the lower-level optimization problem.

V. Experimental Results

A. Experimental Setup

The experiments are performed on a Linux machine with eight 3.6GHz Intel Xeon CPUs and 16G DRAM. We rely on the open-source framework OpenMPL for benchmark parsing, stitch insertion, graph simplification, etc. Our SAT-based optimization framework is mainly based on MiniSat+ [15], a solver capable of translating pseudo-boolean constraints to CNF clauses and SAT solving. We reorganized the codes into a callable dynamic library to better integrate them into OpenMPL. We only use one thread to see the runtime difference of various decomposers.

We conduct experiments on the ISCAS and ISPD19 benchmarks. The first six circuits of ISPD19 benchmarks are relatively easy to handle and have been adopted in [8]. They are used to show the performance of our exact algorithm. The last three benchmarks are denser and are used to show the effectiveness of our approximation algorithms. We also follow the same rule to set minimum coloring distance, stitch weight, and graph simplification level.

B. Evaluation of our SAT-based exact algorithm

The decomposition results by our exact algorithm on ISCAS benchmarks are listed in TABLE I. It shows that our approach can obtain optimal solutions as ILP but with faster convergence. The decomposition results using our exact algorithm on the first six ISPD19 benchmarks are listed in TABLE II. We can see that our SAT-based decomposer can achieve more than $4\times$

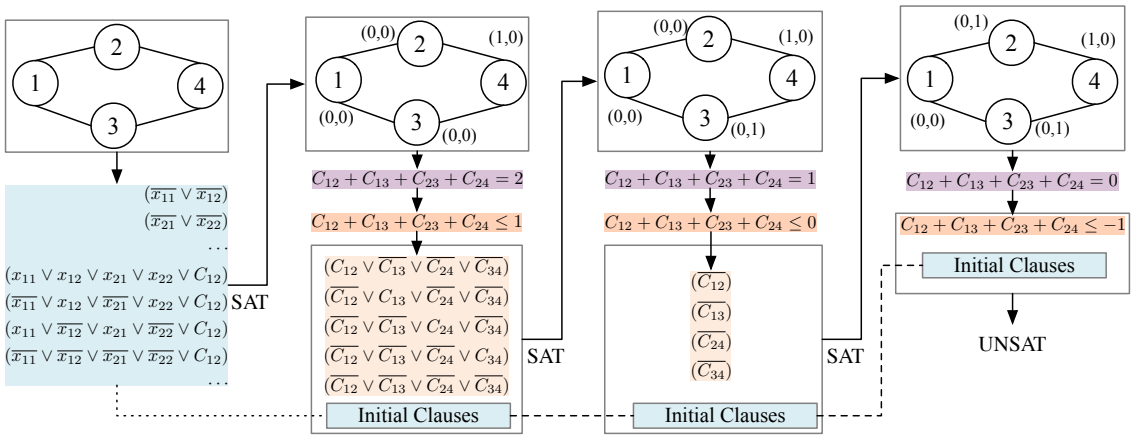


Figure 3 A toy example of our SAT-based decomposer.

Table I Results on ISCAS benchmarks. "RT" indicates runtime.

Circuit	ILP [8]		SDP [2]		EC [7]		Ours	
	Cost	RT (s)	Cost	RT (s)	Cost	RT (s)	Cost	RT (s)
C432	**0.4**	0.087	0.4	0.027	0.4	0.021	**0.4**	0.029
C499	**0.0**	0.081	0.0	0.028	0.0	0.025	**0.0**	0.030
C880	**0.7**	0.083	0.8	0.032	0.7	0.026	**0.7**	0.034
C1355	**0.3**	0.062	0.3	0.039	0.3	0.036	**0.3**	0.044
C1908	**0.1**	0.063	0.1	0.054	0.1	0.051	**0.1**	0.056
C2670	**0.6**	0.109	0.6	0.084	0.6	0.079	**0.6**	0.090
C3540	**1.8**	0.153	1.8	0.112	1.8	0.100	**1.8**	0.123
C5315	**0.9**	0.217	0.9	0.147	0.9	0.130	**0.9**	0.156
C6288	**21.4**	2.999	27.3	0.434	21.4	0.300	**21.4**	0.606
C7552	**2.3**	0.402	2.3	0.235	3.1	0.208	**2.3**	0.255
S1488	**0.2**	0.082	0.2	0.051	0.2	0.043	**0.2**	0.057
S38417	**24.4**	2.352	31.6	1.445	24.4	0.771	**24.4**	2.072
S35932	**48.0**	6.451	66.0	4.248	48.7	2.034	**48.0**	6.069
S38584	**47.6**	6.533	58.5	4.195	47.7	2.216	**47.6**	5.915
S15850	**43.7**	5.854	56.3	3.821	44.0	2.075	**43.7**	5.415
Avg. Ratio	**1.00**	1.79	1.11	0.85	1.02	0.67	**1.00**	1.00

speedup without sacrificing solution quality compared with the ILP-based decomposer. When compared to SDP relaxation and EC, our algorithm can achieve much better solution quality in a comparable time.

C. Analysis of the runtime improvement

In this subsection, we provide an analysis of the remarkable speedup of our SAT-based decomposer over traditional ILP-based decomposer. There are three main reasons. The first one is that the original constraints can be easily translated into CNF clauses without tedious transformation and introducing additional variables. Another reason is that the clause-learning-based algorithm and efficient backtracking in modern SAT solvers have enabled fast SAT solving for large-scale instances.

The last reason is related to the way they get optimal solutions. In general, ILP-based and SAT-based decomposers discover an optimal solution by finding a tighter and tighter upper bound that eventually converges to the optimal value. However, the condition they claim that an optimal solution has been found is quite different. The ILP solver will only assert optimality if the gap between the best and the current upper bound equals zero. This best bound is obtained by taking the minimum optimal objective values of all of the current leaf nodes in the branch-and-bound tree. The current bound is the objective value of the best-known feasible solution, which can be obtained by heuristics or branching. It often happens that

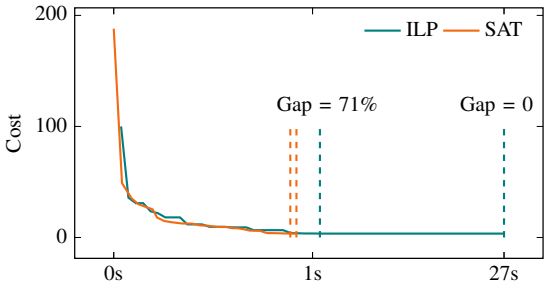


Figure 4 A case study on convergence of ILP and SAT-based decomposers.

the ILP solver has already found an optimal solution, but the optimality gap is not zero. Therefore, the ILP solver still needs to do branching and solve linear program relaxations until the optimality gap reaches zero, which can take a long time. For the SAT-based decomposer, we are confident that we have already found an optimal solution if the SAT instance with a better objective value is unsatisfiable. This can be done much more effectively than solving many linear programs.

Fig. 4 shows the convergence of ILP and SAT-based decomposers on a decomposition graph from circuit `test6_102`. The two dashed lines indicate the time an optimal solution is found and the time the optimality is proven. The SAT-based method can prove optimality only after a short period by detecting an unsatisfiable instance. However, the ILP-based method takes a long time (around 25s) to reduce the optimality gap from 71% to 0, even though it only takes roughly 2s to find an optimal solution.

D. Evaluation of our approximation algorithm

Our approximation algorithm aims at accelerating the decomposition of dense layouts. To verify the effectiveness of our proposed algorithm, we use another three dense layouts from ISPD19 benchmarks, where the decomposition graphs after simplification are still very large, and some are unsolvable for ILP in one hour. The results marked by * in TABLE II show the comparison between our approximation algorithm to previous methods. The table indicates that our method can get lower-cost solutions and converge faster than SDP and EC.

Table II Layout decomposition results on ISPD19 benchmarks. "RT" indicates runtime.

Circuit	ILP [8]		SDP [2]		EC [7]		Ours	
	Cost	RT (s)	Cost	RT (s)	Cost	RT (s)	Cost	RT (s)
test1_100	**242.9**	56.24	297.7	**2.61**	390.5	9.51	**242.9**	5.73
test5_101	**452.0**	78.32	549.8	**5.60**	629.8	16.73	**452.0**	10.65
test6_102	**153.4**	188.56	191.7	**35.58**	344.1	59.21	**153.4**	69.79
test8_100	**6005.9**	82.13	6206.2	**32.27**	6245.6	34.39	**6005.9**	37.55
test9_100	**9223.3**	128.91	9532.4	**52.72**	9664.0	56.08	**9223.3**	60.50
test10_100	**10449.5**	244.93	10910.1	**85.52**	11130.6	128.96	**10449.5**	103.32
Avg. Ratio	**1.00**	4.43	1.13	**0.67**	1.40	1.19	**1.00**	1.00
test1_101*	**71.8**	2370.45	107.4	19.65	168.7	71.51	75.1	**6.87**
test2_100*	**5236.7**	12941.22	7259.4	187.31	9893.7	1404.07	5391.3	**124.58**
test2_102*	213.4	7810.46	526.7	304.76	593.9	2722.24	**211.8**	**149.37**
Avg. Ratio	**0.98**	167.07	1.75	2.13	2.30	13.30	1.00	**1.00**

* Our approximation algorithm is enabled. For ILP, we set the timeout to 3600s.

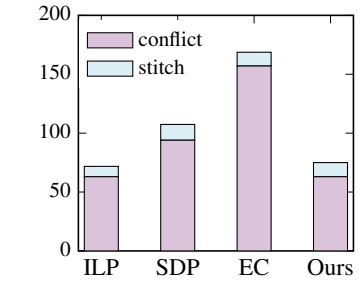


Figure 5 Cost breakdown of various algorithms on test1_101.

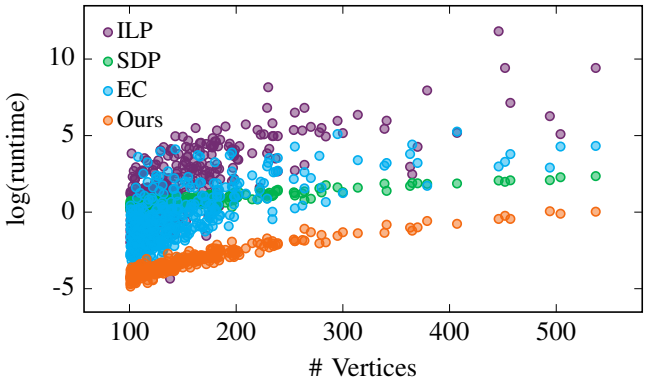


Figure 6 Solving time of decomposition graphs with different scales from circuit test2_100. We take the logarithm of runtime because the range is too big.

We also provide a cost breakdown of different decomposition algorithms, as shown in Fig. 5. It reveals that our approximation algorithm can fulfill optimal conflict cost with a slightly higher stitch cost than ILP. For circuit test2_102, our algorithm can get a better solution than ILP because ILP cannot find an optimal solution for some graphs in 3600s. To see the scalability of our approximation algorithm, we plot the solving time of decomposition graphs with different scales from circuit test2_100, which is shown in Fig. 6. As the graphs get larger, our approximation algorithm remains effective, while the runtime of other methods can grow drastically.

VI. Conclusion

In this paper, we propose a new exact algorithm that tackles layout decomposition by solving a series of boolean satisfiability instances. Our algorithm can preserve optimality and achieve significant speedup compared to ILP. Besides, we provide a new angle to treat the layout decomposition as a bilevel optimization problem. Based on the reformulation, we present a tighter approximation algorithm by solving a two-stage optimization problem. Experiments show that our approximation algorithm converges faster than traditional SDP and heuristics but obtains much lower decomposition costs.

References

[1] X. Tang and M. Cho, "Optimal layout decomposition for double patterning technology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 9–13.

[2] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 3, pp. 433–446, 2015.

[3] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition approaches for double patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 6, pp. 939–952, 2010.

[4] Y. Lin, X. Xu, B. Yu, R. Baldick, and D. Z. Pan, "Triple/quadruple patterning layout decomposition via linear programming and iterative rounding," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, vol. 16, no. 2, p. 023507, 2017.

[5] S.-Y. Fang, Y.-W. Chang, and W.-Y. Chen, "A novel layout decomposition algorithm for triple patterning lithography," in *ACM/IEEE Design Automation Conference (DAC)*, 2012, pp. 1185–1190.

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

[7] I. H.-R. Jiang and H.-Y. Chang, "Multiple patterning layout decomposition considering complex coloring rules and density balancing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 12, pp. 2080–2092, 2017.

[8] W. Li, Y. Ma, Q. Sun, L. Zhang, Y. Lin, I. H.-R. Jiang, B. Yu, and D. Z. Pan, "Openmpl: An open-source layout decomposer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 11, pp. 2331–2344, 2020.

[9] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[10] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[11] J. F. Bard and J. E. Falk, "An explicit solution to the multi-level programming problem," *Computers & Operations Research*, vol. 9, no. 1, pp. 77–100, 1982.

[12] L. Vicente, G. Savard, and J. Júdice, "Descent approaches for quadratic bilevel programming," *Journal of Optimization theory and applications*, vol. 81, no. 2, pp. 379–399, 1994.

[13] Y. Ishizuka and E. Aiyoshi, "Double penalty method for bilevel optimization problems," *Annals of Operations Research*, vol. 34, no. 1, pp. 73–88, 1992.

[14] Y. Yin, "Genetic-algorithms-based approach for bilevel programming models," *Journal of Transportation Engineering*, vol. 126, no. 2, pp. 115–120, 2000.

[15] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into sat," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 1–26, 2006.

[16] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.