A Matching Based Decomposer for Double Patterning Lithography *

Yue Xu and Chris Chu
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011-3060, USA
{yuexu,cnchu}@iastate.edu

ABSTRACT

Double Patterning Lithography (DPL) is one of the few hopeful candidate solutions for the lithography for CMOS process beyond $45nm$. DPL assigns the patterns less than a certain distance from each other on each layer onto two masks instead of one mask in traditional lithography. In this paper, we prove that the conflict graph used to model DPL conflicts in layout is a planar graph. Based on the planarity, we propose a new face merging based framework which formulates DPL decomposition as a problem of pairing odd faces to simultaneously minimize the number of stitches generated and conflicts to eliminate. We employ partitioning and simplification techniques to reduce the problem size and use an $O(n^3)$ time maximum weighted matching algorithm to generate an optimal DPL decomposition.

Categories and Subject Descriptors: B.7.2 [Integrated Circuits]: Design Aids — layout ; J.6 [Computer-Aided Engineerin] : Computer-Aided Design

General Terms: Algorithms, Design

Keywords: Double Patterning Lithography, Planar Graph, Matching Algorithm

1. INTRODUCTION

As one of the most promising candidate technology for lithography beyond $45nm$, Double Patterning Lithography (DPL) assigns patterns on each layer onto two masks and carries out two rounds of independent pattern transfer to form patterns on wafers. In order to increase the minimum pitch for each lithography exposure, DPL requires a pair of patterns assigned onto different masks if the minimum distance between the pair is less than a DPL threshold [1] [2] [3] [4]. Commonly, the threshold is set to be twice the minimum feature size of the desired process. We use Manhattan distance as the measuring metric just like some previous works, such as [4] and [5].

*This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998.

ISPD'10, March 14–17, 2010, San Francisco, California, USA.

We call a pair of patterns conflicting or has conflict if the minimum distance between the two patterns is less than the DPL threshold. A DPL decomposer needs to assign conflicting patterns onto different masks. Sometimes, layout cannot be decomposed unless patterns are sliced up and has the sliced parts assigned onto different masks. When the sliced parts in a single pattern are formed by different rounds of pattern transfer, their touching boundary forms stitch. With stitch generation, DPL decomposers can increase the flexibility of DPL decomposition. This improvement of flexibility comes at the price of lower yield. When patterns from two rounds of pattern transfer process merge and form one connecting pattern, pinching or bridging effects stem up and tend to reduce yield. So DPL decomposers generally try to minimize the number of stitches [5] [6].

Even with the flexibility introduced by creating stitches, layouts generated by traditional back end tools are usually DPL incompatible because the tools simply ignore the DPL issues. We define DPL incompatibility (incompatibility for short) for a layout as its lack of a valid DPL decomposition even if all possible stitches are considered. DPL decomposer needs to send DPL conflicts involved in incompatibility back to designers for layout modification. Fewer DPL conflicts involved in incompatibility indicates less effort and time spent on the modification. So in this paper, we extend the scope of DPL decomposition problem by suggesting a minimum number of DPL conflicts to eliminate. If the layout modification eliminates the suggested conflicts, the resulting layout will be DPL compatible. We notice that the set of stitches that decomposers choose may affect the number of conflicts to eliminate, so we simultaneously reduce the number of stitches generated and conflicts to eliminate.

A lot of previous DPL decomposers simply ignore DPL incompatibility [7] [8] [9] [14]. Kahng et al. [9] use iterative searching of odd cycles and generate candidate stitches heuristically. They neglect incompatibility issues in their ILP based stitch selection. The work is later extended to [10], which considers layout modification by incorporating PSM layout modification works [11] [12] into DPL decomposer. However, PSM problem does not have the flexibility to slice patterns and generate stitches. The application of those techniques only leads to sub-optimal results. Yuan et al. [13] slice patterns into numerous cells with the feature size as side length. They formulate an ILP to assign the cells onto masks to reduce a weighted sum of the number of stitches generated and conflicts to eliminate. Although the work considers the minimization of the number of conflicts to eliminate, its ILP formulation is too slow for practical

designs. It uses some partitioning heuristic to speed up the decomposition but such heuristic destroys the optimality of the decomposer. To speed up the pre-slice and ILP based stitching framework, [14] proposes conflicting pattern clustering and several simplification techniques to carry out partial mask assignment decisions and reduce the problem size of the final ILP.

In this paper, we propose a new matching based DPL decomposer. Our decomposer optimally solves the minimization of a weighted sum of the number of stitches generated and conflicts to eliminate (i.e., the same objective as [13]). Our key contributions include:

1. We prove that the *Conflict Graph* (CG) used to model DPL conflicts is planar. The planarity of CG is crucial in enabling us to use a matching based approach.
2. We find that both stitching and conflict elimination merge neighboring faces in CG. We create a structure called *Face Graph* (FG) to model face merging.
3. We propose a new framework for DPL decomposition that optimally pairs up all odd nodes in FG to eliminate all odd cycles in CG.
4. We prove that the odd-node pairing problem in FG can be reduced to a set of smaller pairing problems by partitioning FG into subgraphs. This reduction greatly speeds up our decomposer.
5. We transform each subgraph of FG into a complete graph that stores the costs of pairing odd nodes. We formulate the odd-node pairing problem for each subgraph as a minimum weighted perfect matching problem.
6. We use an optimal and polynomial time algorithm for minimum weighted perfect matching problem in complete graph. The matching solution corresponds to an optimal pairing of odd nodes in FG.

Comparing to odd-cycle-searching based decomposers [9] [10], our decomposer can achieve the minimum number of stitches generated and conflicts to eliminate. Comparing to [13], it does not slice up patterns to minuscule cells and avoids formulating an ILP. It does not need partitioning heuristic that affects the optimality of solutions to speed up the decomposer.

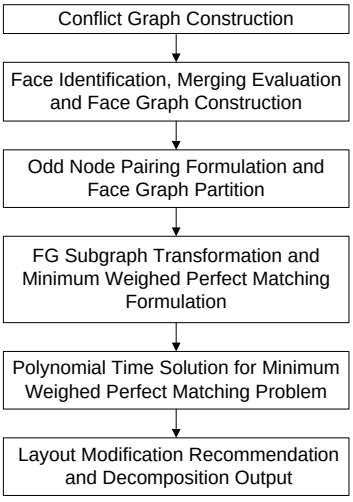


Figure 1: DPL Decomposer Flow

The flow of our decomposer is shown in Fig. 1. It first constructs the CG from layout, identifies faces and evaluates the feasibility of merging neighboring faces. From the merging feasibility, it builds up the FG and carries out partitioning and simplification. Finally it formulates matching problems and solves them to find the optimal stitch generation and conflict elimination solution.

The rest of this paper is organized as follows. Sec. 2 presents the proof that Conflict Graph is planar. Sec. 3 introduces the general concepts and formulation of our face-merging based decomposer. Sec. 4 shows the partitioning and simplification techniques along with the solution for the DPL decomposition problem. The experimental results are provided in Sec. 5 and we conclude in Sec. 6.

2. PLANARITY OF CONFLICT GRAPH

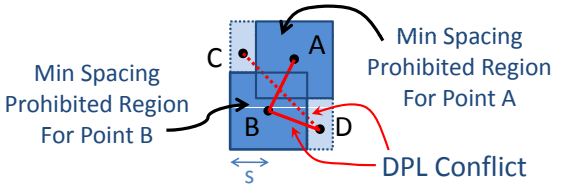


Figure 2: Proof of Planarity

In Conflict Graph, we create one node for each pattern on one layer in the layout and one edge in between two nodes if the minimum Manhattan distance between two corresponding patterns is less than two times the minimum feature size. In other words, we use Manhattan distance as the metric for DPL conflict. To prove that CG is planar, we first consider DPL conflicts induced from abstract points instead of patterns and prove the following lemma:

LEMMA 1. *Conflict edges induced from a set of points can only intersect at the endpoints of the conflict edges, under the condition that points are separated by the minimum spacing rule and the DPL threshold is less than or equal to two times the minimum spacing.*

Proof: We defined a DPL conflict for two points if their Manhattan distance is less than the DPL threshold in Sec. 1. We create a conflict edge for the DPL conflict. It is obvious that conflict edges induced by three points may only intersect at the endpoints of the edges. As shown in Fig. 2, the conflict edges $A \sim B$ and $B \sim D$ intersect at a shared endpoint B.

So we focus on conflict edges between two pairs of DPL conflicting points. In Fig. 2, we try to force conflict edges $A \sim B$ and $C \sim D$ to intersect. We denote the threshold of DPL conflict as t. Without loss of generality, we assume that B is within the lower left quadrant of A. Since A and B are conflicting, we have the following relationship for the coordinates of A and B:

$$x_A - x_B + y_A - y_B \quad < \quad t \tag{1}$$

In order to have conflict edge $C \sim D$ intersecting $A \sim B$ between points A and B, points C and D must reside in the lighter colored regions in Fig. 2. Otherwise, C and D will be too far away to be DPL conflicting or conflict edge $C \sim D$ will not intersect $A \sim B$ at all. Without loss of generality, we assume that C lies in the region left to A and D lies in the region right to B.

We can derive the following inequalities from the minimum spacing rule among A, B, C and D:

$$x_C \leq x_A - s \quad (2)$$
$$y_C \geq y_B + s \quad (3)$$
$$x_D \geq x_B + s \quad (4)$$
$$y_D \leq y_A - s \quad (5)$$

In the above inequalities, s is the minimum spacing between patterns. The minimum Manhattan distance between point C and D is derived as in Inequality (6):

$$\begin{aligned} MD(C,D) &= x_D - x_C + y_C - y_D \\ &\geq x_B - x_A + 2s + y_B - y_A + 2s \\ &> 4s - t \end{aligned} \quad (6)$$

If there exists a conflict between C and D, the Manhattan distance $MD(C,D)$ must be less than the DPL conflict threshold, i.e., $MD(C,D) < t$. Combined with Inequality (6), this will leads to $t > 2s$. $t > 2s$ is a contradiction with the spacing and decomposition rules we have. Thus, C and D cannot be conflicting and it is impossible to construct two conflict edges that intersect each other at the points other than the endpoints of the conflict edges. Thus, Lemma 1 is proved. □

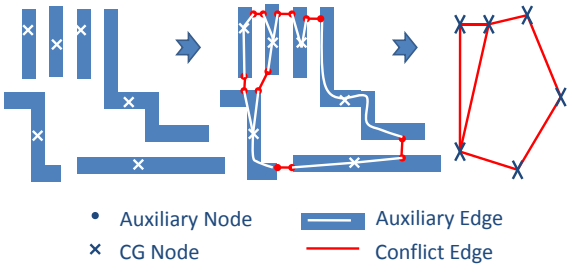


Figure 3: Embedding of Planar CG

With Lemma 1, we can prove the following theorem:

THEOREM 1. *For any Conflict Graph, we can always find a planar embedding such that Conflict Graph is planar.*

Proof: Here is the procedure to construct the embedding. First, we copy the planar layout onto a plane and create a CG node inside each pattern. For every DPL conflict, we find the nearest points that cause DPL conflicts on the two conflicting patterns and create an auxiliary node on each point, as shown in Fig. 3. We create one conflict edge between the auxiliary nodes. We also create auxiliary edges each of which lies within each pattern and connects the CG node and an auxiliary node.

Since conflict edges lie outside patterns and auxiliary edges lie within, they only intersect at the auxiliary points. Obviously, auxiliary nodes from different patterns are separated at least by minimum spacing. By setting the auxiliary nodes for a conflict at the least spaced points on two patterns, their distance should be less than two times the minimum feature size, which is less than two times the minimum spacing. From Lemma 1, conflict edges from entirely different patterns do not intersect. Besides, conflict edges that share the pattern can only intersect at the auxiliary nodes of the shared pattern. So all conflict edges can only intersect at auxiliary nodes.

Next, we contract the existing CG node, auxiliary nodes and auxiliary edges that belong to the same pattern into a new single CG node and form the final CG, in which there exist solely conflict edges. Since the contraction does not change the relative locations of conflict edges, they still only intersects at the endpoints of conflict edges. Thus, we achieve a planar embedding for CG and proved that CG is planar. □

Furthermore, we can see that the maximum DPL spacing threshold that guarantees the planarity of CG is two times the minimum spacing for the layout.

We observe that odd cycles in CG are the sole obstacles for DPL decomposition and we need to break all the odd cycles. So long as we can break every odd face in CG, there would no longer be any odd cycles. Faces are generally smaller than cycles. Besides, the number of faces is linearly proportional to the number of patterns in the layout while the number of cycles are exponential. Both factors will reduce the complexity of breaking odd cycles. Furthermore, the decomposer can easily identify all the obstacles for decomposition by looking at the faces in CG instead of repeating greedy search for odd cycles as in [9].

3. FACE MERGING BASED FORMULATION

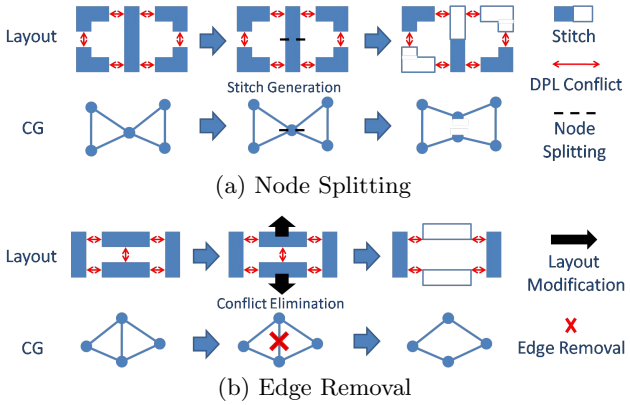


Figure 4: Two Face Merging Operations

The stitch generation and conflict elimination that assist DPL decomposition are represented by node splitting and edge removal in CG, as illustrated in Fig. 4. We observe that both of the operations merge two faces in CG to form a bigger face. We model the feasibility of merging two neighboring faces in Face Graph. FG has one node for each face in CG including the external face. We call the nodes in FG corresponding to even and odd faces in CG as even nodes and odd nodes respectively. We create an edge between two nodes in FG for the following two situations:

- Node splitting: The two corresponding faces in CG share nodes and one or more of the shared nodes can be split. Thus, the faces could be merged by generating stitches.
- Edge Removal: The two corresponding faces share nodes but no shared nodes could be sliced without violating design rules. However, the two neighboring faces also share a DPL conflicting edge, whose removal will merge two faces.

The edge costs for the above two types of edges are set to be c_s and c_r respectively. We keep $c_s \ll c_r$ because conflict elimination is a much more expensive choice than stitch generation.

It is obvious that merging one odd face with an even face will create a bigger odd face, which assists the DPL decomposition in no sense. So it is rational for decomposer to merge neighboring odd faces. If some odd faces have no odd neighbor to merge with, they should be merged with other non-adjacent odd faces, using even faces as agents.

We define pairing as the shortest path in Face Graph for two odd nodes. Each pairing in FG represents the least cost method to merge two odd faces in CG. Because we need to break every odd face in CG to achieve a valid DPL decomposition, we need to pair up every odd node in FG. On the other hand, we can pair up every odd node because there always are an even number of odd nodes in FG. No odd node will be left out in the best pairing solution. So the least cost maximum pairing set would be the best solution for the new DPL decomposition problem.

It may seem that grouping of an even number of odd nodes (corresponding to a Steiner tree) is more general than pairing of two odd nodes. Limiting our consideration to pairing seems to constrain the solution space and produce inferior solution. However, it is not hard to see that any grouping of $2k$ odd nodes can be replaced by a set of k pairings with the same or lower cost to eliminate the $2k$ corresponding odd faces. Hence, it is not beneficial to consider grouping of more than 2 nodes.

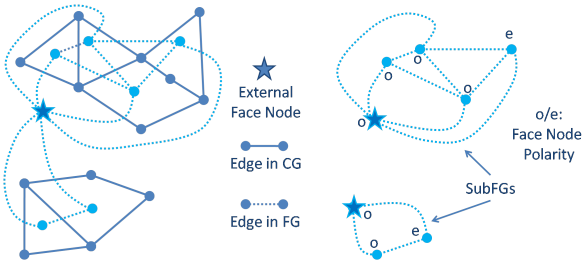
The face merging based formulation brings two benefits. Firstly, our new decomposer only generate a small set of necessary face merging operations as candidate operations for the decomposition problem. We choose one node splitting or edge removal for each pair of neighboring faces. Every operation in our work is essential because ignoring any face-merging operation will lead to sub-optimal solution. On the contrary, [9] generates candidate stitches for every patterns along the odd cycle in hope that some candidate stitches would break the odd cycle. To an extreme, [13] generates candidate stitches at every possible location by slicing patterns into grids. Some of the candidate stitches in [9] and [13] are useful while most are redundant and ineffective in breaking odd cycles. The excessive candidate stitch generation creates many useless candidate stitches that dramatically slow down the runtime, simply because the ILP based selection is not scalable. Secondly, we can transform the face-merging based formulation into a mathematical problem that can be optimized in polynomial time, instead of depending on ILP. The details about the transformation is presented in the following section.

4. DECOMPOSITION ALGORITHM

4.1 Face Graph Partition

If we directly pair up odd nodes in FG, the problem size could be prohibitive. Fortunately, we observe that Conflict Graph contains a vast number of connected components. The nodes in FG corresponding to the faces in one connected component in CG are connected to other nodes only through the single external face node, as shown in Fig. 5(a). We can partition FG into subgraphs called *SubFGs* as illustrated in Fig. 5(b) and perform pairing on them instead.

Each SubFG has the nodes corresponding to the faces in one connected component in CG, one virtual external face node that represents the boundary of the connected component, and the edges between these nodes inherited from FG. We assign the parity of the virtual face node for each



(a) CG and FG (b) SubFG

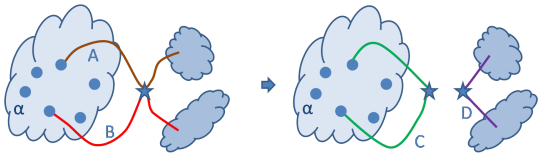
Figure 5: Converting FG into SubFG

SubFG in the following way. If the connected component in CG has even number of odd faces, the virtual node for the corresponding SubFG is even. If the connected components has odd number of odd faces, the corresponding virtual node is odd. In such way, every SubFG always have even number of odd nodes.

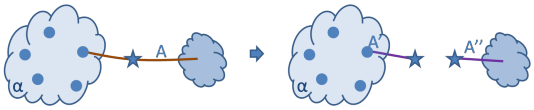
We prove in Lemma 2 that combining the best pairing solution in each SubFG will lead to the best pairing solution in FG.

LEMMA 2. *The combination of best pairing solutions in SubFGs will form a best pairing solution in FG.*

Proof: Assume that in a pairing solution, there is one pairing with two ends coming from two different SubFGs, shown as pairing A in Fig. 6. Let us focus on one of the SubFGs, α. We show all the odd nodes in α in Fig. 6. We have two possible cases depending on whether the α has even or odd number of odd nodes.



(a) Even Number of Odd Nodes



(b) Odd Number of Odd Nodes

Figure 6: Independent Decomposition

For the case shown in Fig. 6(a) where SubFG α has an even number of odd nodes, one odd node will be left without pairing unless it is paired with another odd node in other SubFG. The newly added pairing is B. These two pairings both cross the external node because it is the only node that connects α to the other SubFGs. We can rearrange A and B to form two new pairings. The new pairing C passes the external node for α and fully resides in α.

For the SubFG α with odd number of odd nodes, as shown in Fig. 6(b), it seems one node has to get paired up with a node in other SubFG. On a second thought, the setup of polarity guarantees that the total number of odd nodes stays even in each SubFG. We can separate the original pairings A into two A' and A''.

For both cases presented above, we can always rearrange any pairing sets in FG into pairings that reside fully in SubFGs, without changing of the costs. By solving the least cost pairing problem in each SubFGs and combining the solutions, we can generate a least cost pairing solution for the entire FG. Thus we prove Lemma 2. □

After we split the FG into SubFGs corresponding to connected components in CG, the pairing process would be carried out on each of these SubFGs independently.

4.2 SubFG Simplification

Since odd faces are the only type of faces that need to be eliminated and even faces are only used to assist the elimination of odd faces, we simplify each SubFG to generate a graph only containing the odd nodes in the following way. We first use Floyd-Washall algorithm [15] to find the shortest path for each pair of odd nodes in each SubFG. We create a complete graph for the odd nodes in SubFG, in which the edge weight is set to the cost of the shortest path between the corresponding pair of odd nodes in the SubFG. We call the newly created graph *Pairing Cost Graph* (PCG). The PCG stores the costs of all possible pairings in a SubFG.

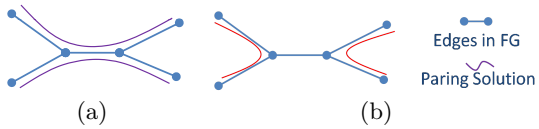


Figure 7: Shortest Pairings

We find that an edge in SubFG may be embedded into multiple edges in PCG. It seems that the decomposer might choose several edges that contain identical SubFG edges and thus overcount the costs in final solution. Such situation does not exist. For two pairings that shared an edge, as shown in Fig. 7(a), the nodes involved could be paired up differently as shown in Fig. 7(b). The second pairing solution costs less than the first one. Thus, the first pairing case shown in Fig. 7(a) would never be chosen in the least cost solution.

4.3 Matching Based Solution

After the simplification in previous subsection, we want to pair up every node in the PCG with a minimum weight. Because every SubFG has even number of odd nodes, the simplification process in last section will generate PCGs with even number of nodes. Besides, every PCG generated is a complete graph. It is obvious that a complete graph with even nodes always has a perfect matching in which all the vertices are matched. So to solve the odd face merging problem for DPL layout decomposition, we just need to find a minimum weighted perfect matching for every PCG.

Since the only matching solvers we can find is for maximum matching problems, we convert the minimum weighted perfect matching on a graph to maximum weighted matching in the following way. For a PCG G with the maximum edge weight MAX_w. We denote H as $MAX_w + 1$. We construct another graph G' with the same topology as G. The weight for edge e'_i in G' is set to $H - W_i$ where W_i is the weight for the corresponding edge in G. In such a way, we transform the minimum weighted perfect matching problem on G into a maximum weighted matching on G'. We create a G' for each PCG by modifying the edge weights and use an $O(n^3)$ maximum weighted matching package [16] to achieve solution for the entire DPL layout decomposition problem.

It is worth mentioning that a paper [17] published in 1975 applying a similar matching based method to solve maximum cut of a planar graph in polynomial time. They used minimum odd circuit cover problem in planar graph as a start point, carried out a series of transformation and ended up in a maximum matching problem. Although the work looks similar to our work, in the DPL decomposition problem, we can use node splitting to merge odd faces, so we have an extra operation, i.e. node splitting, to merge two odd faces. Besides, our face graph is more complex than normal dual graph.

4.4 Dependent Stitches

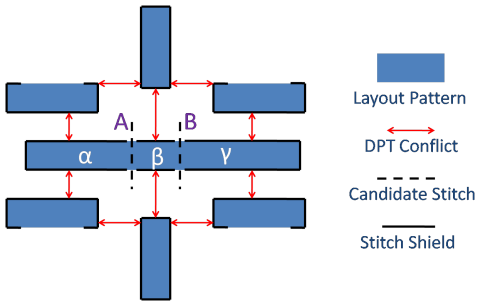


Figure 8: Dependent Stitches

Because our decomposer needs to analyze all the conflicts and possible face merging operations to simultaneously make decisions, it mandates that all the face merging operations should be independent. However, we devised a structure illustrated in Fig. 8 that violates such requirement. In Fig. 8, stitch shields are introduced to prevent stitches on certain locations where stitches will cause design rule violation or pattern boundaries that are in DPL conflicts with other patterns. With the stitch shields, the only places to use stitches to break the odd faces in Fig. 8 are A and B. However, stitches A and B cannot coexist or there would be a DPL conflict no matter how subpatterns α , β and γ are assigned. We see that the stitches require α, β and β, γ assigned onto different masks, so no matter which mask we assign β to, α and γ would be assigned onto the other mask. However, the distance between α and γ is less than the DPL conflict threshold so their coexistence on the same mask violates DPL decomposition rule. Because our decomposer assumes face merging and thus the edges in FG are independent, the matching formulation cannot handle dependent stitches.

Dependent stitches occur only in this carefully constructed example. Even though we did not observe any dependent cases in the experiments, we design a post-processing step in our decomposer to tackle such situation. If dependent stitches are found in matching based solution, we enumerate the two possible cases of disallowing one of the two stitches by removing the corresponding edge in FG. We rerun both cases and the final solution is the better of the two.

5. EXPERIMENTS

We implement our new decomposer in C and test it on a $2.6GHz$ Intel Linux machine with $6GB$ memory. We use the same benchmark as [10] and compare the results of the new decomposer with it. The benchmark AES is a real world design from opencore.org while the rest three are artificial designs instantiated from more than 600 different cell masters. The original minimum spacing and minimum feature size is $140nm$ and $100nm$ respectively. They are scaled down by a factor of 0.4, i.e., $56nm$ minimum spacing and $40nm$ minimum feature size, to simulate future designs. We compare the results of our decomposer to [10] in Table 1.

Table 1: Experimental Results

Design	#Patterns	Our Decomposer			[10]			[14]		
		# ER	# Stitches	cpu(s)	# ER	# Stitches	cpu(s)	# ER	# Stitches	cpu(s)
AES	90394	0	30	4.8	0	35	17.2	0	30	5.9
TOP-B	545000	637	10892	42.6	800	14027	448.1	865	10265	15.4
TOP-C	2725000	3502	67581	205.5	4000	69490	6629.0	4273	64385	310.7
TOP-D	1090000	1328	25853	97.3	1600	27908	1228	1724	24767	85

In Table 1, "# ER" is the number of edges to be removed while "# Stitches" is the number of stitches. Comparing to [10], our decomposer generates solutions with 15% less conflicts to eliminate and 7% less of stitches. Moreover, we achieve 25 times speed-up. We notice that [10] also optimizes the overlap length of stitches, which sacrifices the number of stitches so the results might not be directly comparable. However, the scale of reduction of both stitches generation and conflict elimination is significant enough to demonstrate the effectiveness of our new decomposer. In comparison to [14], we dramatically reduce the number of conflict edge removal by 20% at the cost of merely 5% overhead of stitches. For runtime, the matching based decomposer is marginally slower than [14] on small benchmarks. However, it outperforms the fastest previous work for large designs, which is the common work for current VLSI designs.

We cannot get the results of [13] on the same benchmarks for comparison due to IP issues. However, according to the results in [13], the decomposer needs 70 seconds to decompose just 18975 patterns (their largest benchmark). Considering the fact that the ILP based decomposer is not scalable, the speed is just too slow for modern design with millions of patterns.

We are aware that speed is a secondary criteria to judge DPL decomposer. Design quality factors such as the number of stitches generated and conflicts to eliminate are more important. However, due to the exploding pattern number and the unavoidable rounds of redesign, effective control over runtime is a must for DPL decomposer. The speed is critical especially when, in the early design stage, designers know the incompatibility of layout but want to know where to carry out layout modification. Our decomposer produces optimal solutions in a much shorter runtime than previous approaches.

In our experiments, the largest number of odd faces for a connected component in CG is 10, leading to a PCG of merely 45 edges. Besides, a connected component has 1.24 odd faces on average. By adopting the face-pairing based formulation, we greatly reduce the complexity for DPL decomposition problem.

6. CONCLUDING REMARKS

In this paper, we prove the planarity for DPL Conflict Graph and develop a face merging based optimal solution for the DPL decomposition problem. Our experimental results show that the new framework can achieve significantly less number of stitches generated and conflicts to eliminate. Our future research will focus on DPL guided detail routing.

We want to point out an assumption made in edge removal, the second type of face merging operation. We give our decomposer the power to remove a conflict edge without affecting the surrounding conflicts or void of conflicts. In reality, patterns in a compact layout are spatially correlated so that modifying patterns may not only remove the DPL conflict in focus, but also affect DPL geometric relationships between multiple patterns. It may occasionally remove or generate a few DPL conflicts. Some previous decomposers consider a few detailed objectives about layout modification, like the minimization of displacement of patterns involved in the conflict elimination. Even so, they still leave out critical issues like the connection between layers, timing and noise. These issues are essential for the correctness of layout modification. However, they are too complicated to be included in a DPL decomposer, which makes the partial consideration of layout modification in DPL decomposers useless. So we hold the believe that so long as the DPL decomposer pinpoints and minimizes the conflicts to eliminate for layout modification, the details about the modification should be left for experts or more focused layout modification tools.

7. REFERENCES

[1] Chang-Moon Lim, et al., "Positive and negative tone double patterning lithography for 50nm flash memory," Proc. SPIE 6154, 2006

[2] Jungchul Park, et al., "Application challenges with double patterning technology (DPT) beyond 45 nm," Proc. SPIE 6349, 2006

[3] Mircea Dusa, et al., "Pitch doubling through dual-patterning lithography challenges in integration and litho budgets," Proc. SPIE 6520, 2007

[4] Yuichi Inazuki et al., "Decomposition difficulty analysis for double patterning and the impact on photomask manufacturability," Proc. SPIE 6925, 2008

[5] George E. Bailey, et al., "Double pattern EDA solutions for 32nm HP and beyond," Proc. SPIE 6521, 2007

[6] Martin Drapeau, et al., "Double patterning design split implementation and validation for the 32nm node," Proc. SPIE 6521, 2007

[7] Tsann-Bim Chiou, et al., "Development of layout split algorithms and printability evaluation for double patterning technology," Proc. SPIE 6924, 2008

[8] Anton van Oosten, et al., "Pattern split rules! A feasibility study of rule based pitch decomposition for double patterning," Proc. SPIE 6730, 2007

[9] A.B. Kahng, et al., "Layout decomposition for double patterning lithography," ICCAD 2008, Nov. 2008

[10] A.B. Kahng, et al., "Revisiting the layout decomposition problem for double patterning lithography," Proc. SPIE 7122, 2008

[11] C. Chiang , et al., "Fast and Efficient Bright-Field AAPSM Conflict Detection and Correction," TCAD, Volume 26, Issue 1, Jan. 2007

[12] A.B. Kahng, et al., "New graph bipartizations for double-exposure, bright field alternating phase-shift mask layout," Proceedings of the ASP-DAC, 2001

[13] Kun Yuan, et al., "Double Patterning Layout Decomposition for Simultaneous Conflict and Stitch Minimization," International Symposium on Physical Design (ISPD), San Diego, March 2009

[14] Y. Xu, et al., "GREMA: Graph Reduction Based Mask Assignment for Double Patterning Technology," ICCAD 2009, Nov. 2009

[15] Robert W. Floyd, "Algorithm 97: Shortest path," Commun. ACM 5, 1962

[16] http://elib.zib.de/pub/Packages/mathprog/matching/weighted/

[17] F. Hadlock, "Finding a Maximum Cut of a Planar Graph in Polynomial Time," SIAM J. Comput. 4, 221, 1975