An Effective Triple Patterning Aware Grid-based Detailed Routing Approach

Zhiqing Liu, Chuangwen Liu and Evangeline F.Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{zqliu, cwliu, fyyoung}@cse.cuhk.edu.hk

Abstract— **Triple patterning lithography (TPL) is attracting more and more attention due to further scaling of the critical feature size. How fully the benefits of TPL can be utilized depends very much on both the decomposition and layout steps. However, it is non-trivial to perform detailed routing and layout decomposition simultaneously on a large-scale complicated circuit to achieve decomposability on one hand, and short wirelength, small number of stitches and small number of vias on the other hand. In our approach, routing and coloring are done iteratively but integrated closely to reduce the problem complexity. The routing step is able to detect and avoid native conflicts as much as possible. If any conflicts occur in the coloring step, the router will rip-up and re-route to get rid of them. This technique proves to be effective and efficient in improving the quality of the coloring assignment. Compared with previous works [1] on TPL using simultaneous routing and coloring, the number of stitches and the number of vias are reduced by 76.8% and 2.1% respectively while our running time is 36.6% less and the wirelength is very comparable.**

I. Introduction

Double Patterning Lithography (DPL) [6] has been successfully used in the sub-22nm technology node [7], [8], [9], [10]. To further shrink the process node, TPL is considered a natural extension of the DPL paradigm to resolve the manufacturing problems, especially when the next generation lithography technology such as Extreme Ultra-Violet (EUV) and Electron Beam (E-Beam) are not yet available for mass production. However, using TPL requires a non-trivial step, called layout decomposition. In layout decomposition, features which are too close to each other must be assigned to different masks (different colors). Stitches can be used to resolve coloring conflicts, i.e., two close-by features are assigned to the same mask, by *splitting* a feature into sub-features so that the two sub-features can be layout on different masks. However, stitches are not preferred because of their high sensitivity to overlay error. In some cases, insertion of stitches still cannot resolve the problem because of the existence of native conflict. For example, in TPL, the existence of a K4 (a complete subgraph with 4 nodes) in the conflict graph of a layout is a native conflict and the layout will be undecomposable.

However, if the layout process can take into consideration the decomposition process, many hard problems during decomposition can be avoided. An example is shown in Fig. 1. In this example, if the wires are routed without considering TPL decomposition, a layout as shown in Fig. 1(a) may be resulted but this layout is undecomposable because of the existence of a K4. However, if TPL decomposition is considered, a layout of the same wirelength as shown in Fig. 1(b) may be resulted which can be decomposed without using any stitches. It is thus essential to consider the constraints in decomposition during the layout process and in detailed routing in particular. In the past, there are a number of works on DPL aware detailed routing [3], [4], [5] and a few are on TPL-driven detailed routing [1], [2]. The work in [2] is on gridless routing. In their router called TRIAD, a conflict graph called *token graph-embedded conflict graph* (TECG) is proposed to help detecting TPL conflicts. TECG is used with a gridless router to perform TPL aware detailed routing. Another paper [1] solves a similar problem in grid-based routing. The whole routing region is modeled by a complex routing grid graph with some internal structures to allow color changing in the routes of a net. However, since the graph is huge and maze routing is used, the runtime is long.

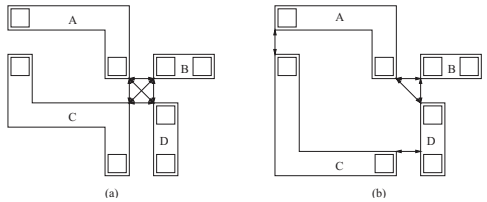
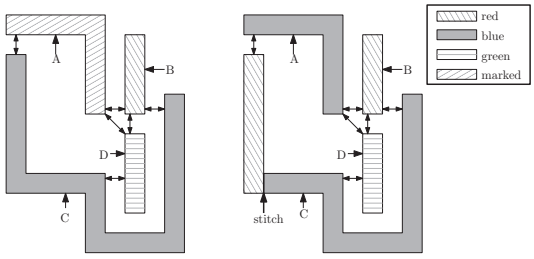


Fig. 1: Motivation example for TPL aware detailed routing.

In this paper, we propose an efficient method to solve this decomposition aware detailed routing problem. In our router, routing and coloring are done iteratively but integrated closely. However, some native conflicts will be avoided during routing in order not to create hard and infeasible cases for the coloring process. The two stages (routing and coloring) will be iterated to fix both the coloring and routing problems simultaneously. We have applied this approach to solve the TPL aware routing problem and can achieve 1.58× speedup comparing with the latest result [1] while the quality is also improved by 76.8% and 2.1% in terms of stitch numbers and via

This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project CUHK14209214.

978-3-9815370-4-8/DATE15/©2015 EDAA



(a) A K4 exists between A, B, C and D (b) Decomposable with a stitch

Fig. 2: An example of resolving conflicts using stitches in TPL

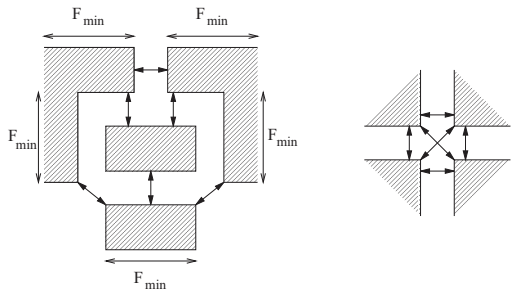


Fig. 3: An example of native conflict in TPL

numbers respectively.

The rest of the paper is organized as follows. Section II introduces some preliminaries on TPL layout decomposition and Section III formulates the problem. Section IV gives an overview of our methodology. Section V and VI describes in details the routing and coloring steps. Finally, experimental results will be shown in Section VII and Section VIII concludes the paper.

II. Preliminaries

A. Layout Decomposition

Layout decomposition assigns the features in a layout to a set of masks, such that two features close to each other (at a distance of smaller than a threshold C_{min}) must be assigned to different masks. Theoretically, a conflict graph can be constructed in which the vertices represent features and an edge exists between two features if they are closer than C_{min} from each other. The problem of layout decomposition can be transformed to a 3-coloring problem on the conflict graph. However, a layout with a conflict graph that cannot be 3-colored may also be decomposable because we can use stitches to split a feature into two. However, stitch usage should be minimized since they are very sensitive to overlay error.

B. Native Conflict

A native conflict occurs in decomposition when no coloring assignment can be made without violating the constraint that features less than C_{min} apart must be colored different. This is a very bad scenario since decomposition is impossible even with stitches. In this case, re-placement or re-routing is needed. Therefore, it is good if native conflict can be detected and avoided during the placement or routing process. However, the existence of K4 in the conflict graph in TPL does not necessarily lead to native conflicts since stitches can be inserted. An example is shown in Fig. 2. In this example, a K4 exists in the conflict graph, but a stitch inserted into feature C can resolve the problem and have the layout properly decomposed into 3 colors.

A native conflict occurs if the K4 is between *unseparable* points of a feature. Here two points on the same feature are said to be unseparable if they are less than F_{min} apart to each other where F_{min} is the minimum feature size, i.e., the two points cannot be separated by a stitch (so the two points must be colored the same). An example is shown in Fig. 3. The K4 in the conflict graph leads to a native conflict for this example. We call such kind of K4 an *unseparable K4*. By eliminating unseparable K4, we can avoid many native conflicts and make the layout easier-to-decompose.

III. Problem Formulation

TPL-driven DR In this problem, we are given a minimum spacing requirement C_{min}, a minimum feature size F_{min}, 3 masks (or colors) to be used, a routing region R of size $p \times q$ and a set N of m two-pin nets with corresponding pins on R. The objective is to assign routes and colors to these nets in such a way that adjacent wires at a distance of less than C_{min} apart should be colored differently. The secondary objective is to minimize the number of stitches and the number of vias.

In order to compare with the most recent work [1] on this problem, we assume two metal layers and wires can go both horizontally and vertically on both layers[1] and stitch can be inserted at any grid along the path except the two pins ($F_{min} = unit\_grid\_length$).

IV. An Overview of our Method

Instead of doing routing and coloring simultaneously as in [1], our approach will iterate between routing and coloring. Most of the nets can be routed and colored very quickly in the first round, leaving behind a small number of harder cases. Then, more sophisticated techniques will be used to route and color the remaining nets. This approach brings a lot of speedup while maintaining a high quality result in terms of decomposability. High decomposability is preserved because firstly, the routing step will avoid generating native conflicts, and secondly it will rip-up and re-route uncolorable nets. Besides, the number of stitches will be minimized because the coloring step will use only one color for each net on a layer as much as possible. The number of vias are minimized by imposing a large via-cost during the routing step. An overview of our approach is shown in Fig. 4 and details will be given in the following sections.

[1]The maze routing we use can be extended to handle preferred routing direction

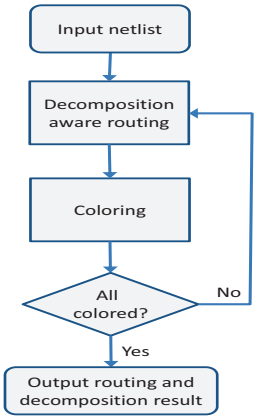


Fig. 4: An overview of our approach

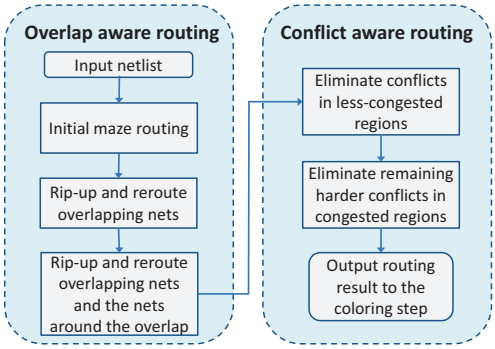


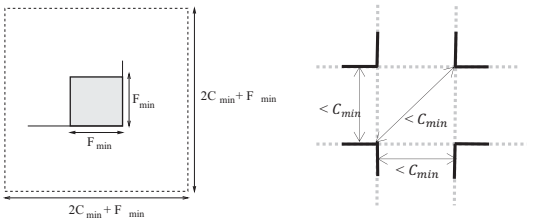
Fig. 5: Decomposition aware routing

V. Decomposition Aware Routing

The decomposition aware routing is divided into a few steps as shown in Fig. 5. Firstly, *overlap aware routing* will be performed to produce a routing solution without wire overlap. This is done by negotiated maze routing. Then we will detect native conflicts over the whole routing region and eliminate them by ripping-up and re-routing some nets. We call this phase *conflict aware routing*. Details of each phase will be given in the following sections.

A. Overlap Aware Routing

The overlap aware routing can be described by the pseudo-code in Algorithm 1. Each routing edge on the same layer has a wirelength cost of one, while the upward and downward edge (vias connecting the upper and lower layer) have a cost of $cost_{via}$ ($cost_{via} = 18$ in our implementation). Each node has a *history cost* which indicates its congestion history and is initialized as zero. Maze routing is performed for all the nets following an order Γ in which a net is routed earlier if the total number of net pins inside its bounding box is fewer. We route a net with the least-cost path, where the cost is the sum of the edge costs and the node history costs. If there is overlap along the path, the history cost of each overlapping node will be incremented by a penalty α ($\alpha = 2$ in our implementation) while the history cost of its 4 neighboring nodes and 4 diagonally adjacent nodes will be incremented by $\alpha/8$ and $\alpha/16$ respectively. Then we get an initial routing solution. If it is not overlap-free, several rounds of rip-up and re-route will be performed. In the first round, we will only rip-up and re-route the overlapping net to check if it can be re-routed easily without overlap. For unsuccessful cases, we will rip-up both the net and all the nets overlapping with it. Then we will re-route the problematic net first before re-routing the others. The number of times a net was ripped-up is recorded during the whole process. After this step, if overlap still exists, we will sort all unsuccessful nets in a decreasing order of their number of times of being ripped-up. Generally speaking, the more times a net was ripped-up, the harder it can be routed. We will further rip-up the nets routed at a distance of less than $ripDist$ from the overlapping nodes. The value of this parameter $ripDist$ will increase from 0 to $maxDist = 10$ ($maxDist = 10$ in our implementation) until all the nets can be routed without overlap. In this way and with the help of the accumulative history cost, all the nets can be routed without overlap in most cases, except if the net density is too high.



(a) General layout (b) Grid-based routing

Fig. 6: Detection of native conflict

Algorithm 1 Overlap Aware Routing

```
Input: A uniform two-layer routing grid G, a netlist N of two-pin nets on the grid
Output: The routing paths of all nets on the grid
Sort all nets in an increasing order of the number of pins inside their bounding boxes
Maze route all nets to get an initial routing solution
while ∃ overlap and curIter++ ≤ maxIter do
    for each net n_i with overlap do
        Rip-up and re-route n_i
        Update history cost and rip-up times
        if n_i still has overlap then
            Rip-up and re-route net n_i and its overlapping nets
            Update history cost and rip-up times
        end if
    end for
end while
while nets with overlap N_remain ≠ ∅ do
    Identify n_critical ∈ N_remain with the largest rip-up times
    Initialize ripDist = 0
    while n_critical has overlap and ripDist++ ≤ maxDist do
        Rip-up and re-route n_critical and surrounding nets N_around within ripDist from the overlapping grids
        Update history cost and rip-up times
    end while
    Update N_remain
end while
```

B. Detection of Native Conflicts

Since two stitches cannot be inserted at a distance of less than F_{min} apart, the minimum feature size, we can

identify some native conflicts by detecting unseparable K4. For general layouts, we can look at the corner of each feature. Let x be a sub-feature of size $F_{min} \times F_{min}$ at a corner of a feature. Then, we define a window of size $2C_{min} \times 2C_{min}$ surrounding x. We can then check for the existence of K4 in the window. An example is shown in Fig. 6a. For grid-based routing, unseparable K4 can be detected by checking whether routes exist in some neighboring grid points in certain fashion depending on the size and dimension of the routing grid graph. An example is show in Fig. 6b.

C. Conflict Aware Routing

The conflict aware routing is to eliminate the detected native conflicts, as described by Algorithm 2. Note that eliminating one of the 4 nodes in a K4 can destroy the K4. For each conflict, we will try ripping-up only one of the 4 nets involved and *blocking* the corresponding node in the clique to see if we can find a new conflict-free and overlap-free path. If there is such a new path for one of the 4 nets, we will re-route the net with the new path. However, if there is no such path after trying all 4 nets, the native conflict is likely located in a congested region. We will leave the conflict for later processing.

For those harder conflicts, we will try ripping-up some neighboring nets to create room for them. To do this, the problematic net i (involved in a K4) will first be ripped-up and re-routed with the corresponding node in the clique blocked. There must be overlap on this new path, because otherwise this net has been resolved in the previous stage. For the nets overlapping with this new path and even those nets surrounding the overlapping nodes, we will rip-up and re-route them. An example of this case is shown in Fig. 7. The initial routing result in Fig. 7(a) contains a K4 between net A, B, C, D in a congested region. When we rip-up and re-route net D, its new path will have conflict with net E, as shown in Fig. 7(b). Then we will further rip-up and re-reroute net E to make room for net D and finally resolve the overlap and K4.

Algorithm 2 Conflict Aware Routing

Input: Overlap-free routing result and a set of detected native conflicts $C_{conflict}$
Output: Overlap-free routing result with no native conflicts detected
while $Iter++ \leq maxIter$ **do**
  for each conflict $c_i \in C_{conflict}$ **do**
    for each net n_i in the K4 **do**
      Rip-up n_i, block the corresponding node and re-route
      if c_i is solved **then**
        Update $C_{conflict}$; **break**
      else
        Restore the path
      end if
    end for
  end for
end while
while unsolved native conflicts $C_{remain} \neq \varnothing$ && # iterations with no improvement $noImprIter \leq threshold$ **do**
  for each conflict c_i in C_{remain} **do**
    Rip-up one of the 4 nets, block the corresponding node and re-route
    if the new path overlaps with other nets **then**
      Rip-up and re-route the overlapping nets and the surrounding nets to make room for solving c_i
    end if
  end for
  Use the overlap aware routing method to eliminate overlap
  Update C_{remain}, $noImprIter$
end while

VI. Decomposition - Coloring

In our coloring scheme, first, a greedy coloring method is used to color as many nets as possible and balance the 3 mask usage. After this step, many nets are colored and the uncolorable nets are marked. Second, we apply a random coloring method to reduce the number of marked nets by re-coloring. Finally, stitches will be inserted to get rid of the remaining uncolorable nets. If some nets still cannot be resolved by stitches, we will rip-up and re-route them in the next round.

A. Step One: Less Flexible First Coloring

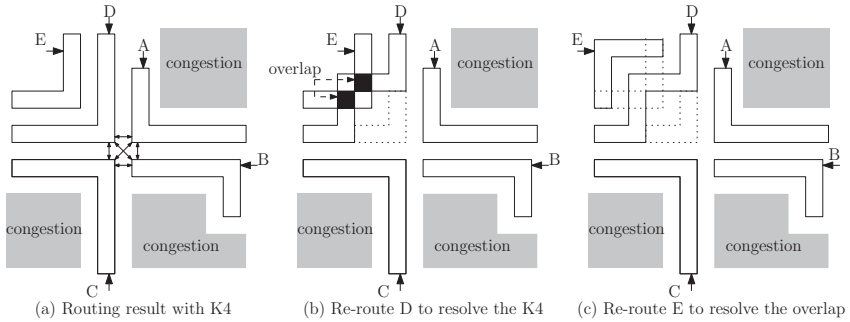
In this step, we will try to color the nets one by one based on the *Less Flexible First* principle. The net with the fewest color choices will be colored first. If multiple nets have the same color choices and are least, we would color the one with longer wirelength first because it is likely to have more coloring constraints. We use a simple method to balance the mask usage here, which is by choosing the color with the smallest total usage from among all feasible choices. After a net is colored, the color choices of its surrounding nets will be updated.

B. Step Two: Random Coloring Scheme

We develop a random coloring scheme that can effectively color most uncolorable nets after step one above. First, for each marked net, the colored nets surrounding it will have their colors removed and the color choices of any affected nets will be updated. We use a parameter $uncolorDepth$ to control the number of nets being uncolored: nets within a distance of $uncolorDepth$ from the marked net in the conflict graph will be uncolored. For example, if $uncolorDepth = 1$, only the nets at a distance one from the marked net in the conflict graph will be uncolored. Then we will color the marked net first and then color the other nets with colors chosen randomly from the feasible color choices. Note that here, instead of choosing a color with the least usage, we will choose a color randomly from the feasible choices. This is to create some randomness in order to resolve conflicts of the remaining nets. This re-coloring process will be performed for a number of iterations and we will choose the best coloring solution to start with in the next step.

C. Step Three: Adding Stitches and Re-routing

Stitches will be inserted if there are uncolorable nets left. For each uncolorable net, we will compute the color choices at each point along the path. If there is at least one color choice for each point on the path, we can add stitches and have it colored. If this is not the case, simply adding stitches will not work. However, not all such cases are native conflicts. To solve those cases which are not native conflicts, we developed a simple but effective heuristic as shown in Algorithm 3. For each marked net n_i, we will first find the set of nets N_c having conflict with it. If there exists a net $n_x \in N_c$ having another feasible color, we simply color n_i with n_x's current color



(a) Routing result with K4 (b) Re-route D to resolve the K4 (c) Re-route E to resolve the overlap

Fig. 7: An example of our conflict aware routing approach

Algorithm 3 Adding Stitches

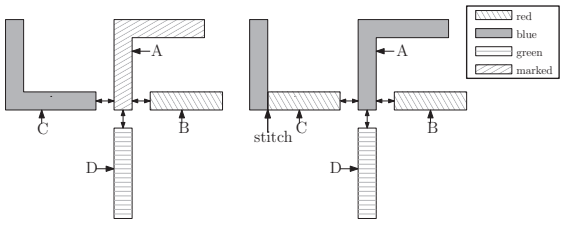
```
1: Input: Original coloring result with a set of marked nets, M
2: Output: Coloring result with stitches to minimize M
3: while M ≠ ∅ && Iter ≤ maxIter do
4:     for each net n_i ∈ M do
5:         Identify nets N_c having conflicts with n_i
6:         if net n_x ∈ N_c has another coloring choice then
7:             Color n_i with n_x's current color and change n_x's color
8:             M = M − {n_i}
9:         else
10:            if every point on n_i has some color choices then
11:                Insert stitches in n_i
12:            else
13:                Find net n_y ∈ N_c to swap the color with n_i.
14:                M = M − {n_i} + {n_y}
15:            end if
16:        end if
17:    end for
18: end while
```

and change n_x's color to that feasible color. If this is not the case, we will find a colored net $n_y \in N_c$ which has less coloring constraints to swap its color with n_i. Now n_y becomes marked but the conflict will be easier to be solved in the next iteration. A simple example shown in Fig. 8 may help illustrating our method. A point on A has conflicts with B, C and D at the same time while B, C and D are all colored differently. Assume that C cannot be colored red because of some coloring constraints with some nets that are not drawn in this figure.(Otherwise, we will assign C with red and A with blue and the conflict is solved easily in $Step\ Two$.) Now assume that inserting stitches in A does not work, we will then check whether any one of B, C and D can change its color. If one does, say X where X = B, C or D, we can simply change X's color and have A taking X's original color. Otherwise, we will swap the role of A and one of B, C or D (call it Y). If any of them has some color choices at each point along its path, the conflicting scenario can be solved, as shown in Fig. 8b.

If stitches cannot resolve all the coloring conflicts, we will rip-up and re-route those problematic nets. It is observed from the experiments that the number of problematic nets decreases quickly after several iterations of routing and coloring.

VII. Experimental Results

We implement a TPL aware detailed router based on the approach described in this paper in C. All the experiments were performed on a 64-bit Linux machine with Intel Xeon 3.4GHz CPU and 32GB memory.



(a)Inserting stitch in A doesn't help (b)Inserting stitch in C helps

Fig. 8: An illustration of inserting stitch

A. Comparison with Previous Works on TPL aware Detailed Routing

In this part, we compared our TPL aware detailed router with the one developed in the most recent work [1] on this problem. The original benchmarks of [1] are not available to us but the author provided us the program to generate the data sets. We generated 6 benchmarks according to the settings in [1], and run both our TPL aware router and the TPL aware router in [1][2]. The results are shown in Table I. Fig. 9 shows a part of the routing solution of test case $test4$. Compared with [1], both approaches can route and color all 6 benchmarks successfully but our approach uses 76.8% fewer stitches and 2.1% fewer vias, and runs about 1.58× faster on average. Although the approach in [1] can handle routing and coloring simultaneously, it splits a node in the original routing graph into 24 internal nodes with 60 edges inside. One single routing edge between two nodes becomes 18. Thus the problem size increases significantly, making it extremely difficult and time-consuming to obtain a good result. However, our two-stage approach simplifies the problem and the re-routing technique based on the coloring result can resolve the hard-to-color cases efficiently.

B. Comparison between With and Without Conflict Aware Routing

In order to have the same setting as [1], the value of C_{min} for the experiments in Table I is set in such a way that a node on a path will only conflict with its perpendicularly adjacent nodes. In order to have a

[2]Provided by the author of [1]

TABLE I: Comparison on TPL-aware Detailed Routing

Test cases	#Net	Grid size	#Stitch		#Via		#Conflict		Wirelength			Runtime(s)	
			[1]	Ours	[1]	Ours	[1]	Ours	HPWL	[1]	Ours	[1]	Ours
test1	$1k$	90×90	29	4	386	368	0	0	4346	4720	4802	5	4
test2	$2k$	130×130	32	0	658	734	0	0	8692	9528	9770	12	10
test3	$4k$	200×200	73	40	1044	1042	0	0	19869	21455	22029	103	29
test4	$6k$	240×240	120	44	1394	1286	0	0	26540	29254	30140	123	69
test5	$8k$	274×274	197	30	1868	1796	0	0	35158	39178	40098	282	185
test6	$10k$	300×300	208	35	2690	2642	0	0	43625	48201	49335	391	284
Avg.			109.8	25.5	1340	1311.3	0	0	23038.3	25389.3	26029	152.7	96.8
Diff.			1	-76.8%	1	-2.1%	—	—	—	1	+2.5%	1	-36.6%

TABLE II: Comparison between With and Without Conflict Aware Routing

Test cases	#Net	Grid size	Initial #K4	#Stitch		#Via		#Conflict		Wirelength			Runtime(s)	
				W/O	With	W/O	With	W/O	With	HPWL	W/O	With	W/O	With
test1c	$1k$	110×110	4	33	29	372	368	9	7	7247	8097	8087	21	29
test2c	$2k$	150×150	10	70	55	908	930	24	18	14552	16458	16476	93	116
test3c	$4k$	220×220	17	139	122	1536	1554	43	33	29299	33041	32963	155	199
test4c	$6k$	270×270	23	221	176	2240	2258	66	58	43933	49951	49819	398	541
test5c	$8k$	310×310	26	244	214	3106	3184	85	72	58716	66150	66100	584	734
test6c	$10k$	350×350	42	308	316	3582	3576	95	90	73031	81823	81679	828	723
Avg.			20.3	169.2	152	1957.3	1978.3	53.7	46.3	37796.3	42586.7	42520.7	346.5	390.3
Diff.			—	1	-10.2%	1	+1.1%	1	-13.8%	—	1	-0.2%	1	+12.6%

closer study on the effectiveness of our conflict aware routing approach, we modify the value of C_{min} so that a node will conflict with both perpendicularly adjacent and diagonally adjacent nodes. We generate another 6 benchmarks (since we now need more space to place the pins) and perform experiments to compare the performance of using and not using conflict aware routing. Results are shown in Table II. The column "Initial #K4" shows the number of K4 after the overlap aware routing step. The approach with conflict aware routing will eliminate all the K4 before coloring while the one without will do coloring directly. We can see from the results that the approach with conflict aware routing finally reduces the conflict number by 13.8% and stitch number by 1.0% on average. The running time of the approach with conflict aware routing is expected to be longer than that of the one without conflict aware routing because of the extra step of eliminating the K4.[3]

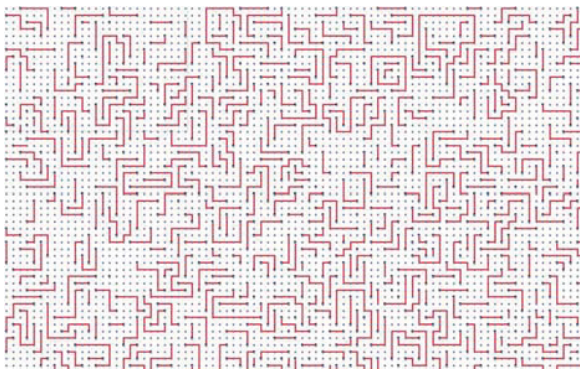


Fig. 9: A snapshot of the benchmark $test4$

[3]The running time of $test6$ with conflict aware routing is shorter because there is randomness in the approach and the one without conflict aware routing for some reason just meets the stopping criteria earlier.

VIII. Conclusion

In this paper, we propose a TPL aware grid-based detailed routing approach. The routing step and coloring step are done iteratively but integrated closely. The routing step will consider the coloring issue and the coloring step gives feedback on coloring conflicts to guide the routing step. Experimental results show that the TPL aware detailed router using our approach yields 36.6% less runtime, 76.8% fewer stitches and 2.1% fewer vias than [1], the latest work on this problem.

Acknowledgement

The authors would like to thank Prof. Martin D. F. Wong and Dr. Qiang Ma (the authors in [1]) for their kind help.

References

[1] Q. Ma, H. Zhang and M. D. F. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology," *Proc. DAC*, pp. 591–596, 2012.

[2] Y.-H. Lin, B. Yu, D. Z. Pan and Y.-L. Li, "TRIAD: A triple patterning lithography aware detailed router," *Proc. ICCAD*, pp. 123–129, 2012.

[3] M. Cho, Y. Ban and D. Z. Pan, "Double patterning technology friendly detailed routing," *Proc. ICCAD*, pp. 506–511, 2008.

[4] Y.-H. Lin, et al, "Double patterning lithography aware gridless detailed routing with innovative conflict graph," *Proc. DAC*, pp. 398–403, 2010.

[5] J. Sun, Y. Lu, H. Zhou and X. Zeng, "Post-routing layer assignment for double patterning," *Proc. ASPDAC*, pp. 793–798, 2011.

[6] A. B. Kahng, et al, "Layout decomposition for double patterning lithography," *Proc. ICCAD*, pp. 465–472, 2008.

[7] G. E. Bailey, et al, "Double pattern EDA solutions for 32nm HP and beyond," *Proc. SPIE*, pp. 65211K-1–65211K-12, 2007.

[8] J. Huckabay, et al, "Process results using automatic pitch decomposition and double patterning technology (DPT) at k1eff < 0.20," *Proc. SPIE*, pp. 634910-1–634910-11, 2006.

[9] Y. Inazuki, et al, "Decomposition difficulty analysis for double patterning and the impact on photomask manufacturability," *Proc. SPIE*, pp. 69251O-1–69251O-10, 2008.

[10] V. Wiaux, et al, "Split and design guidelines for double patterning," *Proc. SPIE*, pp. 692409-1–692409-11, 2008.