TEG: A New Post-Layout Optimization Method [*]

Shuo Zhang
Department of Computer Engineering
University of California
Santa Cruz, California 95064
shzhang@soe.ucsc.edu

Wayne W.-M. Dai
Department of Computer Engineering
University of California
Santa Cruz, California 95064
dai@soe.ucsc.edu

ABSTRACT

Post-layout is an important stage in the modern VLSI design. With the completed detail routing, it is the only stage where extraction and verification tools can get accurate results for further optimization. But the problem is that design optimization or modification are very hard to perform in the post-layout stage, because most layout elements are under tight geometry constraints due to the routing. In this paper we propose a new method to resolve this problem, named TEG. Based on an improved topological layout representation and a set of layout operation algorithms, TEG provides an incremental layout modification environment for the post-layout applications. Experimental results showed that TEG was efficient and effective in processing industry VLSI designs.

1. INTRODUCTION

The modern VLSI industry always targets at higher chip performance. This continuously brings out new design methodologies. And as the product life cycles become shorter and shorter, the design time also become very important: products late to the market cause lose of revenue and market share. Currently the challenge is how to achieve the specified chip performance while keeping the scheduled design time.

In the VLSI design flow, verification and then optimization is almost used in every stage. In all stages before detail routing, design verification inevitably bases on some predictions because of the lack of precise geometry information of wires. But as design enters ultra deep sub-micron and giga-scale era, the divergence between these predictions and the final layout becomes further and further from satisfaction. While after detail routing, which is the *post-layout* stage, because all layout physical information are available, design verifications are able to get accurate results. But the problem is in this stage it is hard to perform design optimizations. Due to the completed detail routing, every layout change is restrained, such as wire sizing, wire spacing, rip-up and reroute, and via relocation, etc. Whether a change can be achieved depends on how much layout resource is available in the local area, which is confined by the surrounding wires; and in most case the local space is too limited to do any layout change. Until now the only solution is back to previous design stages with new design constraints which represent the preferred layout changes. Considering placement and routing usually take days even weeks of CPU time, the additional design iterations for layout change place a heavy burden on the design time issue.

[*]This work is supported in part by MARCO/DARPA Gigascale Silicon Research Center (GSRC).

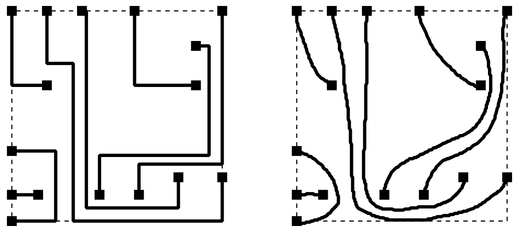


Figure 1: Topological Layout

To solve this problem between design verification and optimization, we propose a new method targeting layout changes in post-layout stage, TEG. TEG can provide a layout change as much as required resources, which is from local area, or the beyond, or even from the whole layout. In addition to actualizing changes for optimization, TEG also keeps intact other parts of the layout. It serves as an incremental layout modification environment for post-layout optimizations. By avoiding design iterations for layout change, TEG preserves the design time with the chip performance.

TEG bases on the topological encoding of the layout, or topological layout. The layout we always mention after the detail routing is actually geometrical layout, in which all layout elements, including wires, pins, vias, obstacles and etc, have specified size, shape and location. Topological layout only captures the relative shapes, positions and connections of elements but no geometry information of the wires; a wire is the spatial relationship with respect to other elements. Figure 1 shows a layout example in geometry and topological. The elimination of wiring geometry releases layout modifications from the local geometry constraints, making them feasible and practical in post-layout stage.

Through TEG, a post-layout optimization works as below:

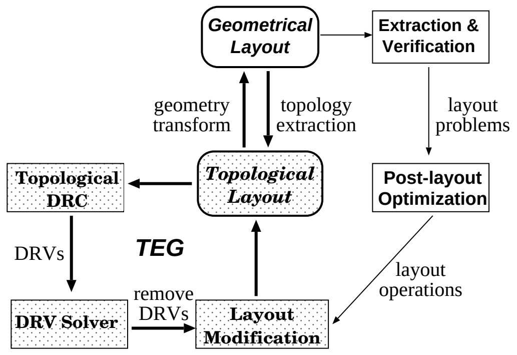


Figure 2: Post-layout optimization through TEG

First, the topological layout is extracted from the geometrical layout; then layout modifications, which are specified by the post-layout optimization, are performed on the topological layout, such as wiring sizing and spacing or rip-up and reroute; a topological design rule check (DRC) and a design rule violation (DRV) solver can be used interactively with layout modifications to ensure the topological layout represents a valid geometry layout; finally the modified layout is converted back into geometry. The flow is illustrated in Figure 2.

The rest of this paper is organized as follows. We first introduce the topological layout encoding, in section 2, including some previous works and our improved encoding model. Then we explain the topological DRC and the DRV solver in section 3. After that we discuss some issues about the modifications on the topological layout in section 4, focusing on an essential operation, vertex moving. At last there are the experimental results and conclusion. Because so far the geometry transform operation is simply based on the work of Staepelaere [11], we didn't include it in this paper.

2. TOPOLOGICAL LAYOUT ENCODING

The concept of topological layout is derived from the research that dealt with topological routing [7]. Topological routing only concerns with determining the rough paths of the interconnect. It provides better results than regular geometry routing in some difficult routing examples [6] [3] [8]. However, since the topological routing algorithm needs to estimate the actual geometry information, it encounters problems while handling large scale IC cases. It motivated other works to use the topological wiring model for cell or layout compaction [9] [13] [2], via minimization [4] and crossing minimization [1].

The widely used topological encoding model is Rubber Band Sketch (RBS). It was first introduced by Leiserson and Maley [7] [9]. Representing wires as rubber-bands allows easy and intuitive manipulation of layout objects. Through RBS, Maley also proposed an algorithm to test the sketch routability. Following his work, Staepelaere [11] proposed a faster geometry transform algorithm. There is also a complete layout system based on RBS, "SURF" [12]. The key advantage of RBS is the layout modification [5]. The modification process is very clear and straightforward because the relationships between the wires and terminals are explicitly represented in RBS. But on the another hand, this explicitness also brings RBS a critical disadvantage: the data size

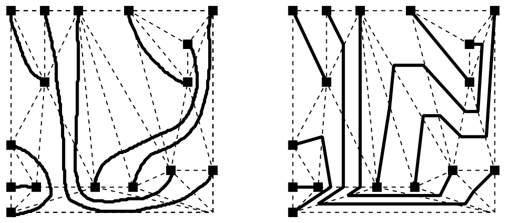


Figure 3: Triangulation Encoding Graph

grows dramatically as the layout size and complexity increases. It limits RBS to the small scale applications only.

Yu [14] proposed a cut-based topological encoding for incremental routability check. A "cut" is a straight line between two non-wire objects in the layout, and it could cross a number of wires in the layout topology. Cuts are the main objects instead of wires in this encoding, so the layout basis is detached from the geometry position of wiring. The number of crossing wires in each cut is used to represent the layout topology. Although data size of this encoding is significantly reduced compared with RBS, the layout operations are hard to perform or need a large amount of calculation, such as layout modifications. The inefficiency limits its usage in practical layout works.

Our topological encoding model targets on both small data size and efficient layout operation supports. It was improved from Yu's Encoding. In this encoding, a layout is described by *Triangulation Encoding Graph*, which is also the origin of the name of our optimization method, TEG.

A ***Triangulation Encoding Graph*** *of a layout instance is a triangulation graph T(V, E). V includes all the terminals, vias, Steiner points, the vertices of obstacles and layout boundary in the layout. Each vertex in V has a geometry position. Each edge in E is embedded with a sequence of the intersecting wires.*

In this encoding, the wiring topology is implicitly represented on a triangulation of the layout. As the example in Figure 3, a triangulation graph is constructed from the topological layout(a). According to wiring topology, each edge is embedded with a sequence of crossing wires (b) (each wire is illustrated as lines between its crossings). For a multi-layer layout, each layer is represented by a encoding graph and different layers are connected by their physical relationship, such as a via crossing two layers.

Through "SURF", the implementation of Yu's encoding and Triangulation Encoding Graph, industry designs showed that using our encoding, the data size of a layout is reduced by 60% on average compared with RBS, and it is only up to 10% larger than Yu's encoding. And Triangulation Encoding Graph has efficient supports for every layout operation required by post-layout optimization flow. The topological layout extraction and design rule check have similar speed as Yu's, at average 10 times faster than RBS; while the the layout modification also runs much faster than Yu's, and even faster than RBS.

3. TOPOLOGICAL DESIGN RULE CHECK AND DRV SOLVER

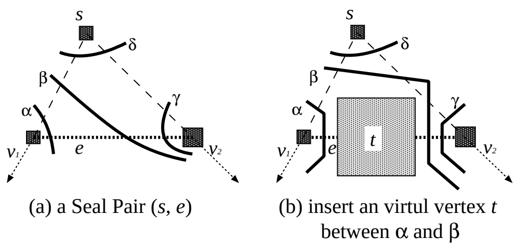


Figure 4: Sealing pair

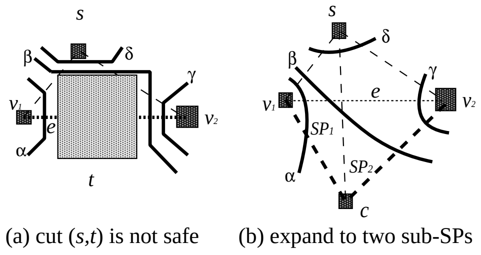


Figure 5: Sealing pair expansion

Through TEG, theoretically any layout modification can be performed. But we need to determine whether the modified topological layout represents a valid geometrical layout, where 'valid' means a planar layout without any DRVs. This is the task of the topological design rule check. A fail topological DRC means the preferred modifications are not acceptable through local available layout resource, but they could be accepted if utilizing the resources from the other part of the layout. The DRV solver can help them to achieve that. This indeed converts the local geometry constraints to the global constraints, bringing TEG advantages over the normal geometry layout methods.

3.1 Cut and Sealing Pair

Like our topological layout encoding model, the topological DRC in TEG is also cut-based. A cut is safe if the flow of the cut is not larger than the capacity [9]. An unsafe cut means there is a DRV between the two vertices of this cut. Leiserson and Maly [7] [9] used the cut safety to determine the routability of the topological sketch, proving the following theorem.

Routability Theorem 1. *A layout is routable if and only if every cut is safe.*

There is $O(n^2)$ cuts inside a layout and each cut crosses $O(m)$ nets, where n is number of vertices and m is the number of wires. So it takes $O(mn^2)$ for one check, unacceptable for a practical DRC algorithm. Valainis and Kaptanoglu et al. [13] introduced a "sealing" concept for design rule check. The "sealing" can help to determine the relationship between the vertex in the center and other vertices. But they cannot prove the relation between "sealing" and the routability of the layout.

We improve "sealing" into a "Sealing Pair" (SP) concept. This improvement speeds up the DRC process because SP is based on the triangles from the encoding graph, while "sealing" requires a pre-calculation for a convex polygon to start with and potentially brings overlapped calculation. And other than the regular rectilinear wiring, SP also supports octilinear and all-angle wiring for versatile routing style. The most important thing brought by "Sealing Pair" is a new layout routability theorem with which the validness of the topological layout can be theoretically determined.

Based on a triangle (s, v_1, v_2), a sealing pair (SP) includes a vertex s and its opposite edge e, as SP(s, e) shown in Figure 4(a). The characteristic of a SP is the status, either sealed or unsealed. In order to determine its status, first of all, we check the three cuts inside the triangle, like (sv_1), (sv_2) and e in Figure 4(a). If any of them is not safe, the SP is unsealed. Otherwise, we insert a virtual vertex t on e, and let t have the largest size while keeping the cut e safe. In Figure 4, assumed using rectilinear wiring pattern, the shape of t is square. It would be octagon for octilinear wiring or circle for the all-angle wiring. We put t at every possible position on e. As Figure 4, there are three crossing nets, $\alpha, \beta and \gamma$, so there are four possible positions for t, between v_1 and α, between α and β, between β and γ or between γ and v_2. We check the cuts between s and t in all cases: if every of them is safe, this SP is sealed. We have proved all these cuts are safe if and only if the two cut at the both ends are safe, i.e., the cut when t is in $[v_1, \alpha]$ or in $[\gamma, v_2]$ in the example. This reduces the running time of this step to constant. Here we skip the proof detail because of its simpleness.

If any cut between s and one t is not safe, as shown in Figure5(a), we need to expend SP in order to determine its status. In the triangulation graph, every edge, except the layout boundary, belongs to two triangles. So we can find the other triangle which share the common edge e, and the opposite vertex c to s, Figure 5(b). Then we swap the diagonal in this quadrilateral (s, v_1, c, v_2) and get two sub-SPs, SP1$(s, (cv_1))$ and SP2$(s, (cv_2))$. We repeat the same process as above to determine the status of SP1 and SP2. If both of them are sealed, SP is sealed. This recursive expansion will continue until we determine the status of SP. If it is not seal, we must have found one cut is unsafe, which is a design rule violation,

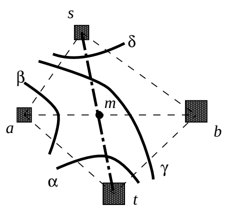


Figure 6: an indirect unsafe cut (s, t)

3.2 Sealing Pair Routability Theorem

Based on the Sealing Pair, we developed a new routability theorem.

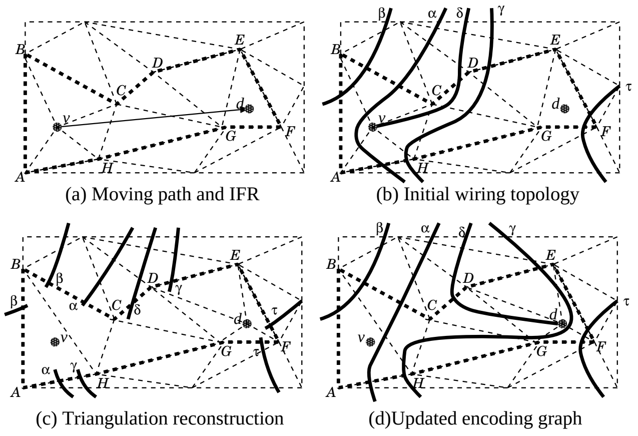
Routability Theorem 2. *A layout is routable if all sealing pairs in the triangulation encoding graph are sealed.*

PROOF Suppose all SPs in the triangulation graph are sealed but there exists at least one unsafe cut. It is apparent that this unsafe cut cannot be a direct cut, e.g. an edge in the graph. Otherwise, the SPs around this cut are unsealed

```
DRV_SOLVER (drvList)

for $\forall d \in drvList$, $d$ is between $v_1, v_2$
    if $d$ is in safe status
    then continue
    if $v_1$ is not frozen
    then move $v_1$ to position $p$ to make $d$ safe
        if more DRVs are created
        then move $v_1$ partially back to avoid new DRVs.
        freeze $v_1$ at current position
    if $d$ is not safe and $v_2$ is not frozen
    then move $v_2$ to position $p$ to make $d$ safe
        if more DRVs are created
        then move $v_2$ back to original position
        freeze $v_2$ at current position
endfor
clear all freezing in this pass.
```

Figure 7: DRV solver algorithm



(a) Moving path and IFR
(b) Initial wiring topology
(c) Triangulation reconstruction
(d)Updated encoding graph

Figure 8: An example of vertex move

as we discussed above. Thus it is an indirect cut in the graph. Let's consider the simplest case, two-edge indirect cut. The "two-edge" indirect cut stands for the indirect cut whose two vertices can be connected through at least two edges in the triangulation graph, as the unsafe (st) in Figure 6, which is not an edge in the triangulation graph. Suppose this cut crosses the edge (ab) at point m, for any situation of wire topology, either cut (sm) or (tm) must be unsafe. Without losing generality, we assume cut (sm) is unsafe. Because SP $(s, (ab))$ is sealed, a virtual vertex v has been tested on the position between net β and δ on (ab) as point m. The cut (sv) has the same flow as cut (sm) and less capacity since the virtual vertex v is not smaller than the point m: so cut (sv) is unsafe. Then two sub-SPs have been expanded and neither is sealed due to the unsafe cut (st) – SP is not sealed, contradicting with our "all sealed" statement. Therefore any two-edge indirect cut must be safe. Similarly we can prove any three-edge indirect cut is safe based on safe direct cuts and two-edge indirect cuts. Finally, all cuts in the triangulation graph are safe: according to Routability Theorem 1, the layout is routable. $\square$

With this theorem, we can perform DRC by checking the status of total $O(n)$ SPs in the layout, where n is the number of vertices. Theoretically, the process to determine the status of a SP could run recursively until reach an obstacle or boundary of the design. However, experiments on real VLSI designs show that there are less than 0.05% SPs need over 4 levels of expansion. And to determine the status of m SPs from the initial layout triangulation, on average $1.1 \cdot m$ sub-SPs are expanded to: the DRC checks $2.1 \cdot m$ SPs in total. So the running time is practically linear to the number of vertices in the layout.

3.3 Design Rule Violation Solver

With the topological design rule check, we could find some DRVs on the topological layout. A DRV means the layout resource is not enough in its local area. The DRV solver can remove a DRV as managing the routing resource [10]: It moves some available resource from other parts of the layout to the violation area; or in other words, moves the DRV vertices toward the area with more resources. It actually gives each layout modification as much resource as possible from global view of the layout.

The DRV solver algorithm is shown in Figure 7. For each DRV in the list, it is an unsafe cut d between two vertices v_1 and v_2 reported by DRC. To make this cut safe, first we move v_1 to a new position in the direction leaving v_2 so that d becomes safe. Then we perform an incremental DRC around v_1: If the total number of DRVs in the layout decreases, we accept the new position of v_1, freezing it so that it will not be moved again in this solver pass; Otherwise, we move v_1 partially back toward its original position so that every new-created DRV can be avoided and also freeze it there. Then we move v_2 in the similar way to make d safe. This algorithm is called repeatedly until no more DRVs can be solved.

4. LAYOUT MODIFICATION ON TEG

With DRC and DRV solver, TEG provides an incremental layout modification environment for the post layout optimization applications. Normally layout modifications include wire sizing, wire spacing, rip-up and reroute, and etc. We can see, all these modifications need to go through DRC and DRV solver to take effect on the final layout. As described in last section, the DRV solver algorithm is based on moving the DRV vertices to remove the DRV, so we must have a fast and robust vertex-moving operation. With vertex-moving, we can also achieve via relocation, obstacle/cell moving, or even buffer insertion. In the following we will discuss the vertex-moving operation in TEG.

4.1 Vertex-moving

The vertex-moving problem is defined as below:

Vertex-moving Problem *Given a triangulation encoding graph S, a vertex v, and a destination d, generate a triangulation encoding graph S', produced by continuous motions of v from its initial position to d.*

Operating within a single routing layer, a vertex is moved by a series of continuous motions which is specified by the moving path. And the influential region (IFR) for a vertex-moving is defined as a set which includes all the triangles that have common points with the moving path. As shown in Figure 8(a), the arrow line (vd) is the moving path and

```
VERTEX_MOVING (Vertex $v$, Destination $d$)

$influReg \leftarrow$ FIND_IFR($v, d$)
$wvList \leftarrow$ GENERATE_WVLIST($iReg$)
RECONSTRUCT_TRIANGULATION( $v, d, iReg$ )
for $\forall e \in wvList$ in order
    if $e$ is a vertex
    then continue
    else $w \leftarrow$ the wire crosses at $e$
        if $e$ is the first crossing of $w$ in $wvList$
        then continue
        else $e' \leftarrow$ first crossing of $w$ in $wvList$
            INSERT_WIRE ($e, e', w, wvList, iReg$)
endfor

INSERT_WIRE($start$, $end$, $wvlist$, $iReg$ )
initial wire stack $ST$
$outWire \leftarrow NULL$
for $\forall e \in wvlist$ from $start$ to $end$ in order
    if $e$ is a vertex
    then for $\forall edge$ connected to $e$ inside $reg$
        put $w$ in the crossing list of $edge$, next to $outWire$
    else if $outWire \neq e$
        then PUSH($outWire, ST$)
            $outWire \leftarrow e$
        else $outWire \leftarrow$ POPUP($ST$)
endfor
```

Figure 9: Vertex-moving algorithm

the bold dot-line illustrates the influential region, which is the polygon ($ABCDEFGH$). It is easy to prove that only the layout topology inside IFR could be changed by this vertex moving.

4.2 Triangulation Encoding Graph Update

To update the Triangulation Encoding Graph inside IFR for a vertex-moving, we need to reconstruct the triangulation as well as the intersecting sequences of new edges. Figure 8 shows an example of the Triangulation Encoding Graph update procedure.

First, we record the initial wiring topology. It is done through a vertex-wire list, which includes the vertices and the wire crossing sequences along the boundary. As Figure 8(b), starting from the cross point between wire α and edge (AH), the vertex-wire list is (α, A, β, B, β, α, C, δ, D, γ, E, τ, F, τ, G, H, γ). Here the wire name stands for a wire crossing. This list keeps the wiring topology by the relative position between wires and vertices along IFR boundary. In the next, we reconstruct the layout triangulation inside IFR by using a standard graph triangulation algorithms, as Figure 8(c). Then, we use this vertex-wire list to generate the intersecting sequences of new edges.

For example, considering the wiring topology of α, we can find the vertex-wire subsequence from the vertex-wire list between two crossings of α, which is (A, β, B, β). There are A and B in this subsequence, meaning that wire α crosses any edge connected to vertex A or B inside IFR: the edge (BH). Wire β is also in this subsequence, meaning for the edge which wire α and β both cross, wire α is after wire β in its wire intersecting sequence. Similarly, considering wire γ, the vertex-wire subsequence is (E, τ, F, τ, G, H) between its two crossings. So wire γ crosses any edge connected to E, F, G or H inside IFR: totally seven edges, (Ed), (Fd), (Gd), (GD), (GC), (HC), (HB). And because wire τ is also in this subsequence, for the edge which both wire γ and τ cross, γ is after τ. So the wire crossing sequence of edge (Fd) is (τ, γ), starting from vertex F.

The main part of the vertex-moving algorithm is outlined in 9. Some implementation details are omitted here, such as selecting the start point of the vertex-wire list, arranging the wires which are incident to the moving vertex, and dealing with the redundant edge crossing for wires.

Table 1: Experimental examples

	dsgn1	dsgn2	dsgn3	dsgn4	dsgn5	dsgn6
Cells	4.9K	8.8K	14.6K	71K	88K	134K
Nets	5.0K	8.8K	17.3K	36.7K	41.8K	41K
Pins	26K	47.5K	76.8K	265K	323K	330K
Layers	3	5	5	4	4	4
TEG	18M	25M	48M	147M	132M	127M

Table 2: Running time of operations in TEG

	dsgn1	dsgn2	dsgn3	dsgn4	dsgn5	dsgn6
Imp.	51	52	79	205	223	207
Mod.	0.08	0.10	0.11	0.12	0.11	0.11
DRC	14	16	19	29	23	25
Solver	2.2	1.8	1.9	2.9	2.6	2.6
Geom.	31	54	121	336	466	384

5. EXPERIMENTAL RESULTS

Our experimental examples (Table 1) are routed and cell-based designs, the scale ranging from 4K cells to more than 130K cells, all in Cadence LEF/DEF format. Most of them are real industry designs. The capability to handle large scale design is one of the advantages of TEG over others topological layout tools. The numbers of cells, nets, pins and metal layers of each design are listed in Table 1, with the memory usage in TEG at the bottom. As the number of cells, nets and pins increase, the memory usage increases in an acceptable rate.

In the experiment, at first we import each example into TEG. Next, some selected modifications are performed on the layout. We run the DRC and DRV solver to ensure that the modified layout can be transformed back into a valid geometry layout. Then, we generate a new DEF file from the modified design. In the end we use Cadence Silicon Ensemble load the new DEF file and verify its connectivity and geometry. Silicon Ensemble doesn't find any connectivity or geometry problem from the new DEF files except some antenna issues which are initially from the original design. TEG is running on a Intel Pentium III 550MHz.

One experiment targets on demonstrating the running time of the basic layout operations in TEG. We simply perform 500 random-selected via-moving through each design. The results are in Table 2, all numbers in seconds. 'Mod.' is the average time for one via-moving operation. 'DRC' is the total time used by topological DRC after these 500 via-movings, in which on average 30000 sealing pairs are processed, 'Solver' is the average running time to remove one DRV. This table indicates that the running time of design import and geometry transform increases as the scale of

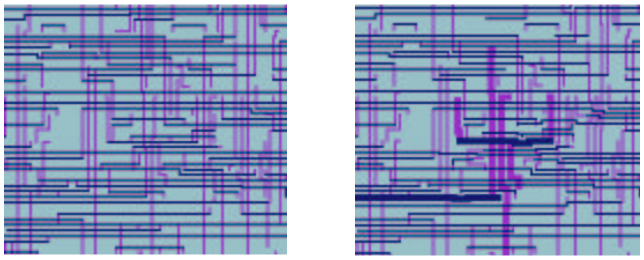

(a) Original layout (b) After wire sizing

Figure 10: Wire sizing through TEG

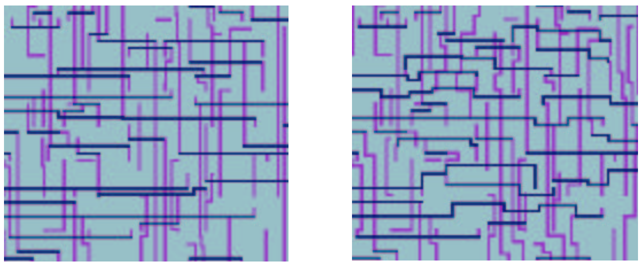

(a) Original layout (b) After wire distribution

Figure 11: Wire distribution through TEG

the design becomes larger. And the running time of vertex-moving, topological DRC and DRV solver is almost constant in each case.

In another experiment, we perform wire sizing on the layout. Increasing the wire width is widely used in post layout optimization such as crosstalk-delay fixing, IR-drop reduction and etc. We randomly picked up 0.5% wires from the original layout, then change their width to 3× minimum width, followed by DRC and DRV solver. Figure 10 shows part of example "dsgn1", displaying only metal layer 2 and layer 3. In the modified layout(b), we can see the vias around the widened wires have been moved outward by the DRV solver.

We pick up wire distribution in the third experiment. Wire distribution increases the wire spacing if there is more space available than the minimum. It can be used to improve the layout yield and manufacturability, or decrease the metal density difference for less chemical-mechanical polishing(CMP) process variation. Figure 11 shows part of example "dsgn1", This example is also from design "dsgn1", displaying metal layer 2 and layer 3. After wire distribution Figure 11 (b), the wire density in both layer become more uniformed without rerouting any wires and filling any dummy metal.

6. CONCLUSION

In post layout stage of the VLSI design, the problem is not whether we can verify and analyze the design precisely, or what and where the layout problem is, or what optimization or modifications should be performed on the layout: many EDA tools are available to do these; the problem remains on how modifications can be achieved with geometry constraints all over the layout. In most cases, even the small change is not acceptable due to the limited local resources.

In this paper a new method TEG was proposed to solves this problem by processing the layout topologically. With a improved topological layout encoding model and a set of efficient layout operation procedures, including layout extraction, design rule check, DRV solver, vertex-moving and geometry transform, TEG provides an incremental layout modification environment for the post layout optimization. The experimental results showed TEG was efficient and effective in processing industry VLSI designs.

7. REFERENCES

[1] H.-F. S. Chen and D. T. Lee. On crossing minimization problem. *IEEE Trans. Computer-aided Design*, 17(5):406–418, May 1998.

[2] H.-F. S. Chen and D. T. Lee. A faster one-demensional topological compaction algorithm withjog insertion. *Algorithmica*, 28(4):390–421, December 2000.

[3] J. Cong, M. Hossain, and N. A. Sherwani. A provably good multilayer topological planar routing algorithm in IC layout design. *IEEE Trans. Computer-aided Design*, 12(1):70–78, January 1993.

[4] J. Cong and C. L. Liu. On the k-layer subset and topological via minimization problems. *IEEE Trans. Computer-aided Design*, 10(8):972–981, August 1991.

[5] W. W.-M. Dai, R. Kong, J. Jue, and M. Sato. Rubber band routing and dynamic data representation. In *Proc. Intl. Conf. Computer-aided Design*, pages 52–55, Santa Clara, CA, November 1990. IEEE Computer Society.

[6] S. Haruyama, D. Wong, and D. S. Fussell. Topological channel routing. *IEEE Trans. Computer-aided Design*, 11(10):1177–1197, October 1992.

[7] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proc. 17th Ann. ACM Symp. Theory of Computing*, pages 69–78, New York, NY, 1985. ACM.

[8] A. Lim, V. Thanvantri, and S. Sahni. Planar topological routing. *IEEE Trans. Computer-aided Design*, 16(6):651–656, June 1997.

[9] F. M. Maley. *Single-layer wire routing and compaction*. MIT Press, Cambridge, MA, 1990.

[10] P. Morton and W. Dai. Routing resource management for post-route optimization. Tech Report UCSC-CRL-02-12, University of California, Santa Cruz, February 2002.

[11] D. Staepelaere. Geometric transformations for a rubber-band sketch. Master's thesis, University of California, Santa Cruz, 1992.

[12] D. Staepelaere, J. Jue, T. Dayan, and W. W.-M. Dai. Surf: A rubber-band routing system for multichip modules. *IEEE Design and Test of Computers*, December 1993.

[13] J. Valainis, S. Kaptanoglu, E. Liu, and R. Suaya. Two-dimensional IC layout compaction based on topological design rule checking. *IEEE Trans. Computer-aided Design*, 9(3):260–275, March 1990.

[14] M.-F. Yu. *Topological Encoding and Interchangeable Pin Routing*. PhD thesis, University of California, Santa Cruz, December 1997.