# Topology-Driven Cell Layout Migration with Collinear Constraints

De-Shiun Fu, Ying-Zhih Chaung, Yen-Hung Lin and Yih-Lang Li
*Computer Science Department National Chiao Tung University Hsin-Chu 300, Taiwan*
*gis93539@cis.nctu.edu.tw;starshow999@gmail.com;* homeryenhung@gmail.com*; ylli@cs.nctu.edu.tw*

***Abstract*—**

**Traditional layout migration focuses on area minimization, thus suffered wire distortion, which caused loss of layout topology. A migrated layout inheriting original topology owns original design intention and predictable property, such as wire length which determines the path delay importantly. This work presents a new rectangular topological layout to preserve layout topology and combine its flexibility of handling wires with traditional scan-line based compaction algorithm for area minimization. The proposed migration flow contains devices and wires extraction, topological layout construction, unidirectional compression combining scan-line algorithm with collinear equation solver, and wire restoration. Experimental results show that cell topology is well preserved, and a several times runtime speedup is achieved as compared with recent migration research based on ILP (integer linear programming) formulation.**

## I. INTRODUCTION

Manufacturing technology has been working on decreasing feature size. Handcrafted re-design of a chip layout and cell library conforming to new manufacturing technology design rules spends considerable time. Moreover, a cost-down issue emerges from the product in severe competitions. Seeking a semiconductor foundry with decreased manufacture cost is an alternative solution. Two semiconductor foundries generally vary in the two manufacture technologies design rules of the same generation, and it demands numerous layout modifications to satisfy the target design rules.

Traditional goal of layout migration is to efficiently shrink and place each component in the layout as compact as possible. Currently available layout migration algorithms were classified into constraint graph based [1, 2, 3] and integer linear programming based [4, 5, 6, 7] algorithms. Constraint graph based algorithms usually scans all components via a virtual scan line of the design and then constructs a constraint graph. New design rules are then employed on the graph to identify the position of every component with the longest path algorithm. However, the conventional constraint graph algorithm intends to produce a compact layout with numerous changes in interconnection shapes and topologies because it only considers space utilization. The resultant changes in the shape and topology degrade the timing delay and other properties. With original design intention destroyed, designers must spend time comprehending and modifying the migration layout. The bulky burden lowers the availability of layout migration. Heng *et al.* proposed a minimum perturbation (MP) objective function to preserve original design intention by minimizing the position changes of all edges [6]. To improve the MP objective function, Fang *et al.* considered that the effect of geometric changed by accumulating the effect of position changing in subsequently processed objects [7]. The integer linear programming approach can not guarantee that the relative relations among an object, and its neighbors are kept unchanged. For example, an object *A* initially placed on the right top of the other object *B*, and the migration algorithm based on the metric of minimum position changes probably places *A* on the right side of *B*. Besides, ILP based approaches are time-consuming, and thus will take a long time to complete migration as the number of layout objects and ILP constraints increase.

Another well-know research about compaction is topological layout model [8, 9]. Topological layout model using rubber-band sketch (RBS) is proposed by Leiserson *et al.* and focuses on the topological relation between wires and objects rather than physical geometries and the imposed design rules [8]. A topological layout is composed of topological points and topological wires, where object polygons are regarded as topological points. A topological wire only stores relative positions to adjacent topological wires and points instead of physical wire size and position. Zhang *et al.* proposed a topological layout model, called triangulation encoding graph (TEG), to partition the entire layout into many triangles. Topological wires are presented by a sequence of passing edges [9]. TEG feasibly handles post-layout optimization when only wires and contacts are under consideration. For topology-driven layout migration, TEG has difficulty to control the topology alteration since a two-dimensional move of a node is allowed. Even if a node is moved unidirectionally, the alteration of a triangle will be non-monotonic since every triangle has at least one sloping edge and at least two edges connecting to a node.

This study proposes a two level hierarchical topology-driven cell migration system. A cell layout is regarded as a composition of devices and interconnections. Conventional edge-based constraint graph algorithm [10] shrinks device layout to meet new design rules while interconnections and devices are transformed into the proposed rectangular topological layout for interconnection-level migration. Corner stitching data structure fixes the corner relations. During migration, the entire layout can be regarded as a compressed sponge. Corner stitches and collinear constraints keep the layout topology unchanged. Experimental results show that cell topology is well preserved, and it hastens runtime several times as compared with recent migration research based on ILP formulation. The rest paper is organized as follows. Section 2 briefly reviews topological layout and our proposed system. Section 3 presents the layout extraction and device migration. Section 4 proposes the rectangular topological layout framework. Section 5 presents our topology-driven migration. Section 6 discusses the cell migration experiments. Finally, Section 7 draws conclusions.

## II. PRELIMINARIES

### *A. Topological Layout*

A physical geometrical layout thoroughly describes the size, shape, and location of each wire, and it identifies the design rule violation clearly. Conversely, in a topological layout, a wire is represented by its relative positions (such as the wire next to this wire) and connection points. This representation relaxes the constraints of design rules on the layout objects and modifies the layout more flexibly. Topological layouts initially perform topological routing [8]. The routability theorem ensures that a topological layout with modifications that pass a routability test can feasibly converted back to a geometry layout. A geometrical layout is transformed into a topo-

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

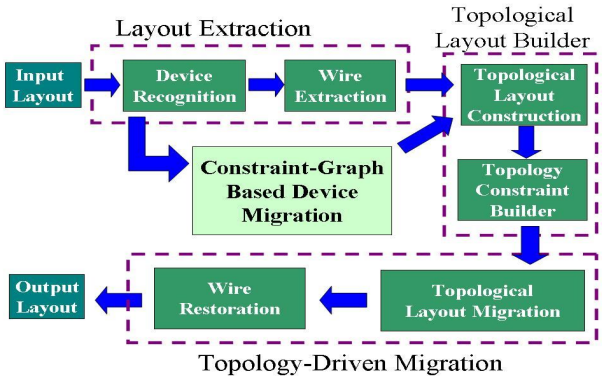


Figure 1: System Flow.

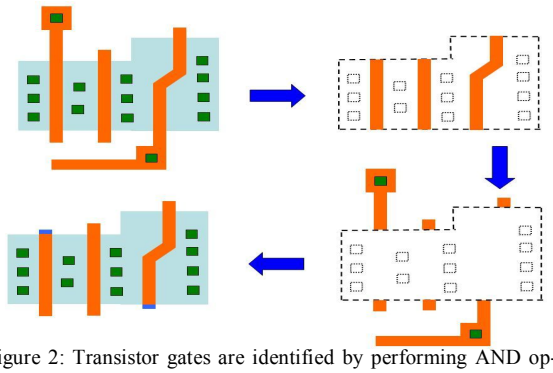

Figure 2: Transistor gates are identified by performing AND operation on polysilicon and diffusion layers. Polysilicon wires are obtained by removing transistor gates from the polysilicon layer. Finally, a transistor gate is inserted via a small extension when it connects to a polysilicon wire.

logical layout via layout encoding, which determines the required memory size, the efficiency of topological layout creation, and operations on a topological layout.

### B. System Flow Overview

As Fig. 1 shown, the proposed migration system comprises four components: *layout extraction*, *topological layout model builder* (TLMB), *constraint-graph based device migration,* and *topology-driven migration*. Since the input layout is composed of polygons, the layout extraction first recognizes devices from polygons by geometrical Boolean operations. After device recognition, remainder polygons will suppose to be wire polygons; and then, the centerline of wire polygons will be extracted from wire extraction stage. All polygons of the devices are sent to device migration process which migrates devices to the target technology by conventional edge-based constraint graph compaction algorithm. Device migration would produce original devices and migrated devices. Before topology-driven migration, the rectangular topological layout is built according to wire centerline and components of device. When building the original topological layout, *topology constraint builder* constructs topological constraints according to the original topological layout in the corner stitching data structure. Those topological constraints ensure that layout topology remains consistent before and after migration. *Topological layout migration* is then realized by invoking tile scan-line algorithm and back-trace mechanism. Finally, physical wires are restored within associated space tiles to transform the topological layout into a physical layout.

## III. LAYOUT EXTRACTION AND DEVICE MIGRATION

### A. Device Recognition And Wire Extraction

Algorithm wire_extraction (Polygon *P*, int *base_width*).
1. *min_width* = Identify_Minimum_Width(*P*);
2. *wire_width* = *base_width* + *min_width*;
3. Shrink polygon *P* by *min_width*;
4. Identify and report all centerlines from *P*;
5. $\beta = P$ – {the points of extracted centerlines};
6. **for** (every polygon $p_i \in \beta$ )
7. wire_extraction($p_i$,*wire_width*);

Figure 3: Wire extraction algorithm.

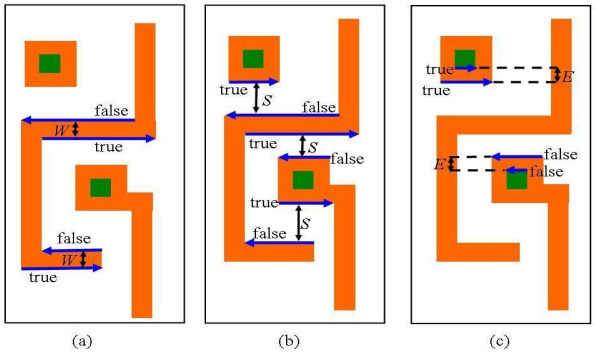


Figure 4: Design rule checking by traversing the point arrays of all polygons. (a) MinWidth *W (F, T)/(T, F)*; (b) MinSpace *S (T, F)/(F, T)*; (c) MinEnc *E (F, F)/(T, T)*.

All devices are extracted by geometrical Boolean operations. For example, diffusion contacts are identified by performing an AND operation on contact and diffusion layers; transistor gates can be obtained in a similar manner—by replacing the contact layer with the polysilicon layer. The OR operation generates transistor devices. Notably, polysilicon wires are viewed as gates and wires. The part of a polysilicon wire used for interconnection is considered as a wire and split from the transistor gate via the NOT operation. Figure 2 displays the process of breaking a polysilicon wire into two parts. The top end of the most left polysilicon wire and the bottom of the most right polysilicon wire slightly extended to indicate a connection point for further routing (Fig. 2). In addition to transistor devices, the contacts are devices in metal layer.

The wire extraction procedure obtains wire information from wire polygons, such as the centerline position and the width of a wire. Only the work by Lakos [6] addressed wire extraction. This study proposes a fast and efficient polygon-to-wire extraction algorithm based on iterative polygon shrinking. Figure 3 displays the proposed wire-extraction algorithm, which repeatedly shrinks a processed polygon based on minimum wire width. The first iteration of the *wire_extraction* function sets *base_width* to zero. After shrinking the polygon by using minimum wire width, the polygon become centerlines, and the others are under-sized.

### B. Constraint-Graph Based Device Migration

This stage uses the edge-based constraint-graph compaction algorithm [11, 12] to construct a constraint graph. The edge-based constraint graph model has some advantages: first, this model can deal with device design rules and produce a minimal migrated device layout. Second, the device topology can be preserved by introducing new constraints. Every edge of a polygon is first traversed in anti-clockwise direction as shown in Fig. 4a. Rightward and downward edges are set as *True*, whereas leftward and upward edges are set as *False*. Space, width, and enclosure rules are thus recognized by the state of an edge pair, represented by ($S_{tl}$, $S_{br}$), where $S_{tl}$ is the state of the top-left edge of a horizontal/vertical edge pair, and $S_{br}$ is the state of the other edge. For example, the (*F*, *T*)/(*T*, *F*)

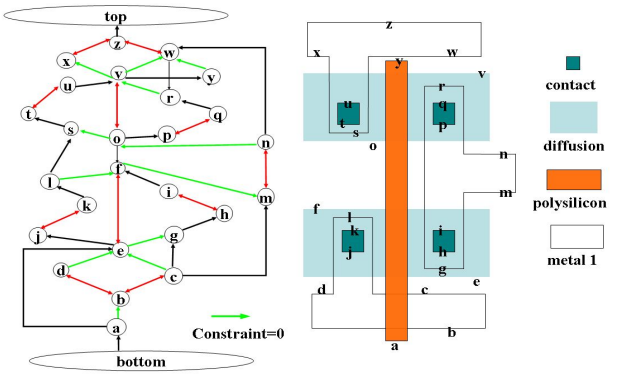


Figure 5. An inverter layout and its edged-based constraint graph.

state of a horizontal/vertical edge pair defines a width rule; the (*T*, *F*)/(*F*, *T*) state of a horizontal/vertical edge pair defines a space rule; and the (*F*, *F*)/(*T*, *T*) state of a horizontal/vertical edge pair defines an enclosure rule.

Because *2-D* migration is an *NP-complete* problem [13], we adopt a *1-D* edge-based constraint-graph compaction algorithm. The vertical migration algorithm is as follows. The perpendicular-plane sweep algorithm [10] is enhanced by introducing topological constraints. A node in the constraint graph represents a horizontal edge of a polygon. If a horizontal line overlaps another horizontal line, a constraint edge is inserted to connect their nodes. Three types of constraint edges are defined in this work. The first type is novel, and the other two types are conventional constraint edges.

1. *Topology constraint edge*: This edge imposes a topological order on two nodes even when their associated shapes have no design rule between the nodes. The conventional constraint graph based algorithm does not create a constraint edge between two shapes in different layers when the shapes have no design rule.
2. *Two-way constraint edge*: This edge primarily fixes the distance between two layout edges. For example, we set two-way constraint edge in a metal wire because its width is immobile.
3. *Basic constraint edge*: This edge is a one-way edge for maintaining space rules.

During the vertical compression, a virtual scan line scans a device upwards to generate its edge-based constraint graph. Figure 5 presents the layout of an inverter and its edge-based constraint graph. The green edges in Fig. 5 indicate topological constraints; the red edges are two-way constraints, and the black edges represent basic constraints. When we complete the constraint graph, the longest-path algorithm determines the lowest position of each horizontal edge.

## IV. RECTANGULAR TOPOLOGICAL LAYOUT MODEL

In this section, we introduce the method to construct a rectangular topological layout and topological constraints. To understand our method easily, some nouns would be defined under following.

**Definition 1 (Isomorphic Tile Planes).** *Two layouts which are constructed by maximum stripped tile plane are said to be isomorphic if they have the same tiling structure. These means that a tile in a tile plane could be one-by-one mapped to another tile plane and their neighboring relation could not change.*

**Definition 2 (Tile Topology).** *In maximum stripped tile plane, every space tile could represent topological relation between two block tiles. Those topological relations are called tile topology.*

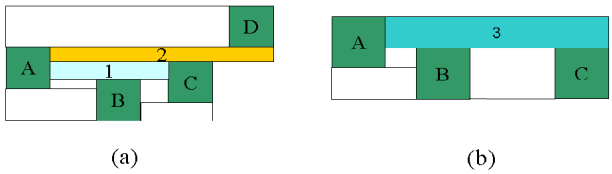


Figure 6. Topology constraints characterized by space tiles. (a) Space tiles 1 and 2 retain "overlap" and "on-the-top" topology constraints between devices *A* and *D* and devices *D* and *C*, respectively; (b) space 3 retains the "align-on-the-top" topology constraint of devices *B* and *C*.

For instance, in Figs. 6(a) and (b), tiles *A, B, C,* and *D* are block tiles, and the other tiles are space tiles. Figure 6(a) displays the first two constraints. Tile 1 retains the "overlap" property between block tile *A* and *C*, while tile 2 retains the "on-the-top" property of block *D* to device *C*. Figure 6(b) displays the topology constraint of devices *B* and *C* aligning on their top.

**Lemma 1.** *If two layouts are tile isomorphism, their tile topology is identical absolutely.*

### *A. Topological Layout Construction*

Because maximum stripped tile plane can represent tile topology, our rectangular topological layout is constructed by corner-stitching data structure. The input data comes from device-level migration and wire extraction. Device-level migration produces and delivers the following information: device layer, the device position in the original layout, the original shapes of devices, the new shapes of devices after migration, and routing demands on every boundary. Wire extraction derives the layer, centerlines, and width of every wire segment. The devices and wires in the proposed rectangular topological layout are introduced below. Every tile contains tile type, tile height and width before cell migration. Every device in the topological layout is composed of block tiles. Because every device was migrated by *constraint-graph based device migration* process, block tile height and width after cell migration are given. Since one-dimensional migration operation is invoked in every pass, it has to construct both the maximum horizontal stripped and vertical stripped tile planes simultaneously. Moreover, the change in one-pass migration needs to be transferred to its adjacent tile planes, and thus every tile needs to recognize its related tiles on adjacent tile plane to synchronize the change.

Topological wires lack for physical information, and the wires are characterized by the tiles that they have passed. To increase the availability of topological wires, physical segments need to represent all wire segments in a tile and to attach them in their tile. A physical segment denotes the centerline of a wire with its width, and it can be horizontal or vertical. A horizontal physical segment can only connect to a vertical physical segment, and devices. Every space tile contains all physical segments, which pass this space tile. Every physical segment has its position and reference objects to indicate the connectivity of the net. A reference object of physical segment implies continuous connectivity of a net, while a reference object of a block tile (device) implies the connection of a wire to a device.

The two endpoints of a horizontal/vertical physical segment are only identified by the segment's *y*-coordinate/*x*-coordinate values since the other values can be derived from its preceding and following reference objects. By this scheme, a physical

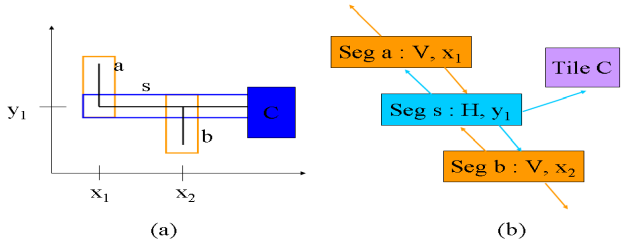


Figure 7. Segment structure. (a) A net connecting to device *c* is described by three physical segments; (b) segment *s* has three reference objects –segments *a* and b and tile *c*.

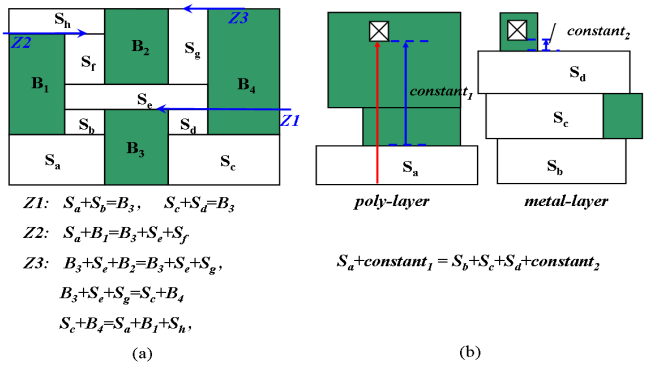


Figure 8. Collinear topology constraints. (a) Three collinear equations are established on the tile plane; (b) one equation is established for the contact position alignment in polysilicon and metal layers.

wire can be quickly deduced from a series of reference object traversal starting at any physical position. The new positions of physical segments have to re-compute during migration. Figure 7 shows the structure of the physical segments and reference objects, where physical segments *a*, *b,* and *s* denote partial wire segments of a net connecting to device *c*. Figure 7(b) shows the relation between physical segments and their reference objects.

Besides general wire segment, power and ground (P/G) wires have difficulties to construct in a topological cell layout since P/G wires are fat wires containing many contact devices. A topological wire connects with two points, while P/G wires are pairs of straight wires, which have more than two contact devices. Therefore, virtual P/G block tiles and additional flags are adopted in this study. Because P/G wires only connect to metal layer's contact devices, we add an additional flag, say *g*, to some contact block tiles, to indicate whether the tile has any passing wires connecting with P/G wires. During migration, a block tile without flag *g* should be located above the ground wire with the distance of ground wire width and metal separation. Only the tile with flag *g* can be moved downwards. This rule can avoid moving the devices not connecting with the ground wire towards the ground wire so that design rule violations occur.

In the beginning, P/G wires extracted their centerlines, like general wire segment; and then, two virtual block tiles are added on the top and bottom of the topological layout to replace P/G wires. The centerlines of wire segments which connect contact block tiles and P/G wires are extracted and connect P/G virtual block tiles and contact block tiles.

### *B. Collinear Constraint Construction*

In order to guarantee that migrated layout and original layout are isomorphic tile planes, we construct collinear constraints according to original layout. Every space tile is assigned a variable to represent its length along the compression direction. When two neighbor tiles' ceilings are collinear,

```
Algorithm vertical_migration.
1. // TileList: tile list array, TileList[i] contains all tiles on
2. // layer i's tile plane in enumeration order;
3. // Ta: currectly processed tile; ha: tile Ta's height;
4. for each layer i
5.       TileList[i] = upward enumeration order on layer i's tile plane;
6. while there is unvisited tile in tile list array TileList {
7.       tile Ta = the lowest tile in tile list array TileList;
8.       compute Ta's minimum height (ha) to accommodate all
         embedded wires using new design rules;
9.       if new ha value can not conform to all collinear equations
         containing ha
10.          back-trace related tiles appearing in mismatched collinear
             equations and adjust tile height to equalize the values of all
             tile height functions of the mismatched collinear equation;
11. } // end while
12. updateTileplane( );
```

Figure 9. Vertical migration algorithm.

a collinear equation is formed. This equation means that the summation of tiles' height until those two tiles must be equal. For example, Fig. 8(a) contains six collinear equations on the line *Z1, Z2,* and *Z3*. Considering tile height $S_e$ of tile height functions *Z2* and *Z3*, constant function $B_3$ is chosen as the basis for further tile height derivation, since it is treated as constant after formulating *Z1*. For instance, the height function of tile $S_f$ is defined as $B_3+ S_e$ rather than $S_a+ S_b+ S_e$ or $S_c+ S_d+ S_e$. Thus, the collinear equation based on the line *Z2* is $S_a+ B_3= B_3+ S_e+ S_f$. Finally, the top boundary also forms a collinear equation.

A vital issue in a layout contact is how to synchronize the contact positions in two adjacent layers. Therefore, a collinear equation needs to link the variables adopted to compute the contact position in two adjacent layers. In Fig. 8(b), the contact position alignment in polysilicon and metal layers is realized by introducing an equation $S_a+consant_1= S_b+ S_c+ S_c+constant_2$.

## V. TOPOLOGY-DRIVEN MIGRATION

The proposed migration algorithm in this study invokes iterative one-dimensional migration. One-dimensional migration involves two stages: topological layout migration and wire restoration. In previous stage, we assign a set of variable to represent tiles' height, so the migration problem can reformulate to determine every tile's height. The vertical migration algorithm is as follows. From *definition 2,* we know that the MHS (maximum horizontal stripped) tile plane keeps the vertical topology, therefore vertical migration works on MHS tile plane.

### *A. Topological Layout Migration*

Figure 9 illustrates the vertical migration algorithm. First, upward enumeration operation on all tile planes decides tile migration order. According to the tile order, migration algorithm computes every tile's height. If the tile is block tile, the tile height which is determined from device migration will be assigned. If the tile is space tile, the minimum height of tile $T_a$ is evaluated by tile scan-line algorithm to accommodate all embedded wires by using new design rules. To comply with all related collinear equations, $T_a$ is generally a value above its minimum value. If $T_a$ cannot feasibly be assigned a value to hold all related collinear equations, then the variables that invalidate collinear equations are identified, and the computation process returns to the related tiles and adjusts their tile heights to equalize all tile height functions of every mismatched collinear equation. Once a tile's height altered, the

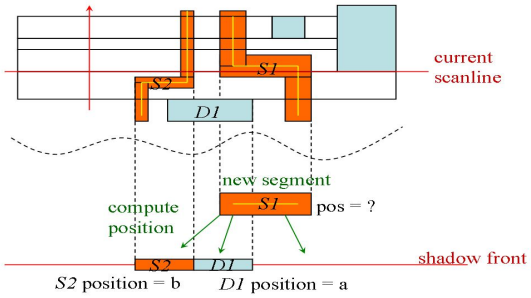


Figure 10. Scan line algorithm moves scan line upwards. Currently processed object *S1* determines its lowest position by examining the positions of the objects in shadow front, say *D1* and *S2*, and their legal separation.

process of checking related collinear equations, back-tracing, and updating tile height variables repeats until all collinear equations hold. Finally, the plane is updated and restructured after vertical migration to reflect tile under-sizing during migration.

*1) Tile Scan-Line Algorithm*

Upward scan-line algorithm provides the objects intersected by the scan line for processing. The shadow front, which is initially empty, continuously stores the objects closest to the advancing scan line. If a newly coming object into the shadow front totally covers an existing object in the shadow front, then the existing object removed from the shadow front, because it can no longer be seen from the scan line. It constituted by the bottom border of current tile and shadow front. Namely, the current object cannot be assigned at a lower position since moving it to another tile may destroy the wire topology.

The objects that must be processed by the scan line algorithm are the segments embedded in the currently processed space tile, which are sorted in increasing order of their *y*-coordinates. Every segment sequentially computes its lowest legal position under the new design rules. Diagonal separation checking and other constraints, including connectivity constraints, must be also considered. After the segment is assigned its new position, the shadow front is updated by inserting the contours of the segment. In Fig. 10, when segment *S1* is being processed, the shadow front contains the contours of segment *S2* and device *D1*. Notably, the left endpoint of device *D1* shifts rightwards because the reduced range is covered by segment *S2*. The new height of the tile can be determined when all the segments in the currently processed tile have been processed, and their new positions have been computed. The final tile height has to consider potential separation influence from its top neighboring tiles.

*2) Back-Trace Mechanism*

The tile height determined by scan line algorithm is a lower bound under the current shadow front. If the tile height variable, $h_a$, can be assigned a value not less than the lower bound in order to comply with related collinear equations, then $h_a$ is a successful set. Conversely, if $h_a$ has to be assigned a value below the lower bound in order to comply with related collinear equations, then $h_a$ setting is infeasible. In this case, if $h_a$ is assigned to its lower bound value, then its tile height function value is greater than other tile height functions, which means that some previously assigned variables have to reassigned a new larger value to comply with the lower bound constraint induced by the embedded wires. Figure 11 shows this case, where the assignment of the $H_E$ of tile *E* with its lower bound makes the left tile height function be greater than the right tile height function. Reassigning the variables in other tile height functions back-traces the variables appearing in related collinear equations. The back-tracing order follows the reverse

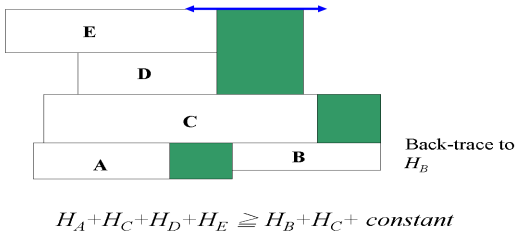


Figure 11. Back-tracing example. After scan-line algorithm determines tile *E's* minimum tile height, the collinear equation does not hold. Back-tracing mechanism increases the lower bound of tile

tile enumeration order.

To remedy the problem of mismatched collinear equations, tile height variables are categorized into two types, namely free variables and constant variables. A free variable can be freely assigned any value not less than its lower bound, while a constant variable has a fixed value after equation derivation, most likely a device tile's height. The lower bound of a free variable is defined by the scan-line algorithm or the back-tracing mechanism. In Fig. 11, tiles *A*, *B*, *C*, *D,* and *E* are sequentially processed to derive their tile heights. After determining the minimum tile height of tile *E*, tile height variables $H_A$, $H_B$, $H_D$ and $H_E$ are free variables, and $H_C$ is a constant variable. The lower bounds of all free variables are currently defined by the scan-line algorithm. However, a new $H_E$ breaks this collinear equation, therefore the back-tracing mechanism is initiated. In this case, variable $H_B$ and its lower bound both increased to equalize the two tile height functions. Notably, adjusting $H_B$ initiates the checking of other collinear equations containing $H_B$, and it causes new back-tracing mechanisms to start if these collinear equations do not hold. This process repeated until all collinear equations hold.

### *B. Wire Restoration*

Since the segments store helpful physical information, wire restoration becomes straightforward and simple. It simply links connected segments to form wires, and creates power and ground wires.

Because all tiles migrated one by one, the number of tile would not change after layout migration. Moreover, collinear constraints preserve tile isomorphic property. Therefore, from *lemma 1,* the tile topology of the migrated layout is the same with the original layout.

## VI. EXPERIMENTAL RESULTS

The proposed migration system has implemented in C/C++ language on a PC operating at 2.8*GHz* with 992*MB* memory. The cell layouts for test come from a standard cell library which used 0.18*um* technology. The proposed migration system compresses the cell layout from 0.18*um* technology to 90*nm* technology. In Table 1, the column "No. of traversed tiles" is the total number of traversed tiles in all tile planes during topological layout migration, and the column "No. of back-tracing" is the number of back-tracing for solving the inequalities of the tile height functions of a collinear equation. The runtime for topological layout migration includes three parts: device-level migration, preprocessing, and topological migration, where preprocessing includes layout extraction and topological layout construction. The runtime of device-level migration is proportional to the constraint graph size.

The runtime of topological layout migration mainly depends on the number of traversed tiles and has no direct connection with the number of equations. For example, cell AND3X2 needs 41 back-tracings while it only cost 0.063 seconds. On

the other hand, BUFFX20 produces the most equations, but its runtime is less than that of cell DFFX2. Equation number connects with the runtime indirectly because increasing number of equations may increase the number of back-tracing.

Table 2 compares the experimental results of rectangular topological layout migration with Calligrapher. Calligrapher was running on a SUN Blade 100 system operating at 500*MHz*. The source and target technologies are 1.2*um* and 0.25*um*. Since the results of two migration systems are obtained from different cell layouts and run on different computer platforms, only similar designs are selected for comparison. The runtime of Calligrapher is composed by generating constraints and solving ILP. Our runtime is much faster than Calligrapher several times after normalization. Figure 12 displays the layouts of cell DFFX2 before and after migration. The layout after migration is in the left part.

## VII. CONCLUSION

This work presents a new rectangular topological layout for topology-driven cell migration. Topological layout migration combines its flexibility of handling wires with traditional scan-line based compaction algorithm for area minimization.

Beside, our migration algorithm guarantees that the tile topology of original layout and migrated layout is not change. Experimental results show that cell topology is well preserved, and a several times runtime speedup is achieved as compared with recent migration research based on ILP formulation.

## REFERENCES


[1]. Gershon Kedem and Hiroyuki Watanabe, "Graph-Optimization Techniques for IC Layout and Compaction", in *Proceedings of the 20th annual conference on Design Automation*, pp.113 –120, 1983.

[2]. David G. Boyer and Bellcore, "Process Independent Constraint Graph Compaction", in *Proceedings of the 29th annual conference on Design Automation*, pp. 318 –322, 1992.

[3]. Joseph Dao, Nobu Matsumoto, Tsuneo Hamai, Chusei Ogawa, and Shojiro Mori, "A Compaction Method for Full Chip VLSI Layouts", in *Proceedings of the 30th annual conference on Design Automation*, pp. 407 –412, 1993.

[4]. So-Zen Yao, Chung-Kung ChengT, Debaprosad Dutt, Surendra Nahar and Chl-Yuan Lo, "Cell-Based Hierarchical Pitchmatching Compaction Using Minimal LP", in *Proceedings of the 30th annual conference on Design Automation*, pp.395 –400, 1993.

[5]. Fang Fang and Jianwen Zhu, "Automatic Process Migration of Datapath Hard IP Libraries", in *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pp. 887 –892, 2004.

[6]. Fook-Luen Heng, Zhan Chen, and Gustavo E.Tellez, "A VLSI artwork legalization technique based on a new criterion of minimum layout perturbation", in *Proc. Int. Symp. Physical Design*, pp.116 –121, 1997.

[7]. Fang Fang, Jianwen Zhu and Qianying Tang, "Calligrapher: A New Layout Migration Engine Based on Geometric Closeness", *IEEE Transaction on Computer-aided Design of Integrated Circuits and Systems*, vo. 24, pp.1347 –1361, Sep., 2005.

[8]. C.E. Leiserson and F.M. Maley, "Algorithms for routing and testing routability of planar VLSI layout", in *Proccedings of the seventeenth annual ACM symposium on Theory of Computing*, pp.69 –78, 1985.

[9]. Shuo Zhang and Wayne Dai, "TEG: A new post-layout optimization method", *IEEE Transaction on Computer-aided Design of Integrated Circuits and Systems*, vol. 22, pp. 446 –456, Apr., 2003.

[10]. Jiaji Fang , Joshua S. L. Wong', Kaihe Zhang, Pushan TangA, "A New Fast Constraint Graph Generation Algorithm for VLSI Layout Compaction",*IEEE International Symposium Circuits and Systems*, pp.2858 –2861, Jun., 1991.

[11]. Shuilong Chen, Xiangqing He and Zhilian Yang, "Edge based Layout Compaction," in *Proceedings of the 4th annual conference on ASIC*, 2001, pp. 198-201.

[12]. Michael A. Riepe and Karem A. Sakallah, "The Edge-Based Design Rule Model Revisited," *ACM Transactions on Design Automation of Electronic Systems*, 1998, pp.463-486.

[13]. Williams, J. D. "STICKS – Graphics Editor for High-Level LSI Design," in *Proceedings of National Computer Conference*, 1978, pp. 289-295.


| Cell Name | Migration | | Run Time(second) | | |
|---|---|---|---|---|---|
| | No. of traversed tile | No. of back-tracing | Device migration | Preprocessing | migration |
| AND3X2 | 609 | 41 | 0.546 | 0.078 | 0.063 |
| AOI21X2 | 1125 | 92 | 0.718 | 0.094 | 0.125 |
| AOI211X2 | 953 | 71 | 0.938 | 0.141 | 0.140 |
| BUFFX20 | 1102 | 21 | 3.765 | 0.25 | 0.250 |
| DFFX2 | 2586 | 106 | 3.406 | 0.328 | 0.641 |
| INVX8 | 375 | 6 | 0.764 | 0.078 | 0.047 |
| INVX12 | 547 | 3 | 1.752 | 0.125 | 0.094 |
| NAND3X4 | 1169 | 62 | 1.359 | 0.156 | 0.204 |
| NAND4X4 | 944 | 17 | 1.75 | 0.203 | 0.187 |
| NOR3X2 | 545 | 23 | 0.656 | 0.109 | 0.078 |
| NOR4X4 | 1816 | 90 | 1.75 | 0.203 | 0.375 |
| OAI21X2 | 561 | 26 | 0.672 | 0.062 | 0.078 |
| OR3X1 | 425 | 20 | 0.516 | 0.047 | 0.047 |
| XNOR2X2 | 737 | 31 | 1 | 0.219 | 0.125 |
| XOR2X2 | 654 | 19 | 1.032 | 0.094 | 0.110 |
| FA1D1 | 7948 | 375 | 7.406 | 0.469 | 3.109 |

Table 1.Migration statistics.

| Calligrapher | | Our work | |
|---|---|---|---|
| Cell Name | Run Time(s) | Cell Name | Run Time(s) |
| andf301 | 21.21 | AND3X2 | 0.687 |
| buff102 | 7.17 | BUFFX1 | 0.343 |
| nanf201 | 4.38 | NAND2X1 | 0.281 |
| norf211 | 12.05 | NOR2X1 | 0.334 |
| oaif2201 | 18.97 | OAI21X2 | 0.812 |
| of401 | 50.58 | OR3X1 | 0.61 |
| xnof201 | 42.27 | XNOR2X1 | 1.125 |
| xorf201 | 37.96 | XOR2X1 | 1.142 |
| adder | 1244.9 | FA1D1 | 10.984 |

Table 2. The comparison of migration results between Calligrapher and ours.

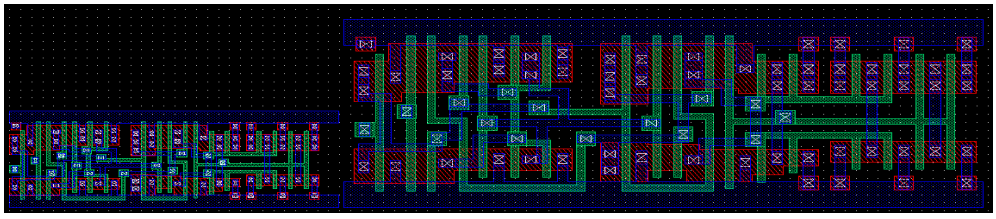

Figure 12. The layouts of cell DFFX2 before and after migration.