Technology Migration Techniques for Simplified Layouts with Restrictive Design Rules

Xiaoping Tang
IBM T.J. Watson Research
Yorktown Heights, NY 10598
xtang@us.ibm.com

Xin Yuan
IBM Corp.
Essex Jct., VT 05452
xinyuan@us.ibm.com

ABSTRACT

Designs using simple geometric layout objects (such as points, sticks and rectangles) with Restrictive Design Rules (RDRs) on each layout object (i.e., it must be placed on a set of grids subject to a set of ground rules) have been introduced as an approach to better enable design for manufacturability (DFM) in ultra-deep submicron designs[9]. In this paper, we study the problem of migrating the conventional shape-based layouts to the simplified layouts with RDR constraints. We present a migration flow which consists of two process steps: (1) conversion where shapes (such as polygons) are converted to simple geometric objects (such as sticks) while the topology is maintained, and (2) grid legalization for RDRs where the simple geometric objects are placed on grid subject to the given set of ground rules by using a novel legalization algorithm, the Minimum Perturbation-driven Graph-based Grid Legalization(MP-GGL) algorithm. We demonstrate the effectiveness of the flow by successfully migrating a set of library cells in the conventional shape-based technology to the simplified layouts with RDR constraints.

1. INTRODUCTION

With the advance of ultra-deep submicron technology, manufacturability has become one of the major problems in VLSI design. Because the ability to control the physical properties of fabricated devices and interconnects is decreasing, the variability of finally printed shapes and their physical properties is increasing. Therefore, design for manufacturability (DFM) has become one of the most challenging topics among designers and researchers. Post-layout manufacturability enhancement techniques, such as optical proximity correction (OPC) and resolution enhancement techniques (RET), have been a key step to compensate shape variations and ensure the manufacturability of designs. However, these post-layout processes are very expensive. The complexity of these techniques is increasing as well. For the emerging technologies (65nm and beyond), the computation cost and complexity of the post-layout processes are becoming the bottle-necks in the design-to-silicon flow.

Therefore, regular layout styles have been proposed to improve the manufacturability and achieve manageable post-layout processing complexity. In [7], Gupta and Kahng predicted that the layouts in the technology of 65nm and beyond will look like a regular grating. The researchers in leading technology companies have proposed to restrict the layouts such that all the gate-forming polysilicon conductors have the same orientation, the same narrow width, and the same pitch[13, 5]. Most recently, Lavin et al[9] have explored how design restrictions could improve the manufacturability and extended the design restrictions on polysilicon conductors to all the layers of a design and proposed to use a set of simple geometric objects, such as points for contacts and vias, sticks for wiring and rectangles for diffusion, to represent the layout. An example of a simplified layout is shown in Figure 1. In addition to the ground rule constraints, each simplified layout object must be placed on a specified grid among a given set of grids (called multiple grid constraints in this paper). The

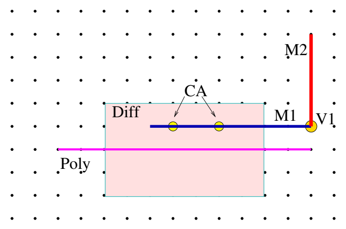

Figure 1: An example of a simplified layout.

down-stream post-layout processes will convert the simple geometric objects to the conventional-look target shapes by considering manufacturing requirements and OPC/RET effects. As technology further scales down, the simplification and restriction is the trend to facilitate post-layout processing and to achieve manufacturable designs and predictable performance. We refer to this kind of design style as simplified layout with RDR constraints in this paper.

Use of simplified layouts with RDR constraints has implications to the design migration process. Automatic design migration is an essential process to achieve maximum productivity. Many of the existing migration tools are based on layout compaction techniques[12, 11, 4] that were developed in the 1980's when layout area and wire length minimizations were the major objectives. Heng et al[8] proposed a new objective for the layout migration, minimum layout perturbation, to preserve the integrity of the original layout as much as possible. They formulated the legalization problem as a special case of linear programming (LP) problem and proposed a graph-based Simplex algorithm to solve it. Later, Zhu et al[16] presented an integer linear programming (ILP)-based approach to preserve the geometric closeness in the layout migration, in the similar manner of minimum layout perturbation. Most recently, Yuan et al[15] addressed the problem of putting polysilicon conductors on grid during the technology migration process. However, none of these works addresses the problems of abstracting layout shapes into simple geometric objects and legalizing them to satisfy the multiple grid constraints.

In this paper, we study the problem of migrating a conventional shape-based layout into a simplified layout subject to a set of ground rules and multiple grid constraints. As a solution, we present a migration flow. It consists of two steps. (1) Conversion: converting shapes (such as polygons) to simple geometric objects, i.e., points, sticks and rectangles. We use rectilinear medial axis to represent the abstract skeleton of conventional shapes (polygons) and transform them into sticks. (2) Legalization: putting objects on grids and satisfying the ground rules (for example, spacing rules). Handling multiple grid constraints is essentially an integer linear programming (ILP) problem. Unfortunately, we observe that in reality the ground rule constraints together with the multiple grid constraints often contain conflicts, and thus a generic ILP solver often fails to return a feasible solution. Therefore we have to look for a practical way to obtain a valid layout (i.e., objects are on grid) while fixing the ground rule violations as many as possible. For this reason, we adopt a graph-based approach to solve the problem. The constraints are represented by a graph (called constraint graph) where the nodes in the graph represent the layout objects and the arcs represent the ground rule constraints. Each node is associated with a grid constraint specified by the technology. Conflicts in the constraints can be expressed by the existence of positive cycles in the constraint graph. We resolve the positive cycles by relaxing the minimum number of arcs (relaxing some ground rules). The legalized result is obtained with the flavor of minimum layout perturbation. We have successfully migrated a set of library cells from the conventional shape-based technology to the legal simplified layouts while satisfying the RDR constraints. We have made the following contributions in developing the migration flow.

(1) Rectilinear medial axis. We propose a new concept, rectilinear medial axis, for converting polygon shapes to sticks, as the existing solutions are not effective. Medial axis (skeleton) has been widely studied in the computational geometry literature. The existing medial axis algorithms may generate arbitrary angle of segments. We restrict the orientation of segments to be either vertical or horizontal and develop an algorithm to create the rectilinear medial axis for rectilinear polygons.

(2) Theoretical bound of iterations in computing the longest path in a directed graph with multiple grid constraints. To compute the longest path in a graph without any grid constraint, Bellman-Ford algorithm may need $n-1$ iterations, and Yen's algorithm may need $n/2$ iterations, where n is the number of nodes in the graph[3]. Lee and Tang[11] presented a theoretical bound of iterations in computing the longest path in a graph with single grid constraints, i.e., only some of the nodes are required to be placed on a single pitch grid, while other nodes are not required to be on grid. We derive a theoretical bound in computing the longest path in a graph with multiple grid constraints, i.e., each node is required to be placed on a specified grid among a set of grids, and to distinguish it from the conventional longest path on a directed graph without grid constraint, we call it the grid longest path in this paper.

(3) Positive cycle removal in computing the grid longest path. For a directed graph without grid constraint, we can use a *prev* arc to record the previous node triggering the value update of the current node during the longest path computation. By traversing the *prev* arcs we can get the nodes on the longest path. If there is no positive cycle, the *prev* arcs and the nodes should form a tree (or forest for an unconnected graph). Thus if we find a cycle when traversing the *prev* arcs, then the cycle is a positive cycle. This approach is no longer valid in finding a positive cycle in a graph with grid constraints. We propose an efficient algorithm to identify the positive cycles and resolve them by relaxing the minimum number of arcs for such graphs.

(4) Minimum Perturbation-driven Graph-based Grid Legalization (MP-GGL) algorithm. Multiple grid legalization with minimum layout perturbation can be formulated as an integer linear programming (ILP) problem. However, in general, ILP solvers not only take very long runtime for large scale problems, but also have difficulty in resolving the conflicts of grid constraints and ground rule constraints. We propose a graph-based algorithm to solve it with minimum layout perturbation flavor. Experimental results have demonstrated the effectiveness of this algorithm.

The rest of the paper is organized as follows. Section 2 defines the problem of migrating conventional layouts into simplified layouts. Section 3 presents the method of converting the shapes to simple geometric objects. The approach to legalize the layouts to satisfy ground rules and multiple grid constraints is described in Section 4. Experimental results are reported in Section 5, followed by the conclusion and future work in Section 6.

2. PROBLEM DEFINITION

A conventional shape-based layout consists of a set of polygons, each of which is associated with a layer including diffusion, polysilicon (poly), metals, contact, vias etc. Usually a layout is restricted to be rectilinear. Simplified layouts with RDR are represented by a set of simple geometric objects, for example, points representing connections between wiring, sticks representing wiring, and rectangles representing diffusion. Each object is required to be on a pitch grid that is some multiple of the baseline grid in the design. For example, some objects must be on the grid of 1X, and some other objects must be on the grid of 4X. In addition to the grid constraints, design ground rules are defined to ensure manufacturability. Typically ground rules include spacing rules specifying the minimum space between objects, length rules specifying the minimum length of some objects, and methodology rules specifying the design requirement for assembling cells (for example, power bus must be placed at some predefined location, the height of the cells is pre-fixed, and so on). We want to preserve the integrity of the original layout in the migration process by using the minimum layout perturbation objective [8]. The perturbation is defined as the absolute difference between the original location and the new location for a layout element. Thus the problem can be defined as follows.

PROBLEM 1. *Given a conventional shape-based layout, we migrate it to a valid simplified layout with minimum layout perturbation subject to a set of restrictive design ground rules and multiple grid constraints.*

3. CONVERSION

Since contacts and vias are square or rectangles, it is straightforward to convert them to points by using their center points to represent the objects. To convert a polygon to rectangles, we can adopt a scan-line algorithm [6] to decompose the polygon to a set of abutted rectangles.

However, it is not trivial to convert polygons (such as polysilicon, metal) to sticks. Existing solutions are not effective. Decomposing the polygon into a set of rectangles using a simple scan-line scheme and then converting each rectangle to a stick (for example, using centerline) may lose the global view of the original polygon and cause mis-oriented or disconnected parts. An example is illustrated in Figure 2.

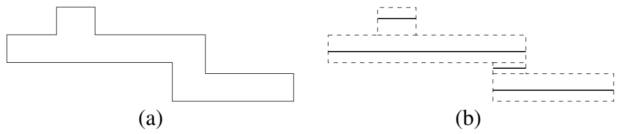

Figure 2: (a) The original polygon. (b) A scan-line algorithm is applied to decompose the polygon into a set of rectangles and each rectangle is converted to a stick using the centerline. The sticks are disconnected.

Instead, we use the medial axis to represent the skeleton of the original polygon, and then convert it to the sticks. Medial axis has been widely studied in the literature of computational geometry [1, 2, 10]. Existing medial axis algorithms may generate non-rectilinear segments for a rectilinear polygon (If Euclidean metric (L_2) is used, medial axis algorithm may generate even parabolic curves[1]). In the simplified layout with RDR, all sticks must be rectilinear. So we propose a method to create the rectilinear medial axis for rectilinear polygons. We first compute the medial axis using L_∞ metric[14]. The result contains only straight segments, some of which may be non-rectilinear. Then we build a graph to represent the connectivity of the segments of the medial axis. Each node in the graph represents a segment. A pair of directed arcs between two nodes represent the connection between the two representing segments. Then we traverse the graph, remove or transform the non-rectilinear nodes/segments into rectilinear ones, while maintaining the connectivity. The conversion result for the polygon in Figure 2 is illustrated in Figure 3.

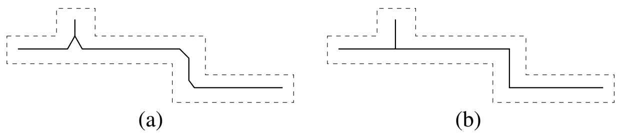

Figure 3: (a) The medial axis result using L_∞ metric. (b) The rectilinear medial axis obtained by our algorithm.

The algorithm consists of six steps shown in Figure 4:

(1) Compute the medial axis of the polygon using metric L_∞.

(2) Build a graph to represent the segment connectivity.

(3) Mark the orientation for each node/segment as one of the followings: horizontal, vertical or non-rectilinear.

(4) Merge non-rectilinear nodes. There may be more than one non-rectilinear nodes between two rectilinear nodes. We loop on each non-rectilinear node, and remove some non-rectilinear nodes such that there exists at most one non-rectilinear node/segment between any two rectilinear nodes/segments.

(5) Mark each node as "kept" or "disposed". All rectilinear (horizontal/vertical) segments will be "kept". Whether a non-rectilinear node/segment is "kept" or "disposed" depends on the situations.

[1]A metric L_n is defined as follows. Given two points, $p_1 = (x_1, y_1)$, and $p_2 = (x_2, y_2)$, the distance between p_1 and p_2, $|p_1 - p_2| = \sqrt[n]{|x_1 - x_2|^n + |y_1 - y_2|^n}$. Euclidean metric is L_2. In L_∞ metric, $|p_1 - p_2| = \max(|x_1 - x_2|, |y_1 - y_2|)$.

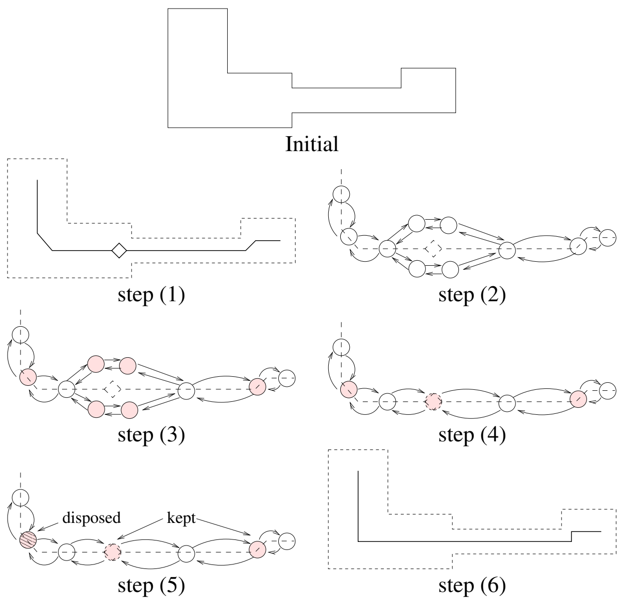

Figure 4: The example of converting a polygon to rectilinear wires by computing rectilinear medial axis. Step (1): compute the medial axis using L_∞. Step (2): construct a graph to represent the connectivity. Step (3): mark node orientation as vertical, horizontal or non-rectilinear. "Shaded" nodes mean non-rectilinear. Step (4): merge non-rectilinear nodes. Step (5): determine to "keep" or "dispose" a node by examining the pattern. Step (6): output rectilinear medial axis and wires.

Since we want to keep as few bends as possible, a non-rectilinear node in the graph is marked "kept" only when the corresponding segment is definitely needed to maintain the connectivity. Otherwise, it will be marked as "disposed". The patterns are illustrated as follows. We check the connected nodes/segments at each end of a non-rectilinear node/segment:

1. If the two ends connect the segments which have different orientations (vertical or horizontal), then the node is "disposed" as we can extend the two connected segments (with different orientations) to maintain the connectivity with fewer bends. The patterns are shown in Figure 5(a).
2. If the two ends only connect the segments with the same orientation, i.e,. two parallel segments, then the node is "kept" as it is needed to connect those two parallel segments. We extend the parallel segments to the middle point and transform the non-rectilinear segment to a rectilinear one to maintain the connectivity. The patterns are shown in Figure 5(b).

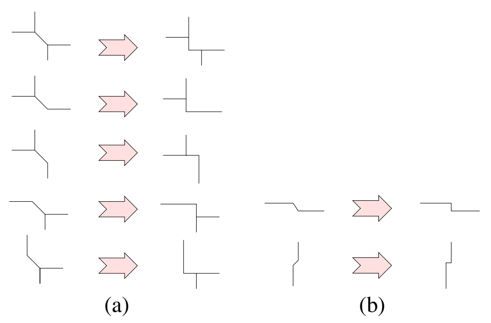

Figure 5: (a) The patterns to mark a non-rectilinear segment (node) as "disposed". (b) The patterns to mark a non-rectilinear segment (node) as "kept".

(6) Output the rectilinear medial axis and wires. We only output the nodes/segments marked as "kept". If a 'kept' node is a non-rectilinear segment, we transform it to a rectilinear segment based on the orientation of the connected nodes/segments, as shown in Figure 5(b). A segment may be degenerate (0 length), and can be ignored. If a 'kept' node is a rectilinear segment, we may need to extend its ends to abut with the adjacent nodes (segments) to maintain the connectivity. We can merge the connected segments with the same orientation and the same width, and remove the redundant ones.

4. LEGALIZATION

After converting shapes into simple geometric objects (points, sticks and rectangles), we need to legalize the simplified layout to satisfy the given set of restrictive design ground rules and grid constraints. Typically, legalization is performed in two successive steps, first in the x-direction and then in the y-direction, or vice versa. The successive 1-D legalizations can meet most of the ground rule constraints in a realistic layout migration environment and are capable of producing good results in practice with much less runtime compared with the 2-D legalization. In this paper, we adopt the approach of two successive 1-D legalizations. Without loss of generality, in the following we only describe the legalization in the x-direction.

A constraint graph , $G = (V,A)$, is used to represent the ground rule constraints. In the x-direction, each node v_i in the graph represents one of the following layout elements: a vertical edge of a rectangle, an endpoint of a horizontal stick, a vertical stick, and a point. Without confusion, we also use v_i to denote the layout element in the paper. Let $x(v_i)$ denote the x-location of layout element v_i, and $x^{old}(v_i)$ denote the initial x-location of a layout element v_i in the given layout. The constraint specified by a ground rule between two layout elements v_i and v_j is represented by a difference constraint of the form $x(v_j) - x(v_i) \geq w_{ij}$ (the equality constraint can be expressed by two difference constraints). The constraint corresponds to a directed arc, $a_{ij} = (v_i, v_j)$, from node v_i to node v_j with weight w_{ij} in the constraint graph, where v_i is called *arc tail* and v_j is called *arc head*. In addition, each node v_i is associated with a grid constraint: being placed on grid of g_iX. The grid constraint can be expressed as: $x(v_i) = g_i \times x'(v_i)$, where $x'(v_i)$ is an integer. For example, for a device with double contacts as shown in Figure 6(a), we can use two nodes d_1 and d_2 to represent the left and right edge of diffusion, two nodes p_1 and p_2 to represent the left and right endpoint of polysilicon, and two nodes c_1 and c_2 to represent the two contacts, respectively. The constraint graph is shown in Figure 6(b). Typically, the device size will not be changed in the legalization process, i.e., fixing the value of $x(d_2) - x(d_1)$ to be the original. Let s_0 be the original value, and then $x(d_2) - x(d_1) = s_0$. Thus we add two more arcs, (d_1, d_2) with weight s_0 and (d_2, d_1) with weight $-s_0$, resulting in the constraint graph as shown in Figure 6(c).

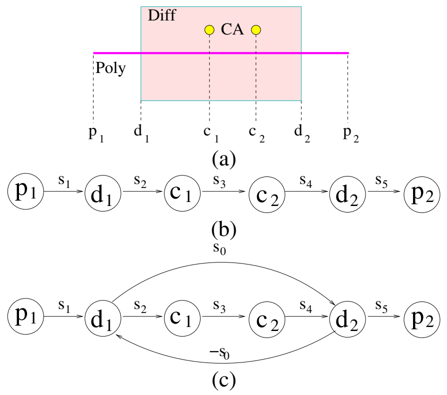

Figure 6: (a) A device with double contacts. (b) The corresponding constraint graph. (c) The constraint graph with fixed device size ($x(d_2) - x(d_1) = s_0$).

The legalization problem can be formulated as follows:

$$\min \sum_{v_i \in V} |g_i \times x'(v_i) - x^{old}(v_i)| \tag{1}$$

subject to:

$$g_j \times x'(v_j) - g_i \times x'(v_i) \geq w_{ij}, \qquad \forall a_{ij} \in A$$
$$x'(v_i) \text{ be integer}, \qquad \forall v_i \in V$$

We can linearize the objective function by using the technique presented in [8]. Thus the problem (1) is essentially an integer linear programming (ILP). In addition to the fact that it is expensive to solve an ILP problem, unfortunately, in reality the inputs of restrictive design ground rules with the grid constraints often contain conflicts and thus a generic ILP solver fails to return a feasible solution. Conflicts of the constraints can be identified as positive cycles in the constraint graph. For example, in the constraint graph in Figure 6(c) where we fix the device size, if the separation space $s_2+s_3+s_4 > s_0$, then there is a conflict in constraints and $d_1 \rightarrow c_1 \rightarrow c_2 \rightarrow d_2 \rightarrow d_1$ forms a positive cycle. In fact as we discuss later, with the grid constraints taken into account, the condition that the sum of the weights along a cycle is greater than 0 is only sufficient but not necessary to imply a positive cycle.

Therefore we propose a Minimum Perturbation-driven Graph-based Grid Legalization(MP-GGL) algorithm for multiple grid legalization problem. Our algorithm can produce a valid simplified layout with RDR (i.e., all the objects will be on grid) while trying to fix the ground rule violations and preserve the integrity of the original layout as much as possible. For the case shown above, the algorithm will relax some constraint (arc). Designers can manually fix the ground rule violations later (for example, to remove one of the double contacts or to change the device size in this case). MP-GGL algorithm is based on the grid longest path computation. In the following sections, we first present how to compute the grid longest path with positive cycle removal, and then describe the MP-GGL algorithm.

4.1 Computing the Grid Longest Path

Computing the longest path in a directed graph without the grid constraints $G=(V,A)$, the inverse version of computing the shortest path, is a classical problem. Bellman-Ford algorithm[3] can handle arbitrary weights of arcs, and may take $n-1$ iterations (n is the number of nodes). In each iteration, all the arcs are labeled by updating the value of the *arc head* based on the arc weight and the value of the *arc tail*. Yen's algorithm[3] improves Bellman-Ford algorithm by assigning an order to the nodes, for example, v_1, v_2, ..., v_n. So the arcs are partitioned into two disjoint sets, forward arcs A_f and backward arcs A_b, where $A_f=\{a_{ij}|i<j\}$ and $A_b=\{a_{ij}|i>j\}$. Both (V,A_f) and (V,A_b) are directed acyclic graphs (DAG). At each iteration, each node is visited in the order of v_1,v_2,...,v_n, and its connected arcs in A_f are labeled (referred as *forward pass* in the paper), and then each node is visited in the reverse order, and its connected arcs in A_b are labeled (referred as *backward pass*). In this way, it only takes at most $n/2$ iterations for the labeling process to converge if there is no positive cycle in the graph. Observing that the constraint graph derived from a layout has some special property, Liao and Wong[12] proposed to sort the nodes according to their original locations, and assign the order to the nodes. In this way, $|A_b| << |A_f|$. Thus they proved that the number of iterations needed is at most $|A_b|+1$. Typically in a constraint graph of a layout, $|A_b|+1 << n/2$. If the number of iterations exceeds the bound and the labeling operation still updates the value of some node, then there exists a positive cycle. Basically, if there is no positive cycle, the longest path is well-defined, and all the longest paths from the source to all other nodes form a tree rooted at the source.

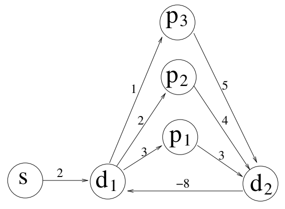

Figure 7: A graph with grid constraints. s is the source; d_1 and d_2 are on the grid of 1X; and p_1, p_2 and p_3 are on the grid of 4X. From s to d_1, the grid longest path is $s(0) \rightarrow d_1(2) \rightarrow p_1(8) \rightarrow d_2(11) \rightarrow d_1(3) \rightarrow p_2(8) \rightarrow d_2(12) \rightarrow d_1(4) \rightarrow p_3(8) \rightarrow d_2(13) \rightarrow d_1(5)$, where the node value is shown in () adjacent to each node. Note that d_1 appears multiple times in the grid longest path due to the rounding-up of labeling p_1, p_2, p_3 for their grid constraints. On the other hand, the value change on d_1 does not affect p_1 due to the fact that p_1 is on a coarser grid and thus its labeling process can "absorb" the value increase from d_1.

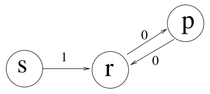

Figure 8: A graph with grid constraints. s is the source; r is on the grid of 3X; and p is on the grid of 4X. The pair of arcs between r and p implies that the value of r must equal the value of p. From s to r, the grid longest path is $s(0) \rightarrow r(3) \rightarrow p(4) \rightarrow r(6) \rightarrow p(8) \rightarrow r(9) \rightarrow p(12) \rightarrow r(12)$, where the node value is shown in () adjacent to each node.

With grid constraints taken into account, we can still compute the longest path by iteratively labeling the arcs. In each labeling operation, we need to round the value of the *arc head* up to the next grid location. We illustrate it by two examples, shown in Figure 7 and 8. The following lemma states the path property with grid constraints.

LEMMA 1. *For a directed graph $G=(V,A)$ with l different grids, g_i, $i=1,2,...,l$, given a path from v_1 to v_2 that the initial value of v_1 is x_1 and the value of v_2 is x_2 after labeling the path, if the value of v_1 is assigned to x_1' such that $x_1' \geq x_1$, then the value of v_2 will be at least x_2 after re-labeling the path; and if the value of v_1 is increased to g_M+x_1, then the value of v_2 will be g_M+x_2 after re-labeling, where g_M is the least common multiple of all g_i, $i=1,2,...,l$.*

Proof: Obvious. □

It can be seen that due to the grid constraint the grid longest path is no longer a tree and a node may appear several times in the grid longest path. However, in the grid longest path in Figure 7, the node, d_1 or d_2, appears at most 4 times; and in the grid longest path in Figure 8, the node p appears no more than 3 times. The result is summarized in the lemma as follows.

LEMMA 2. *For a directed graph with l different grids, g_i, $i=1,2,...,l$, if there is no positive cycle, a node v, which is on the grid of g_k, will appear in a grid longest path at most g_M/g_k times, where g_M is the least common multiple of all g_i, $i=1,2,...,l$.*

Proof: Let us suppose that v appears t times in a grid longest path, $t > g_M/g_k$. Obviously, g_M/g_k is an integer since g_M is the multiple of g_k. Thus $t-1 \geq g_M/g_k$. We denote the grid longest path to be $s \rightarrow ... \rightarrow v \rightarrow ... \rightarrow v \rightarrow ... \rightarrow v \rightarrow ...$. Thus the sequence of nodes between two consecutive v form a cycle (beginning with v and ending with v). Since v is on the grid of g_k, the labeling process will update the value of v to be the multiple of g_k. When the labeling process visits v for a second time along the grid longest path, it should increase the value of v at least g_k, otherwise the change of value is ≤ 0 and the cycle of nodes shall not appear on the longest path. If v appears t times, $t-1 \geq g_M/g_k$, then the value increase from the first v to the last v along the path will be $\geq (t-1)g_k \geq g_M$. Note that the path from the first v to the last v forms a cycle. According to Lemma 1, the labeling process will keep increasing the value of v at least g_M each time by visiting the sequence of nodes from the first v to the last v. Thus the labeling process will not converge, i.e., the sequence of nodes from the first v to the last v will form a positive cycle, which contradicts the assumption. □

When there are grid constraints, we may need more iterations in labeling the arcs. Lee and Tang[11] presented a theoretical bound of iterations in computing the longest path in a graph with single grid constraints, i.e., only some of the nodes are required to be placed on a single pitch grid, while other nodes are not required to be on grid. It is not trivial to generalize the result to the multiple grid constraints. The following theorem states the theoretical bound in computing the longest path in a graph with multiple grid constraints.

THEOREM 1. *For a graph with l different grids, g_i, $i=1,2,...,l$, supposing that there are n_i graph nodes on the grid of g_i, and m_i be the number of backward arcs in A_b such that the max grid of the two nodes it connects is g_i, if there is no positive cycle, then the number of labeling iterations is at most*

$$\min\left(\left(\sum_{i=1}^{l}(n_i g_M/g_i)\right)/2, \sum_{i=1}^{l}(m_i g_M/g_i)+1\right)$$

where g_M is the least common multiple of all g_i, $i=1,2,...,l$.

Proof: According to Lemma 2, a node v which is on the grid of g_i appears at most g_M/g_i times in a longest path. Thus the number of nodes in a longest path (including repeats) is at most $\sum_{i=1}^{l}(n_i g_M/g_i)$. Each iteration labels at least 2 arcs along any longest path (at least one in the forward pass and at least one in the backward pass). Thus the number of iterations needed is at most $(\sum_{i=1}^{l}(n_i g_M/g_i))/2$.

In each iteration, the forward pass labels some forward arcs in A_f, and the backward pass labels at least one backward arc in A_b (if A_b is not empty) along any longest path. Since a node v which is on the grid of g_i appears at most g_M/g_i times in a longest path, the arc which connects v will also appear at most g_M/g_i times. We are given that m_i be the number of backward arcs in A_b such that the max grid of the two nodes it connects is g_i. Thus $|A_b| = \sum_{i=1}^{l} m_i$ and the number of backward arcs in a longest path (including repeats) is at most $\sum_{i=1}^{l}(m_i g_M/g_i)$. Since each iteration labels at least one backward arc, the number of iterations is at most $\sum_{i=1}^{l}(m_i g_M/g_i)+1$ (one additional iteration may be needed if the last arc in a longest path is a forward arc).

Therefore, in summary the bound of iterations is
$\min((\sum_{i=1}^{l}(n_i g_M/g_i))/2, \sum_{i=1}^{l}(m_i g_M/g_i)+1)$. □

COROLLARY 1. *The bound of iterations in computing the grid longest path:* $\min((\sum_{i=1}^{l}(n_i g_M/g_i))/2, \sum_{i=1}^{l}(m_i g_M/g_i)+1)$ *is tight.*

The bound of iterations can be reached in the worst case. An example is shown in Figure 9. For the graph, we need $n+1$ iterations to compute the grid longest path from s to d_3, and $n+1 = \min((\sum_{i=1}^{l}(n_i g_M/g_i))/2, \sum_{i=1}^{l}(m_i g_M/g_i)+1)$. Note that only 2 iterations are needed if there are no grid constraints.

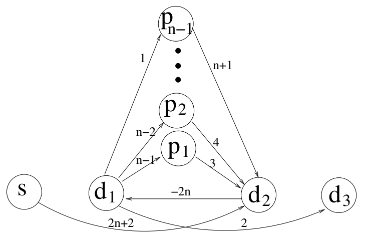

Figure 9: A graph with grid constraints. s is the source; d_i, $i = 1,2,3$, are on the grid of 1X; and p_i, $i = 1,...,n-1$, are on the grid of nX. From s to d_3, the grid longest path is $s \to d_2 \to d_1 \to p_1 \to d_2 \to d_1 \to ... \to p_i \to ... \to d_2 \to d_1 \to p_{n-1} \to d_2 \to d_1 \to d_3$, and the final value of d_3 is $n+3$. The node order is $s, d_1, p_1, ..., p_{n-1}, d_2, d_3$. The number of iterations required to compute the grid longest path is $n+1$.

4.2 Positive Cycle Removal

When the number of iterations exceeds the bound and the value of some node is still changing in the labeling process, there exists a positive cycle. In a directed graph without the grid constraints, we can use a *prev* arc to record the previous node during the longest path computation which triggers the value update of the current node. Thus we can traverse the *prev* arcs back to eventually find the positive cycle if there exists one, and then remove it by relaxing some arc. When grid constraints are taken into account, a node/arc may appear multiple times even within one longest path. The way of using a *prev* arc to record the previous node does not work any more. How to remove positive cycle for a directed graph with grid constraints is not considered in [11]. It should be noted that the sum of the arc weights along a positive cycle may be less than or equal to zero on a graph with grid constraints. Such an example is shown in Figure 10. We propose an efficient algorithm to remove positive cycles on such graphs.

When the number of iterations exceeds the bound, we mark the arcs triggering the value update of some nodes as "potential bad" arcs, and put them into an array. Note that not all the potential bad arcs are "true" bad arcs. Then we use binary search to find the true bad arcs (maybe more than one). The arcs in the second half of the array are marked as "bad". Then we re-compute the grid longest path. Those arcs marked as "bad" will be ignored in the labeling operation. We take one of the following actions based on the result.

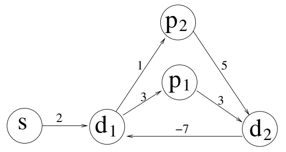

Figure 10: A graph with a positive cycle. s is the source; d_1 and d_2 are on the grid of 1X; and p_1 and p_2 are on the grid of 4X. The positive cycle is $d_1 \to p_1 \to d_2 \to d_1 \to p_2 \to d_2 \to d_1$. The labeling process along the cycle will not converge. Note that the sum of the weights along the cycle is less than 0.

1. If the computation converges, then the arcs in the first half of the array are "good" arcs and the size of potential bad arcs is reduced by half. We continue the search process on the second half of the array, recursively.
2. If the computation does not converge within the bound of iterations, then there exists at least one "true bad" arc in the first half of the array. We recursively search for true bad arcs on the first half of the array until one is found.

In this way, we can identify one bad arc. It should be noted that we need to continue the search process on the rest of "potential bad" arcs, since there may exist some other "true bad" arcs. Obviously, the number of iterations to identify the 'true bad" arcs is bounded by $b \log p$, where b is the number of "true bad" arcs and p is the number of potential arcs we have identified.

It should be noted that two potential arcs may be mutually exclusive in terms of "true bad", i.e., one is "true bad" and then the other is "good", if they belong to the same positive cycle. Our approach will identify exactly one of them as "true bad". The one that is placed in the later position in the array (i.e., with bigger index) will be identified as "true bad". Therefore, we can assign some priority to the potential arcs, and sort them in the array.

4.3 MP-GGL Algorithm

We first construct the constraint graph for a layout and add two additional nodes: a source node for the left boundary and a sink node for the right boundary. We compute the grid longest path beginning with the source, i.e., compacting all layout objects to the left boundary subject to the given set of ground rule constraints and multiple grid constraints. Thus the value of a node is the lower-bound of valid on-grid locations to place the corresponding object in the layout. Then we invert the directions and weight signs of all the arcs in the constraint graph and compute the grid shortest path beginning with the sink, i.e., compacting all layout objects to the right boundary subject to the given constraints. Note that the expansion of the layout area may be needed in order to contain all the objects. In this way, we obtain the upper bound of valid on-grid locations to place each node. Then the nodes are sorted in the order regarding to the original locations. We visit each node in this order, place the node on grid position between its lower bound and upper bound honoring its original location, and then propagate the value to other nodes by updating their lower bounds or upper bounds if needed. The algorithm of legalization is summarized as follows.

```
Algorithm MP-GGL
1.  Constructing the constraint graph $G(V,A)$
2.  Obtaining the lower bounds by computing the grid
        longest path beginning with source
3.  Obtaining the upper bounds by inverting the directions
        and weight signs of arcs and computing the grid
        shortest path beginning with sink
4.  FOR EACH node in the order regarding to original loc
5.      Placing it between lower bound and upper bound
            honoring the original location
6.      Propagating the update of other bounds
```

5. EXPERIMENTAL RESULTS

We implemented the proposed approaches, conversion and legalization, in C++ and integrated them to a migration flow and tested it on a 1.6GHz Linux box with 1024MB memory by migrating a set of library cells in the conventional shape-based technology to the simplified layout designs with RDR constraints. Figure 11 shows a layout in the conventional shape-based technology and its simplified representation after being converted and legalized by our flow.

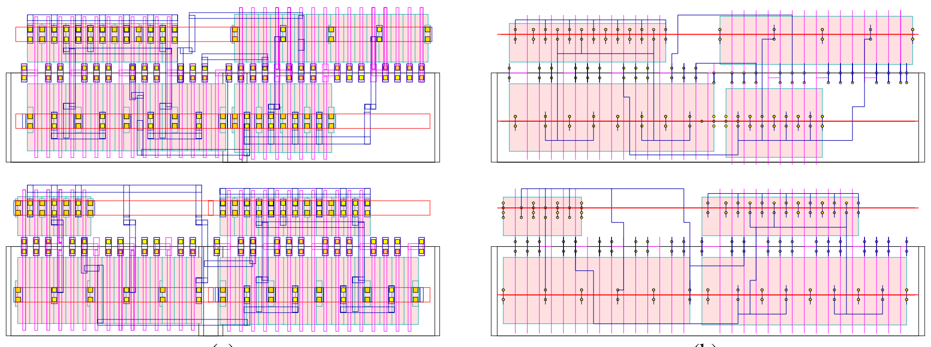
(a) (b)

Figure 11: (a) The original layout. (b) The simplified layout with RDR.

Table 1: Comparison on legalization between MP-GGL and ***LP+Snap***

test case	#nodes	#arcs	ground rule errors		runtime(s)	
			LP+Snap	MP-GGL	LP+Snap	MP-GGL
cell1	2799	19742	138	0	2.12	1.17
cell2	2718	19156	110	0	1.94	1.14
cell3	2049	14237	80	0	1.31	0.83
cell4	1286	8575	80	0	0.92	0.62

Table 2: Experimental Results on Positive Cycle Removal

test case	#nodes	#arcs	#potential bad arcs	#removed bad arcs	runtime (s)
cell5	1883	9382	544	52	3.03
cell6	573	3769	180	19	0.61
cell7	587	3629	54	25	0.52
cell8	455	3078	46	16	0.43

In order to show the efficiency of the MP-GGL algorithm presented in Section 4, we compare the legalized results provided by MP-GGL algorithm with a two-step flow where the layout is legalized without the grid constraints using a minimum perturbation-based legalization engine[8], and then we snap the object to the nearest grid point without considering any ground rule constraints. This two-step flow is referred as *LP+Snap* in Table 1. Table 1 reports the results of legalizing four library cells. The inputs are the simplified layouts generated by the conversion process in our migration flow and have grid violations (not all of the objects are on grid) and ground rule violations. The MP-GGL algorithm can generate legal results (the objects are placed on grid and meet the ground rule constraints) when there is no positive cycle in the constraint graph. For those four cells, we only apply spacing and length constraints, thus there is no positive cycle and there are zero ground rule errors in the legalized results. On the other hand, the *LP+Snap* flow can not fix all the ground rule violations due to the heuristic snap-on-grid process. The size of the layout is reported in terms of the number of nodes and arcs in the constraint graph.

In order to test the efficiency of our positive cycle removal technique in MP-GGL, we tested it on another set of testcases with not only spacing and length constraints but also methodology constraints which cause positive cycles in the constraint graph. Because of the relaxation in removing the positive cycles, there may be some ground rule violations. The results are shown in Table 2.

6. CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of migrating the conventional shape-based layouts to the simplified layouts to meet the restrictive design rules with multiple grid constraints. We presented a migration flow which consists of two steps: conversion (converting the shape-based layout to the simplified layout using rectilinear medial axis) and legalization (modifying the layout to satisfy the set of ground rules and multiple grid constraints). We formulated the multiple grid legalization problem as an integer linear programming problem and proposed the MP-GGL algorithm, which is based on computing the grid longest path with positive cycle removal, to solve the legalization problem. Experimental results on migrating library cells from the conventional shape-based layouts to the simplified layouts with RDR have demonstrated the effectiveness of our approach. Future work includes the study on design migration for hierarchical layouts from the conventional shape-based technology to the simplified layout style designs with RDR.

Acknowledgments

We would like to thank the people in IBM for their collaborations: Kevin W. McCullen, Fook-Luen Heng, Robert F. Walker, Jason Hibbeler, Michael S. Gray, Rani R. Narayan, Sharon K. Wu, Matthew T. Guzowski, Cam V. Endicott, and Sriram V. Peri.

7. REFERENCES

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Comput. Surveys*, 23(3):345–405, September 1991.

[2] J. W. Brandt. Convergence and continuity criteria for discrete approximations of the continuous planar skeleton. *CVGIP: Image Understanding*, 59(1):116–124, January 1994.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[4] S. K. Dong, P. Pan, C. Y. Lo, and C. L. Liu. Constraint relaxation in graph-based compaction. In *Proc. ACM/SIGDA Physical Design Workshop*, pages 256–261, 1996.

[5] P. Gelsinger. Gigascale integration for teraops performance – challenges, opportunities, and new frontiers. Design Automation Conference 2004, Keynote speach.

[6] A. Gupta. ACE: a circuit extractor. In *Proc. Design Automation Conf*, pages 721–725, 1983.

[7] P. Gupta and A. B. Kahng. Manufacturing-aware physical design. In *Proc. Int. Conf. on Computer Aided Design*, pages 681–687, 2003.

[8] F.-L. Heng, Z. Chen, and G. E. Tellez. A VLSI artwork legalization technique based on a new criterion of minimum layout perturbation. In *Proc. Int. Symp. on Physical Design*, pages 116–121, 1997.

[9] M. Lavin, F.-L. Heng, and G. Northrop. Backend CAD flows for "restrictive design rules ". In *Proc. Int. Conf. on Computer Aided Design*, pages 739–746, 2004.

[10] D. T. Lee. Medial axis transformation of a planar shape. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 4(4):363–369, 1982.

[11] J.-F. Lee and D. T. Tang. VLSI layout compaction with grid and mixed constraints. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(5):903–910, September 1987.

[12] Y.-Z. Liao and C. K. Wong. An algorithm to compact a VLSI symbolic layout with mixed constraints. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2(2):62–69, April 1983.

[13] L. W. Liebmann, A. Barish, Z. Baum, H. Bonges, S. Bukofsky, C. Fouseca, S. Halle, G. Northrop, S. Runyon, and L. Sigal. High-performance circuit design for the RET-enabled 65nm technology node. In *Proc. of SPIE Design and Process Integration for Microelectronics Manufacturing II*, volume 5379, pages 20–29, 2004.

[14] E. Papadopoulou and D. T. Lee. The L_∞-voronoi diagram of segments and VLSI applications. *Int. J. Comput. Geometry Appl.*, 11(5):503–528, 2001.

[15] X. Yuan, K. W. McCullen, F.-L. Heng, R. F. Walker, J. Hibbeler, R. J. Allen, and R. R. Narayan. Technology migration technique for designs with strong RET-driven layout restrictions. In *Proc. Int. Symp. on Physical Design*, pages 175–182, 2005.

[16] J. Zhu, F. Fang, and Q. Tang. Calligrapher: a new layout-migration engine for hard intellectual property libraries. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(9):1347–1361, September 2005.