Calligrapher: A New Layout Migration Engine for Hard Intellectual Property Libraries

Jianwen Zhu, *Member, IEEE*, Fang Fang and Qianying Tang

***Abstract*— Modern system-on-chips depend heavily on hard intellectual properties (IP), such as standard cell and datapath libraries. As the foundries accelerate their update of advanced processes with increasingly complex design rules, and the libraries grow in flexibility and size, the cost of library development becomes prohibitively high. Automated layout migration techniques used today, which are based on layout compaction developed a decade ago, corrupt advanced design considerations by honoring only design rules, and cannot cope with some of the new challenges involved. In this paper, we present a new, integer linear programming (ILP) based layout migration engine, called calligrapher, and make the following contributions: First, we extend the recently proposed minimum perturbation metric (MP) designed to retain original layout design intentions, while overcoming its shortcoming of biased treatment of layout objects. Second, we propose a new design rule constraint algorithm, and prove its linear complexity for the number of constraints generated. Compared with what has been achieved in the literature, the proposed algorithm can significantly reduce the ILP solver time by limiting the constraint size. Third, we propose an iterative migration framework based on the concept of soft constraint. With this framework, two-dimensional compaction quality can be achieved with a runtime comparable to one-dimensional compaction. We demonstrate the effectiveness of calligrapher by migrating the Berkeley low power libraries, originally developed for 1.2um MOSIS process, into TSMC 0.25um and 0.18um technologies. We show that even for very compact layout, our metric and MP metric can make a difference by as much as 20-45%. We also show that our iterative algorithm can improve area by 10% on average compared to traditional technique using MP metric, and inflates area by merely 7.5% comparing to traditional technique using minimum area metric.**

***Index Terms*— Design Reuse, Layout Migration, Layout Compaction**

I. Introduction

It is generally felt that the complexity involved in systems-on-chip (SOC) design can only be tamed by intellectual-property (IP) based design, where the reuse of existing components is advocated to boost design productivity. Two forms of IP are generally used: *soft IPs*, delivered in the form of synthesizable RTL, and *hard IPs*, delivered in the form of layout. Since soft IPs will be re-synthesized by the SOC integrator for their chosen technology, they are favored for their portability. On the other hand, soft IPs are only limited to digital logic and SOC is by no means only a sea of gates. In fact, 70% of the silicon area will be occupied by hard IPs such as SRAMs, FIFOs, CAMs, datapaths and analog front-ends, which are typically designed by abutting a *library* of hand crafted cells. With custom layout design, hard IPs can be delivered with higher performance and better predictability than the soft IPs, and therefore less effort in the SOC integration. Furthermore, soft IPs eventually have to be implemented by standard cells, which themselves are hard IPs.

The major bottleneck that prevents the wide adoption of hard IPs is, as its name suggests, the dependency of layout on process. The cost of initial custom layout design is already very high. This applies even to IPs such as standard cells, which were considered simple compared to other IP libraries. Today's standard cell libraries contain hundreds of cells with different functions and driving strengths. Most fabless companies choose to use libraries offered by hard IP companies precisely because of the high cost associated with library development. To make things worse, manufacturing processes are updated every 18 month, each with a different set of design rules. This makes the development cost of hard IPs too high even for hard IP vendors, since they have to offer different versions for different foundries as well. Automatic layout migration technology, which can amortize the high development cost associated with custom design across different foundries and processes, is therefore crucial for the sustained growth of hard IPs for the small, and the viability of IP-based design for the large.

Unfortunately, layout migration techniques reported in the past cannot cope with all the challenges involved. First, all migration tools are based on layout compaction, a technology developed a decade ago, when layout area is the primary concern. In modern designs that use aggressive circuit styles in deep submicron processes, space is often among the first class citizens of *advanced layout considerations*, for example, to combat signal integrity. It is instructive to see examples by Heng et. al. [1] (page 2) on which layout compaction, the technology where most commercial migration tools are based, fails. Second, all techniques reported in the literature are designed to migrate a specific circuit that uses a library of cells, rather than the library itself. Without considering the *overall library architecture* such as power/ground net width, routing track number and port matching, the cell layouts migrated under this circuit-driven strategy work only for the specific circuit, and there is no guarantee they work under all occasions. As such, a re-emerging interest for new migration algorithms in order to address the need of new metrics [1], the adherence to library layout architecture [2]. Other important considerations include the special treatment of analog IPs [3], and the new issues for complex design rules and resolution

Manuscript received July 29th, 2004. This work was supported by NSERC and University of Toronto.

J. Zhu, F. Fang and Q. Tang are with the Department of Electrical and Computer Engineering, University of Toronto, Ontario M5S 3G4, Canada

enhancement technologies [4].

Most traditional migration engines perform two successive layout compaction, first along the X direction, followed by the Y direction. This strategy is referred to as one-dimensional (1-D) compaction. In each compaction, the constraints and objective functions can be exclusively formulated as difference inequalities. Fast shortest-path algorithms, such as Bellman-Ford, can therefore be used. This strategy is usually favored over two-dimensional (2-D) compactors for its speed, but is known to produce inferior result. Given the additional disadvantage that such engines are not sufficient to handle new, non-area-centric metrics and new architectural constraints, the core of the migration algorithm, including constraint generation, objective function and migration strategy, needs to be revisited. In this paper, we report several key techniques we employ in a general, integer linear programming (ILP) based layout migration engine, called **calligrapher**, designed specifically for hard library IPs, including standard cell and datapath libraries. We make the following contributions.

- **geometric closeness metric**: Inspired by the minimum perturbation metric by Heng et. al. [1], we introduce a new optimization objective, called *geometric closeness*, to reward geometric resemblance of migrated layout to the original layout. Under this metric, space is explicitly represented. Preservation of space and non-space polygons are given equal priority. This ensures that the circuit and layout level considerations of the original design are not corrupted. The proposed metric is also an improvement of [1], which as we elaborate later, bias left (bottom) polygons over right (top) polygons.
- **linear design rule constraint generation algorithm**: Since the ILP runtime is sensitive to the constraint size, and the constraints in the migration problem are dominated by design rule constraints, we propose a new constraint generation algorithm, and show that the number of constraints generated is of $O(n)$, where n is the number of *tiles* in a corner-stitched layout. In contrast, the best constraint generation algorithm among numerous attempts reported in the literature achieved is of $O(nlogn)$, and this complexity is only observed experimentally, rather than proven theoretically.
- **soft constraint satisfaction**: We propose a new technique such that the 2-D compaction quality can be achieved by iterative 1-D compaction. The key idea is to “soften” those constraints that are artifacts of 1-D compaction – instead of hard constraints that must be satisfied, they can be violated. During the iteration, these constraints are gradually tightened, eventually leading to a feasible solution. Our experience indicates that for those hard instances we encounter with the traditional 1-D compaction strategy, the proposed method converge in only a few iterations.
- **library architecture constraint satisfaction**: We show how architectural constraints can be generated to satisfy the specific needs of library IPs, such as port matching, power/ground size and routing track matching. Unlike hierarchical compactors which consider many cells at the same time, we employ a *dual-pass* strategy such that each cell can be compacted separately, thereby significantly reducing the runtime.

The rest of the paper is organized as follows. We first give the problem formulation in Section III. We then briefly describe our migration framework and discuss in detail the rationale of geometric closeness metric in Section IV. We describe our design rule constraint generation algorithm and give complexity analysis in Section V. In Section VI, we describe the concept of soft constraint satisfaction and show how the usual problem of 1-D compaction can be avoided. We describe issues related to architectural constraints for standard cell and datapath libraries in Section VII. Detailed experimental results is given in Section IX before we draw conclusions.

II. Related Work

Automatic layout migration was among the oldest CAD problems investigated and a large body of research was carried out under the layout compaction problem. Surveys of layout compaction can be found in [5] and [6]. Early compactors are performed on symbolic layout in which circuit elements are presented by simple lines or rectangles, known as *sticks*. Modern compactors, including ours, operate directly on mask layout.

A. Virtual Grid Compaction and Shear-line Compaction

The virtual grid method used by MULGA [7] views the geometric plane as a $m \times n$ matrix, where each entry of the matrix is marked either by movable objects or blank space. The compaction is achieved by repeatedly identifying a path of blank cells, and remove it until no such path can be found. The shear-line approach is very similar to virtual grid compaction except that the grid spacing is fixed to the worst case design rule [8], [9].

One problem with this approach is the difficulty in choosing the grid size. A coarse grid results in smaller matrix and thus less run time, but the compaction may not be effective. On the other hand, high resolution may lead to large matrix and thus slow down the run time. Another disadvantage of this method is that the optimization goal is limited only to layout area.

B. Constraint Graph Compaction

The constraint graph compaction was first discussed in CABBAGE [10] and FLOSS [11], which uses graph to capture the design rule constraints. It remains the most popular approach thanks to its flexibility.

The key part of constraint graph compaction is design rule constraint generation, which is used to identify the spacing relationship among all the elements according to design rules and layout topology. The constraint generation process is very similar to design rule checking except that the circuit elements outside of checking region need to be considered as well because they may be pushed into the checking region after compaction. One of the most often used methods to generate constraints is called “shadow-propagation” method used in

CABBAGE [10]. This approach trims the searching area by shining an imaginary light from the element being checked. Based on the assumption that the relative positions between two elements will not be changed, meaning elements originally outside the shadow will not move into the shadow, only the constraints between the given element and the elements that the shadow first meet are generated. The worst case complexity for shadow-propagation method is proven to be $O(n^{1.5})$, where n is the total number of elements in the symbolic layout.

Another approach to generate constraints come from a design rule checking method called *scan line approach* [12]. The circuit elements that stay too far from the checking element are filtered out by a scanning bar of width D, which is the worst-case design rule distance. The experimental results indicates that with n elements in the layout, the expected time complexity required is $O(nlogn)$ [12].

After the constraint graph is built, there are two ways to solve the graph. One is to decide each vertex's longest path length from the boundary vertex, which amounts to finding the shortest path in a directed graph with edge weight negated. Most of early compaction tools are based on this approach. However, the implied compaction goal of this approach is to minimize the layout area.

The *minimum perturbation objective function* proposed in [1] was the first work that departed from the traditional area minimization optimization goal and argued the importance of rewarding geometric similarity between the migrated layout and the original layout. However, the quantitative measure that they develop for geometric similarity is asymmetrical and penalizes both right edges in the X direction and upper edges in the Y direction.

C. Hierarchical Layout Compaction

The majority of the hierarchical layout compactors reported in literature focus on solving the pitch matching problem, which means that certain elements among different cells must match in size and position when cells are abutted. The method described in [13] first compacts leaf cells and then locates the abutting ports to fixed grids. After that, the compacted cells are assembled and compacted at the higher level of hierarchy.

The pitch matching method introduced in [14] matches the abutting ports by directly adding abutting constraints between cells to the constraint graph. This method assumes that hierarchical layout is given. The constraints for all leaf cells have to be solved together. As the number of leaf cells increases, this method will be limited by its capacity.

A powerful pitch matching algorithm is reported in [15] based on port abstraction method. The port abstraction graph can be considered as a simplification of the constraint graph for each cell, where constraints unrelated to the ports are removed. The longest path between ports are computed and the port locations of each cell are then solved by solving the combined port abstraction graph of the circuit that uses the cell to be migrated. The result is then set as constraints to drive the migration of each leaf cell. The port abstraction method is very powerful in solving pitch matching problem. However, the main problem related is that the port position is decided by the longest path algorithm, which means that the ports will be placed as close as possible to each other. It implies that the total layout area will be minimized and it is not suitable to handle other objective functions. So our migration tool takes another approach called *dual-pass strategy* that can decide port locations with considerations of both design rules and the objective function.

D. 2-D compaction

Many techniques have been proposed for simultaneous two-dimensional compaction [16], [17], [5], [18], [19], [20]. Proposed methods include zone-refining and simulated annealing. However, the zone-refining approach requires a reasonably optimal layout to start with, while the simulated annealing method may require a significant run time at each cooling stage. Our migration method works directly on the original layout to be migrated, and has a runtime comparable to 1-D compaction.

E. Other Developments

The automatic jogging wires and wire length minimization are the two problems that have been investigated together with compaction problem. One of the first approach for wire jogging was presented in [10]. The usage of wire jogging is limited because it could reduce the layout size in one direction but potentially, increasing the size in another direction [6]. Our tool does not alter the layout topology and therefore does not address the automatic jogging insertion problem.

Another effort that somewhat alleviate the problem of over-compressing space region is the minimization of wire length along non-critical path. This is achieved by uniformly distributing spaces among circuit elements [10] or by “pulling” circuit elements back so that the length of connection wire is not increased drastically. In contrast, our migration tool attempts to optimize space directly.

As fabrication technology in the IC industry advances, some foundries are demanding the use of more complex rules such as conditional design rules. For example, some conditional rules require that the spacing of two edges depend on the context in which edges are situated [21]. Finding the optimal solution under conditional rule has been proven to be NP-complete. Heuristic method can be used to perform compaction for simple conditional rules such as bridge rules [21].

III. Problem Formulation

In this section, we present a formal model of the layout, design rules and a simplified problem formulation.

Unlike most reported migration tools, which operate on symbolic layout, our tool directly works on mask layout.

Layout consists of a set of polygons, each associated with a different layer, such as metal, poly, or diffusion. For simplification, people often constrain the polygons to be rectangular, called *Manhattan layout*, and organize related polygons of related layers in finite set of logical layers, called *planes* [22]. To simplify our analysis, we consider only Manhattan layout on a single plane. This is a reasonable assumption since the planes are organized in such a way that the *majority* of design rules constrain only polygons in the same plane.

Definition 1: A **layout topology** is a planar graph $G = \langle C, E = E_H \cup E_V, F, T \rangle$, where C is a set of geometric points, called **corners**, in a two dimensional plane, and $E_H \subset C \times C$ is a set of horizontal edges connecting two corners, and $E_V \subset C \times C$ is a set of vertical edges connecting two corners, and $F \subset 2^E$ is a set of faces, called **tiles**, each of which is an area covered by a cycle of edges, and $T : F \mapsto \mathcal{T} = \{space, poly, metal1, ...\}$ defines the layer type of each tile. Furthermore, the degree of each corner $c \in C$ should be no more than four.

Definition 2: A **layout** $\langle G, X \rangle$ is a layout topology G together with its geometric embedding $X : E \mapsto \mathcal{Z}$ such that each horizontal edge is defined with a coordinate in Y direction, and vertical edge a coordinate in X direction. Note that X fully determines the coordinate of each corner. For any edge $e \in E_V$, we denote y_e^b, y_e^t, x_e as the Y coordinates of its bottom corner, top corner and X coordinate respectively, and L_e, R_e its left and right tile respectively. For any edge $e \in E_H$, we denote x_e^l, x_e^r, y_e as the X coordinates of its left corner, right corner and Y coordinate respectively, and T_e, B_e its top and bottom tile respectively.

Example 1: A Manhattan layout is shown in Fig. 1.

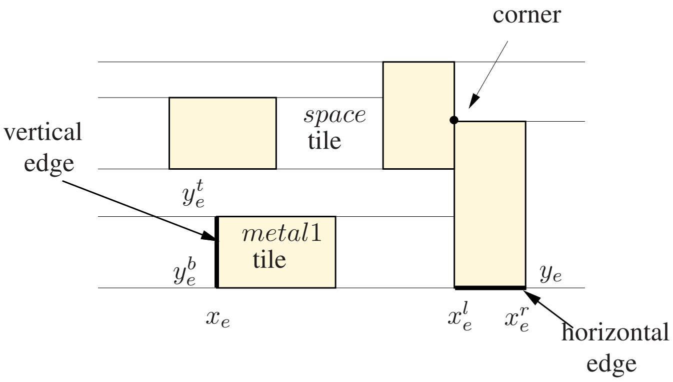


Fig. 1. A layout example.

Design rules are abstractions defined by the foundry to ensure manufacturability. They are typically defined in terms of constraints for the minimum width of a tile, minimum spacing between tiles, minimum extension between tiles, etc, which we call *macro rules*. The majority of macro rules can be translated into a set of constraints on layout edges, called *edge based rule*, which constrain the spacing between overlapping edges. Without loss of generality, we only consider the application of edge-based design rules along the X direction from left to right, and it can be then formally defined as follows.

Definition 3: A **design rule** r is a tuple $\langle t_1, t_2, t_3, d, cd \rangle \in \mathcal{T} \times \mathcal{T} \times \mathcal{T} \times \mathcal{Z} \times \mathcal{Z}$ constraining the spacing between any vertical edges $e1, e2 \in E_V$, where $T(L_{e1}) = t_1$, $T(R_{e1}) = t_2$, $T(R_{e2}) = t_3$, $[y_{e1}^b - cd, y_{e1}^t + cd] \cap [y_{e2}^b, y_{e2}^t] \neq \oslash$, such that $x_{e2} - x_{e1} \geq d$.

Example 2: A layout that violates design rule $\langle t_1, t_2, t_3, d, cd \rangle$ between e_1 and e_2 is shown in Fig. 2.

Definition 4: Given a layout $\langle G, X^{old} \rangle$ and a set R of

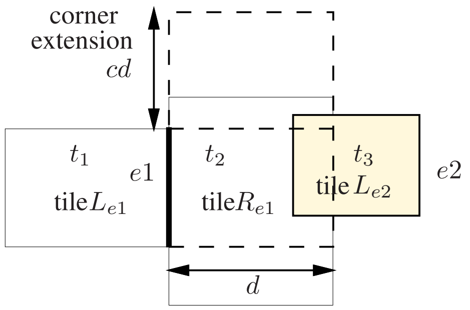


Fig. 2. An edge rule example.

design rules, a layout migration problem finds a new layout $\langle G, X \rangle$, such that it is design rule checking (DRC) clean, or every design rule $r \in R$ is satisfied.

IV. Iterative ILP framework by Geometric Closeness

In this section, we describe our integer linear programming (ILP) based migration framework, which generate the migrated layout by determining new positions of horizontal and vertical edges of all tiles.

Our migration framework is iterative. Each iteration resembles a classical 1-D migration framework: it first migrates along the X direction, or determining positions of vertical edges, and then the Y direction, or determining positions of horizontal edges. Without loss of generality, in the discussion that follows, we assume migration in the X direction only.

A *constraint-based migration* can be formulated as an integer linear programming (ILP) problem:

$$\boxed{\begin{array}{lrcl} minimize & o^T x & & \\ subject\ to & Ax & \geq & b \\ & x & \geq & 0 \end{array}}$$

Here x is a vector of variables to be determined, and in the simplest case would just be coordinates for all vertical edges. Vectors o represent the coefficients of x in the objective function of optimization. A is the constraints, each row of which represents coefficients of x in an inequality. Typically, an inequality is in the form of *difference inequality* $x_i - x_j \geq b_k$, where the constant b_k represents the minimum distance between two edges.

The constraint generation is a key component of the migration process. It discovers the relative position requirements between layout edges coming from different sources. As detailed in Section V, one source of constraints is imposed by the *design rules*. As detailed in Section VII, the second source of constraints is imposed by *high-level architectural requirements*, which have to do with the global structure of the entire library, rather than individual cells. This includes the routing track constraints, the power line constraints, the transistor size constraints and the port matching constraints. The third source of constraints, unfortunately, are *artificial*. This is the primary reason that layout generated by 1-D compaction is inferior to 2-D compaction. We examine the sources of such constraints and elaborate our treatment in Section VI.

The performance of migrated layout is largely influenced by the choice of objective function. The traditional minimum area objective function takes layout area as the only criterion and shrinks area to a maximum extent without considering layout design issues such as keeping two important signals apart to reduce coupling between them. In order to take full advantage of the original design and make minimum changes to the migrated layout, *Minimum perturbation* objective function is proposed in [1], which is defined as:

$$minimize \quad \sum |x - x^{old}| \tag{1}$$

where x is the vector of variables that determine X coordinates of all vertical edges and x^{old} is the vector of constants that are the old X coordinates of all vertical edges in the layout. The minimum perturbation function minimizes the position changes of all edges and snaps the edges to their original positions as much as possible. However, the disadvantage of this function is that it minimizes the absolute coordinates of edges and will penalize more on the movement of edges on the right side of the layout.

Example 3: Consider a simple layout given in Figure 3 with two tiles of $metal2$ type. Because of the technology change, the minimum width requirement of tile 1 is changed from 4λ to 5λ and distance requirement between tile 1 and tile 2 is changed to 5λ too. Based on minimum perturbation objective function, the layout will be migrated to the one on the lower part of Figure 3. The right edge of tile 1 will be stretched rightward by 1λ and left edge of tile 2 will be moved rightward by 2λ. However, the right edge of tile 2 stays at the old position because without change of its position, its width is 5λ which satisfies the new design rule. Otherwise, the objective function value will be greater than the one with $x_2^{'}$ unchanged.

It can been seen from this example that the movement of edges on the left side will add penalties on the edges on the right side if we want to preserve the original shape of tiles on the right.

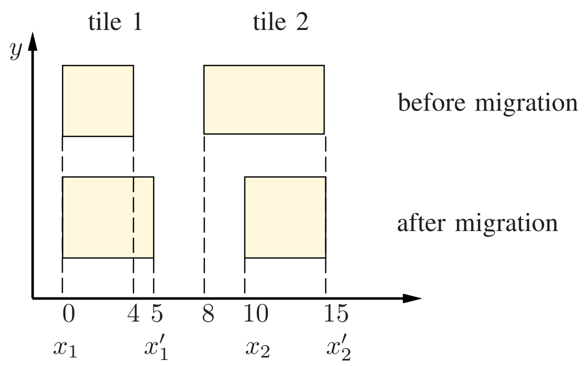


Fig. 3. The old layout and migrated layout with minimum perturbation objective function.

To remove this penalty, a new objective function, *geometric closeness* objective function is used in our migration tool which is defined as :

$$\sum \left|(x_{ri} - x_{li}) - (x_{ri}^{old} - x_{li}^{old})\right| \tag{2}$$

Here, x_{ri} and x_{li} are the X coordinates of the right and left edges of each tile in the layout. The constants x_{ri}^{old} and x_{li}^{old} are the X coordinates of the right and left edges of the corresponding tile in the original layout. Instead of minimizing the absolute coordinate changes of edges, this function minimizes the shape changes of all tiles so that each tile change will not add penalty to other tiles.

Example 4: With geometric closeness objective function, the example given in Figure 3 will be migrated into the layout in Figure 4. It can be seen that the width of tile 2 is set to the original value and the topology of the old layout is preserved to a maximum extent.

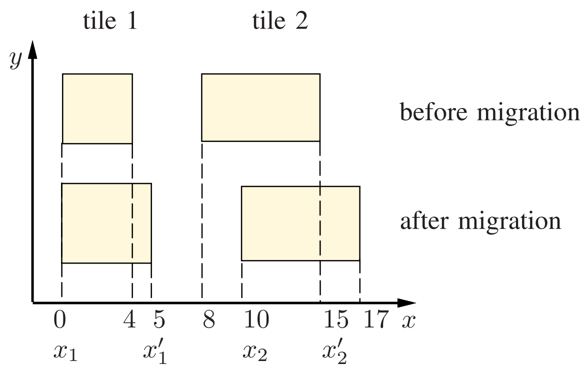


Fig. 4. The old layout and migrated layout with geometric closeness objective function.

To linearize, or to remove the absolute value computation in (2), we use a method similar to [1] by introducing two variables R_i and L_i for each rectangle, such that (3) are introduced as constraints,

$$\begin{aligned} R_i &\geq x_{ri} - x_{li} \\ R_i &\geq x_{ri}^{old} - x_{li}^{old} \\ L_i &\leq x_{ri} - x_{li} \\ L_i &\leq x_{ri}^{old} - x_{li}^{old} \end{aligned} \tag{3}$$

and the objective function is replaced by (2):

$$\sum_i (R_i - L_i) \tag{4}$$

V. Design Rule Constraint Generation

The variable to be determined in layout migration is the X/Y coordinates of each layout edge, or equivalently, the coordinate of each tile's left-bottom corner. The primary goal is to obtain the values of these variables such that migrated layout is DRC clean. Deriving the design rule constraints is therefore the most important task. Unlike design rule checking against a fixed layout, where design rules need only to be checked for neighboring edges, design rule constraint generation has to assume that every edge can potentially move, and therefore has to generate constraints for every pair of edges in a naive solution.

We consider constraints for vertical edges. Constraints for horizontal edges can be derived in a similar fashion. To obtain more insight into the problem, we first build a graph that can capture the neighboring relation.

Definition 5: Given a layout topology $G = \langle C, E, F, T\rangle$, its neighborhood graph $N = \langle V, E^N\rangle$ is a directed graph whose vertices correspond to the the tiles of G, and an edge $\langle u, v\rangle \in E^N$ iff there is a common edge between the tiles corresponding to u and v.

It is important to note that our input mask layout is directly captured by Ousterhout's corner-stitch data structure [22], where layout edges are already total-ordered when stored on disk. The neighborhood graph is thus implicitly available and takes no time to build.

We first observe that since constraints should be generated only for layout edges overlapping in the Y direction, the neighborhood graph can help to prune the search space: only pairs of tiles that are connected by a path in N need to be considered. The candidate pairs can be identified by finding the transitive closure of the neighboring graph. This can be achieved by iterating on each tile, called the source tile, where a preorder depth-first traversal of its descendents is performed, adding the closure edges along the way.

We can then apply the shadowing principle to further prune the size of the transitive closure: during the depth-first traversal, we first examine if the left edge of the discovered tile overlaps with the source tile under consideration, and stop further exploration of the tile if it turns out false.

While the shadowing was effective in the past to limit search space in the Y direction, we propose a new strategy to further limit the search space in X direction. We observe that as we explore along the X direction, each tile has a minimum width requirement dictated by the design rule. This can be summed up along the path and be used as the lower bound estimate of the distance between the source tile and the current tile. If the lower bound exceeds the constraint distance d of the design rule under consideration, further exploration of the current tile can be stopped.

Example 5: Consider the layout example in Fig. 5 (a). The shadow is indicated by the dashed line. A faction of the neighborhood graph is shown in Fig. 5 (b), where each tile is labeled with its minimum width requirement of d_1, d_2 etc. Suppose the constraint distance is D, and $d_1 + d_2 + d_3 \leq D$, $d_1 + d_2 + d_3 + d_4 \geq D$, then tile 10 and tile 11 will be pruned from the search space.

As such, it is guaranteed that there exists an upper bound of the *depth* we have to search. Algorithm 1 shows how the closure graph defined in Definition 6 can be built, upon which design rules can be directly constrained by examining the design rule set.

Definition 6: Given a neighborhood graph $N = \langle V, E^N\rangle$, its **depth-K shadowing closure** is a directed graph $\overline{N} = \langle V, \overline{E^N}\rangle$ such that $\langle u, v\rangle \in \overline{E^N}$ iff $\exists p \in u \rightsquigarrow v.|p| \leq K$ and $[y_u^b, y_u^t] \cap [y_v^b, y_v^t] \neq \emptyset$.

Algorithm 1: Depth-K shadowing closure algorithm.

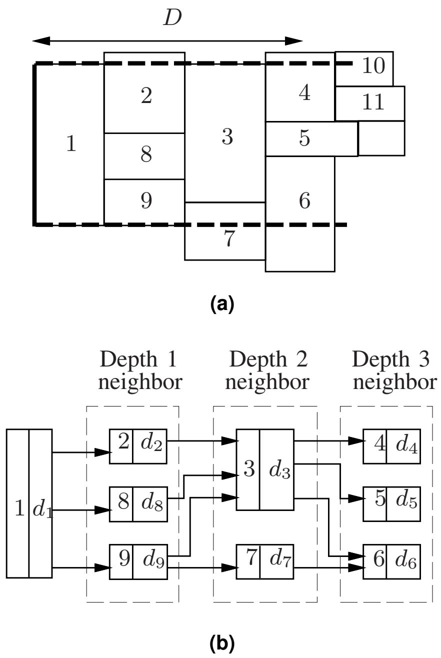


Fig. 5. (a) A layout consisting of tiles $1 - 11$; (b) Neighborhood graph.

input*:* $N = \langle V, E^N\rangle$;
output*:* $\overline{N} = \langle V, \overline{E^N}\rangle$;
func *explore(*u, v, d*)* {
  if($[y_u^b, y_u^t] \cap [y_v^b, y_v^t] = \oslash \vee d{+}{+} > K$)
    return;
  $\overline{E^N} = \overline{E^N} \cup \langle u, v\rangle$; *// add depth-d closure edges*
  forall($w \in \{w | \langle v, w\rangle \in E^N\}$)
    explore(u, w, d*);*
}
func *depthKShadowingClosure()* {
  forall($u \in V$) {
    forall($v \in \{v | \langle u, v\rangle \in E^N\}$)
      explore($u, v, 1$*);*
  }
}

We now consider the complexity of Algorithm 1. We are interested to determine both the time complexity, and the space complexity in terms of the number of constraints it generates. The latter is important since it determines the runtime of the solver. Since Algorithm 1 generates one constraint at a time, the time and space complexity is in fact equivalent.

It is important to first observe that the number of edges in a planar graph is in the same order of vertices.

Lemma 1: The number of edges in the neighborhood graph $N = \langle V, E^N\rangle$ is of $O(n)$, where $n = |V|$.

Proof: N is the dual of the layout topology $G = \langle C, E, F, T\rangle$. N is therefore also planar. Let F^N be the set of faces of N, then according to Euler's formula, $|V| + |F^N| - |E^N| = 2$. Since $\forall f \in F^N.|f| \geq 3$, we have $2|E| = \Sigma_{f \in F^N}|f| \geq 3|F^N|$. By Euler, we have $|E| \leq 3|V| - 6$. □

Unfortunately, we have to search more than the immediate neighbors. We now consider the closure edges, formed by the so-called *triangle principle*.

Example 6: Consider a depth-K_1 edge $\langle a, b\rangle$, and a depth-K_2 edge $\langle b, c\rangle$, as shown in Fig. 6 (a), a depth-K_3 edge is formed as long as $K_3 = K_1 + K_2$, and $K_3 \leq K$.

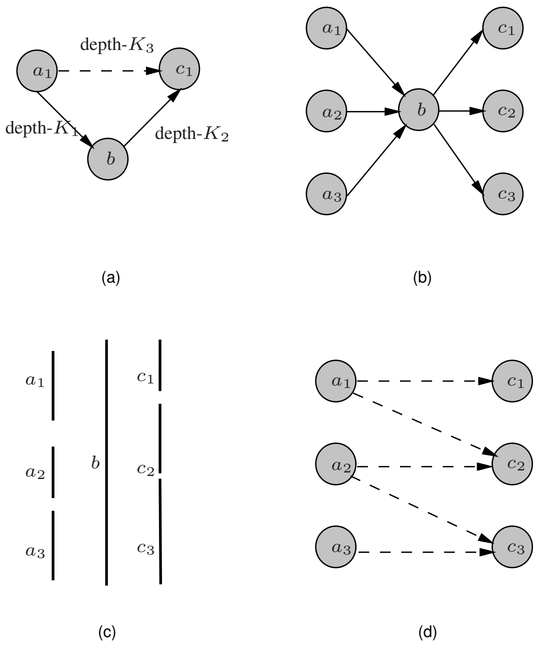


Fig. 6. Formation of closure edges.

Lemma 2: Let $K_1, K_2, K_3 \leq K$ be integer numbers such that $K_3 = K_1 + K_2$. If the number of depth-K_1 closure edges is of $O(n)$, and the number of depth-K_2 closure edges is of $O(n)$, then the number of depth-K_3 edges formed by depth-K_1 and depth-K_2 edges are of $O(n)$.

Proof: Consider all edges of the form $\langle a_i, b\rangle$ and $\langle b, c_j\rangle$, as shown in Fig. 6 (b), where $\langle a_i, b\rangle$ is of depth-K_1 and $\langle b, c_j\rangle$ is of depth-K_2. We first observe that all vertical edges corresponding to $a_1, a_2, ...$ do not overlap: if they do, without loss of generality, assume a_1 overlaps a_2, then we can conclude that the depth of $\langle a_1, b\rangle$ is not the same as the depth of $\langle a_2, b\rangle$ – a conflict. Similarly, as shown in Fig. 6 (c), we can conclude that all vertical edges corresponding to $c_1, c_2, ...$ do not overlap. The valid closure edges $\langle a_i, c_j\rangle$ are therefore *not* formed by every pair of a_i, c_j. They can instead found by following a top-down scanline algorithm. It follows that the number of closure edges is at most the sum of edges $\langle a_i, b\rangle$ and edges $\langle b, c_j\rangle$. Therefore, the complexity of depth-K_3 edges is the same as the complexity of their formation edges, and our conclusion follows. □

We are now ready to give our main complexity result.

Theorem 1: The number of constraints generated by Algorithm 1 is $O(n)$, where n is the number of tiles in the layout, or vertices, in the neighborhood graph $N = \langle F, E^N\rangle$.

Proof: We prove by induction that the complexity of depth-I closure edges, where $I \leq K$, is of $O(n)$. When $I = 1$, the claim is true according to Lemma 1. We now assume that the complexity of depth-1, depth-2, ..., depth-$(I-1)$ is of O(n). The depth-I closure edges are formed by triangulating depth-1 and depth-$(I-1)$, depth-2 and depth-$(I-2)$, etc. According to Lemma 2, we can conclude that the complexity of depth-I edges is of $O(I \times n)$. Since $I \leq K$ where K is a small constant, it follows that the complexity of depth-I, including depth-K edges, is of $O(n)$. □

Example 7: Fig. 7 shows the constraint graph generated by Algorithm 1 for a multiplexor whose layout is shown in Fig. 12 (a).

VI. Soft Constraint Based Migration

In the 1-D migration strategy, X direction and Y direction are migrated separately, and artificial constraints are introduced to ensure that there is no arbitrary movement of layout rectangles in one direction when performing migration in the other direction. However, these artificial constraints compromise the optimality of the migrated layout. An improvement is to relax these constraints by introducing additional variables, called *elastic* variables. With the elastic variables, the artificial constraints can be violated under a controlled way.

In a corner-stitched representation, layout rectangles, or tiles, are organized in different planes. Accordingly, design rules are imposed between tiles within the same plane, called *intraplane* rules, and across different planes, called *interplane* rules. In the 1-D migration framework, artificial constraints are introduced for both intraplane and interplane rules.

When migration is performed in the X direction, the depth-K shadowing closure algorithm generates constraints on relative position of tiles according to their Y coordinates. When we proceed to migrate in the Y direction, these assumptions have to be preserved. Otherwise the solution along the X direction will no longer be valid, or DRC-clean. For example, in Fig. 8, the following artificial constraints have to be generated during the Y compaction.

$$\begin{aligned} y_2^t - y_1^t &\geq 0 \\ y_1^b - y_3^b &\geq 0 \\ y_1^b - y_4^b &\geq 0. \end{aligned}$$

Here y_{it} denotes the top edge of the ith tile and y_{ib} denotes the bottom edge of the ith tile respectively.

We relax these constraints by introducing three additional variables s_1, s_2, s_3 and rewrite the above constraints as follows

$$\begin{aligned} y_2^t - y_1^t &\geq 0 - s_1 \\ y_1^b - y_3^b &\geq 0 - s_2 \\ y_1^b - y_4^b &\geq 0 - s_3 \\ s_1, \ldots, s_3 &\geq 0 \end{aligned}$$

Artificial constraints associated with interplane rules are similar, except that in this case, the source edge and the constraint area are on separate planes, which are migrated independently. We therefore need two additional artificial constraints on the X direction to ensure that the relative position of tiles in the constraint area to the source edge are kept. For example, in Fig. 9, the two additional artificial constraints are $x_{edge} - x_1^l \geq 0, x_1^r - x_{edge} \geq 0$, where x_{edge} denotes the position of the source edge, x_1^l and x_i^r denotes

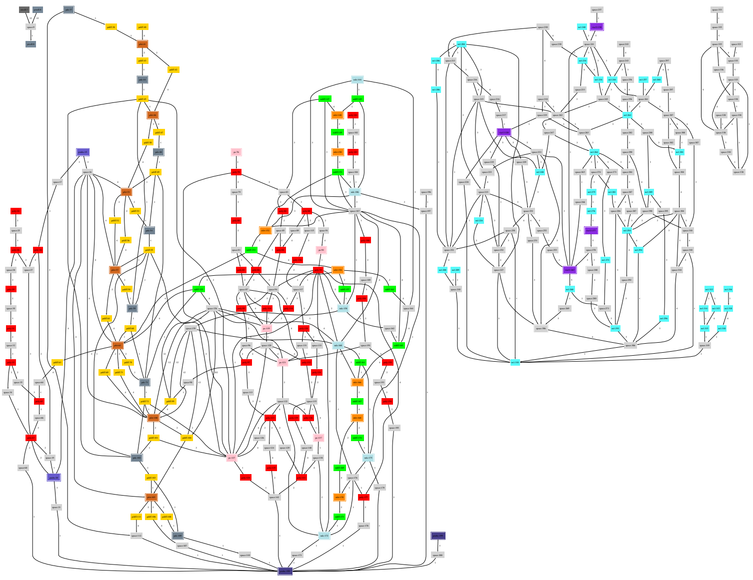


Fig. 7. Constraint graph for a multiplexor.

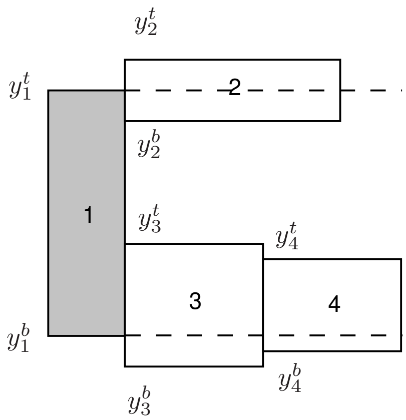


Fig. 8. Intra plane artificial constraints.

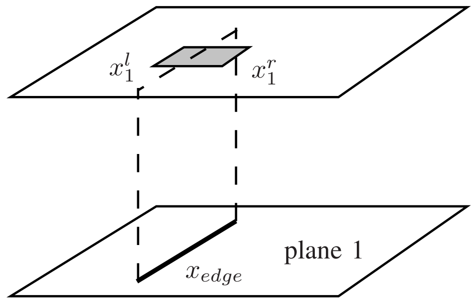


Fig. 9. Interplane artificial constraints.

the left and right edge of tile 1 respectively. Similarly, these constraints can be formulated as soft constraints as follows,

$$
\begin{aligned}
x_{edge} - x_1^l &\geq -s_1 \\
x_1^r - x_{edge} &\geq -s_2 \\
s_1 &\geq 0 \\
s_2 &\geq 0
\end{aligned}
$$

We now consider how we can control the elastic variables so that ultimately we can obtain DRC-clean layout. The idea is to use the objective function. Our objective function is formulated as

$$o^T x + C \sum_i s_i,$$

Here the term $o^T x$ is the same as the objective function of the original one in 1-D migration. The term $C \sum_i s_i$ accounts for the contributions of soft constraints. Variables s_i denotes the elastic variables introduced in generating soft constraints. The scalar C is used as weight factor.

It is easy to see that if the value of C is small, we practically relax all the artificial constraints introduced and approximating a 2-D migration. When the value of C approaches infinity, requiring all the elastic variables s_i to approach zero, the problem becomes the same as the original 1-D migration.

To migrate the cell with soft constraints, we adopted an iterative optimization procedure in which the value of the weight factor C is adjusted at each iteration. Starting with a small value of C, the objective function described above, together with both the soft and hard constraints, is fed into an ILP solver, after which solutions for each edge position are found. We migrate the cell based on the solution. Next we check to see if the cell is free of design rule violations. If so, we stop at this point, and return the migrated cell. Otherwise, a new set of soft and hard constraints for the migrated cell are derived in a similar manner for the original cell. The value of C in the objective function is updated with an appropriate step size. The new objective functions and constraints are again fed to ILP solvers to find solutions for next migration. The entire procedure is repeated until the migrated layout is free of design rule violations. The scheme for updating C can affect the optimality of the solution obtained. In our experiments, we used the following scheme:

initial value	$C = 0.001$
1st iteration	$C = 1$
succeeding iterations	$C_{new} = 10 * C_{old}$

VII. Architectural Constraint Satisfaction

The library IPs typically have predefined layout architecture. For example, all standard cells are constrained with the same height. As another example, the datapath cells have predefined routing architecture where the data signals run horizontally, and the control signals run vertically [23]. These architecture features have to be maintained during migration, in addition to satisfying design rules. In this section, we examine how such requirements can be accommodated in our ILP framework by the generation of architectural constraints.

A. Port Matching

The area efficiency in the datapath is achieved by the *tiling* strategy where important signals, such as controls and carries, are implicitly routed by abutment. This requires the ports of different cells to *match* exactly in position and size, as shown in Fig. 10.

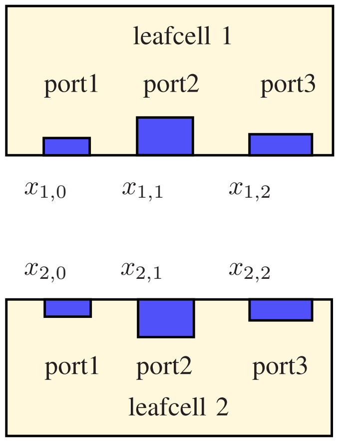


Fig. 10. Port matching of leaf cells.

One naive way of translating the port matching constraints is to make the relative port position on the cell boundary and port widths equal for all matching cells. The constraints result from Fig. 10 are then:

$$x_{1,1} - x_{1,0} = x_{2,1} - x_{2,0} \tag{5}$$

$$x_{1,2} - x_{1,1} = x_{2,2} - x_{2,1} \tag{6}$$

The problem of this approach is that the constraints bind variables from different cells together. Therefore they have to be migrated simultaneously. This may increase the number of variables in the underlying ILP solver substantially. With large datpath library, this approach quickly becomes infeasible.

We employ a *dual-pass strategy* to break the interdependence between different leaf cells. Our tool accepts a library of datapath leaf cells with layout information from one process, and a specification of the library architecture. In the first pass, it analyzes the layout of each leaf cell, generates the design rule constraints, and drives an ILP solver under the geometric closeness objective function to arrive at a temporary migration solution of each cell. Note that the ILP problems are solved separately for different cells. In the second pass, the different architectural and circuit requirements are translated into linear constraints. Here, the temporary solution obtained in the first pass is exploited so that the new constraints relate variables originating from the same cell. The ILP solver is then called again to obtain the new and final solution to accommodate the new constraints. Note again that the ILP problems are solved separately for each cell.

For the previous example, we introduce a pair of constants, d and w to break the dependency between different cells:

$$x_{1,1} - x_{1,0} = d \quad x_{2,1} - x_{2,0} = d \tag{7}$$

$$x_{1,2} - x_{1,1} = w \quad x_{2,2} - x_{2,1} = w \tag{8}$$

We obtain d and w by taking the maximum value of $x'_{i,1} - x'_{i,0}$, and $x'_{i,2} - x'_{i,1}$ among all the leaf cells that have to be matched, where x' is the temporary solution we obtain in the first pass. This method effectively stretches the port whose distance to boundary can be smaller than the maximum value among all the leafcells, making all ports align together.

B. Routing Track Matching

Data signals in the datapath are always routed horizontally (perpendicular to control signals), and over-the-cell. Typically, they have to be aligned to a routing grid, for which the new migrated library may be different than the original. For example, for a leaf cell layout given in Figure 11, and a routing grid characterized by $\langle rs, ro, rw \rangle$, representing the routing pitch, offset and width respectively, the Y coordinate of the lower and upper edge of each routing track, y_{si} and y_{wi}, must satisfy constraints:

$$y_0 + ro + K \cdot rs = y_{si} \tag{9}$$

$$K \geq 0 \tag{10}$$

$$y_{wi} - y_{si} = rw \tag{11}$$

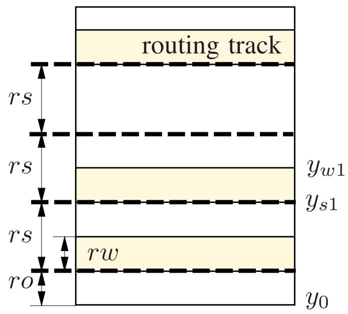


Fig. 11. Cell with routing track for data signals.

C. Power/Ground Net Size and Transistor Sizing

There are two kinds of constraints involved with power/ground nets. First, power/ground ports of different cells have to match. This can be solved by the port matching method described earlier. Second, the width of the power/ground net needs to satisfy the architectural specification. Often, the width is determined by separate power/grid design methodology and layout migration has to faithfully follow the specification. These constraints are expressed as follows:

$$x_{i,2} - x_{i,1} = W_P \tag{12}$$

where $x_{i,2}$ and $x_{i,1}$ are the two edges of the power/ground net and W_P is the user specified width of power line.

Similarly, a circuit-level transistor sizing tool may determine an optimal transistor size that is different from the original layout, and it will rely on the migration tool to perform the change. After the identification of transistors in the layout, the sizing requirement can be expressed as an equality constraint. Note that we only expect modest change in the transistor size, and therefore mildest change in layout topology. For example, the introduction or elimination of transistor fingers is not needed.

VIII. Limitations

The proposed techniques were implemented in an academic layout migration tool, called *calligrapher*, which targets hard IP libraries. The tool is built on top of an IP-centric CAD infrastructure, called *ipsuite* [24], which uses Magic's corner-stitching data structure [22] for layout manipulation at the mask level, as well as its design rule database compatible to those offered by MOSIS [25]. An open-source ILP solver is used as well [26]. The migration tool itself is implemented by 15K lines of C code.

To understand the capabilities and the limitations of the tool, it is important to first draw the line between layout migration and layout optimization. The goal of layout migration is to obtain a DRC-clean layout, while retaining its performance as much as possible. The change to the layout topology, in our case none, is expected to be minimal. If a dramatic layout change is needed for the new process, then layout migration is not an appropriate technology to use. Instead, either manual redesign or layout synthesis should be performed.

While the minimum perturbation and geometric closeness metric are designed to alleviate physical problems in deep submicron processes, such as crosstalk, the problems are dealt with rather indirectly. For example, if the original layout was designed to avoid crosstalk by spacing the aggressor and victim apart, then our tool is able to retain that design intention. However, our tool does not intend to alternate the original layout, for example, by inserting buffers, to combat crosstalk.

Modern processes often offer more routing layers. With the constraint of maintaining layout topology, our tool is not able to take advantage of it. However, since our tool targets only cell libraries, which use the first few metal layers for intra-cell routing, this is not a significant issue. The better use of the additional metal layers are usually for inter-cell signal routing, or the power grid, or the clock network.

Our focus on the library IPs also makes the use of ILP affordable since in general we migrate one cell at a time. The power of ILP allows one to express complex constraints. For example, device matching constraints, which are common in analog IPs [3], can be readily processed under an ILP framework. Such constraints are difficult to handle in the traditional compaction frameworks, which are based on shortest path algorithm. Our tool has the potential to be extended to migrate analog IPs.

Modern processes, especially those at the 90nm technology node, introduce complex design rules that are beyond the modeling capability of the design rule model used in this paper. Some of those rules are conditional rules, which can be modeled by extension of edge-based design rules [27], and be dealt with by techniques similar to Cheng et. al. [21]. Some of those rules are rules that target manufacturability under sub-wavelength technology. The handling of such rules is still an open question [4] and is outside the scope of this work. In our academic tool, we primarily use the design rules offered by MOSIS [28], which works around many of the problems by using conservative design rules (scmos-subm, scmos-deep).

IX. Experimental Results

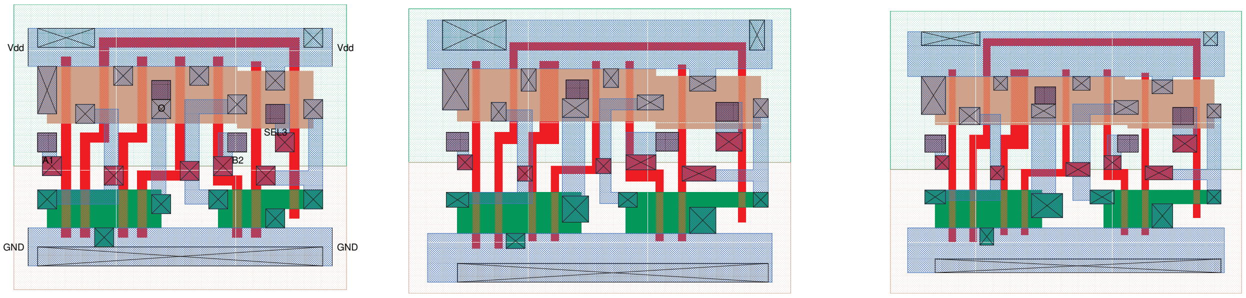
To test the effectiveness of our tool, we apply our tool on the low-power standard cell library and datapath library developed by Burd [23] at University of California, Berkeley. This library was based on SCMOS 1.2μm technology. The standard cell library contains 94 cells, including gates performing common logic functions with different drive strength. The datapath library contains 285 cells, including those to construct adders, shifters, multiplexors and register files. Both libraries are silicon proven and have been used at Berkeley for at least a dozen chips (`www.faqs.org/faqs/lsi-cad-faq/part4`). Our targeted processes are TSMC 0.25 μm and TSMC 0.18μm technologies. All cells are migrated successfully: they are all DRC clean and do not require manual editing. All experiments are carried out on a SUNBlade 100 system running at 500 MHZ.

A. Sample Layout

In this section we show sample migration results to demonstrate the effectiveness of the proposed method.

We first compare migration result produced by geometric closeness (GC) metric, and the minimum perturbation metric (MP). Figure 12 (a) shows a two-input multiplexer leaf cell layout in the standard cell library. Figure 12 (b) shows the layout migrated to TSMC 0.25μm technology based on the MP metric and Figure 12 (c) shows the layout migrated to the same technology based on the GC metric. Inspecting the contact in the left hand corner of the layout, it can be observed that the GC metric produces better result than the MP metric.

We then compare results generated by the regular 1-D framework and our soft constraint based 2-D framework. Fig. 13 (a) shows the layout of the *scantspcr_cs1* cell in the library. Fig. 13 (b) and (c) show the migrated layout by the



(a) Original layout (b) Migrated in TSMC 0.25μm with MP (c) Migrated layout in TSMC 0.25μm with GC

Fig. 12. A two-input Multiplexer.

traditional 1-D framework and by our soft constraint based 2-D migration. The widths of $metal1$ that are noticeably stretched in the 1-D migration layout. This is not the case for 2-D migration. In addition, several contact stretches observed in the 1-D migration are also avoided in our method.

Similarly, Fig. 14 shows the migration result for *shcs* cell. The width of the vertical metal 1 in the middle of the cell in (c) is not as stretched as in (b). In addition, the aspect ratios of the contacts between metal1 and p-diff is better preserved in (c).

Fig. 15 shows the migration result for cell *csamsbodd*. The stretch in the y-direction is significantly reduced in the new migration method. It is apparent that the cell migrated in the new method not only resembles the original cell much better, but also has significantly smaller area.

To show that layout architecture can be handled properly, in Fig. 16 we show the migration result of an 8-bit adder, which is constructed by abutting of eight identical full adder cells as well as a control cell. Note that many of the ports, for example *carryin* signal and *carryout* signal need to be matched between cells. And power line widths are user defined.

B. Quantitative Results

In this section, we provide a more comprehensive, quantitative results.

We first show in Table I the runtime and quality of our baseline migration algorithm (without soft constraint based iteration). Here, related cells that need port matching are migrated in groups. The number of tiles (including space) in the layout, and the total numbers of design rule constraints generated are shown in the third and forth column. The fifth column and sixth column demonstrate the run time for ILP solver and constraint generation respectively (measured in seconds). The last two columns, *MP layout change* and *GC layout change* present layout change values for layout based on minimum perturbation objective function and geometric closeness perturbation objective function respectively. As we can observe, the ILP solution dominates the computation time. Since our dual-pass framework allows us to run ILP one cell at a time, the total migration time is limited in minutes and is therefore tolerable. It is interesting to note that while both the standard cell and datapath library is designed very compactly and use minimal size as much as possible to reduce power consumption and leave very little flexibility in the solution space, GC and MP produces as much as 20-45% difference for some cases.

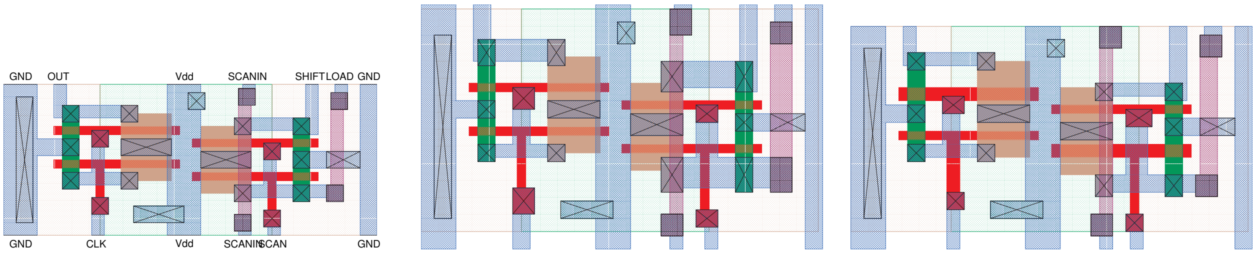
We demonstrate the effectiveness of soft constraint based 2-D migration in Table II. The value of the geometric closeness (GC) objective function are evaluated for (a) traditional 1-D migration, (b) each iteration in the soft constraint based migration, and (c) overall value for the soft constraint based migration. The overall value of the objective function for soft constraint based migration are calculated by adding that of each iterations. Comparing the results in column (a) and column (c) suggests that in general, our method can find a better solution to the minimization problem than traditional 1-D migration. Furthermore, we note that at most three iterations are required to obtain the final solution, which means that the running time for our method is at most four times longer than the regular 1-D migration.

Finally in column 9-13 of Table II list the detailed area result: Column 9 (orig) corresponds to the layout area after linear scaling, which is not DRC-clean. Column 10 (MA) lists the area obtained by minimum area layout compaction. Column 11 (MP) lists the area obtained by minimum perturbation metric. Column 12 (GC) lists the area obtained by 1-D geometric closeness metric. The last column (2D) lists the area obtained by geometric closeness metric under the 2-D iterative framework. It can be observed that our solution is approximately 11% better than minimum perturbation method in area, and only 7.5% worse than minimum area compaction, even though area is not our primary optimization goal.

X. Conclusion

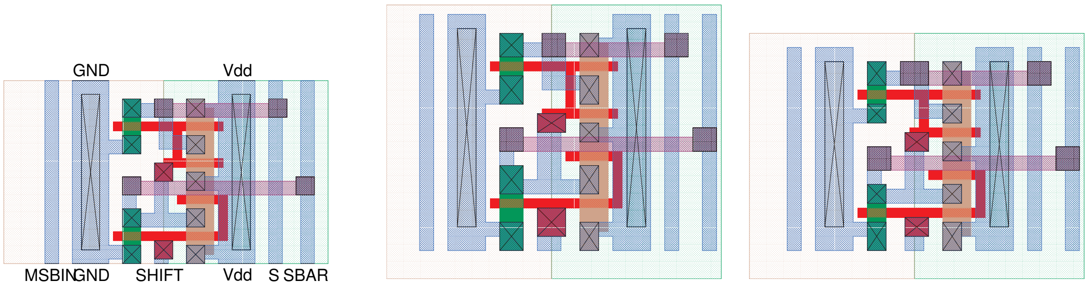
In conclusion, we have argued that migration technology is essential for the success of hard IPs. We have demonstrated a layout migration engine featuring a linear time/space constraint generation algorithm, a new optimization metric based on geometric closeness, and a soft-constraint based iterative framework for 2-D migration.

Based on our study, we conclude that the proposed method is effective to handle the new challenges in modern design addressed by the recently proposed minimum perturbation based metric, while overcoming some of its limitations. In addition, our iterative framework can produce more compact layout: it improves area by 10% on average as compared to minimum perturbation based method, and inflates area



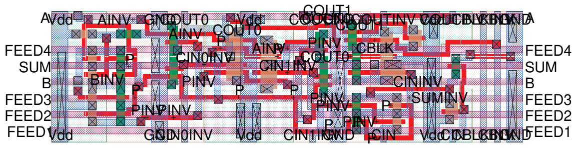
(a) Original layout (b) 1-D migration result (c) 2-D migration result

Fig. 13. Migration of scantspcr cell.

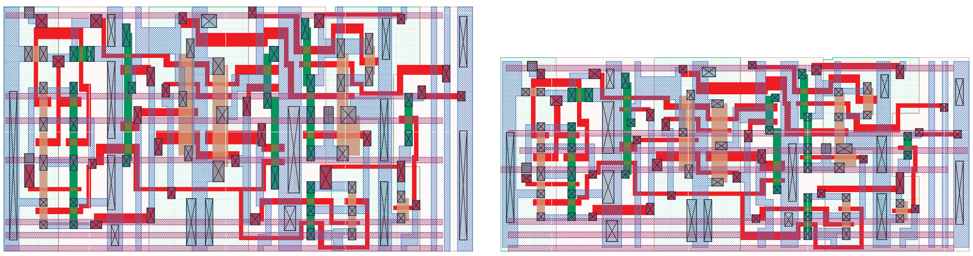


(a) Original layout (b) 1-D migration result (c) 2-D migration result

Fig. 14. Migration of shcs1 cell.



(a) Original layout



(b) 1-D migration result (c) 2-D migration result

Fig. 15. Migration of csamsbodd cell.

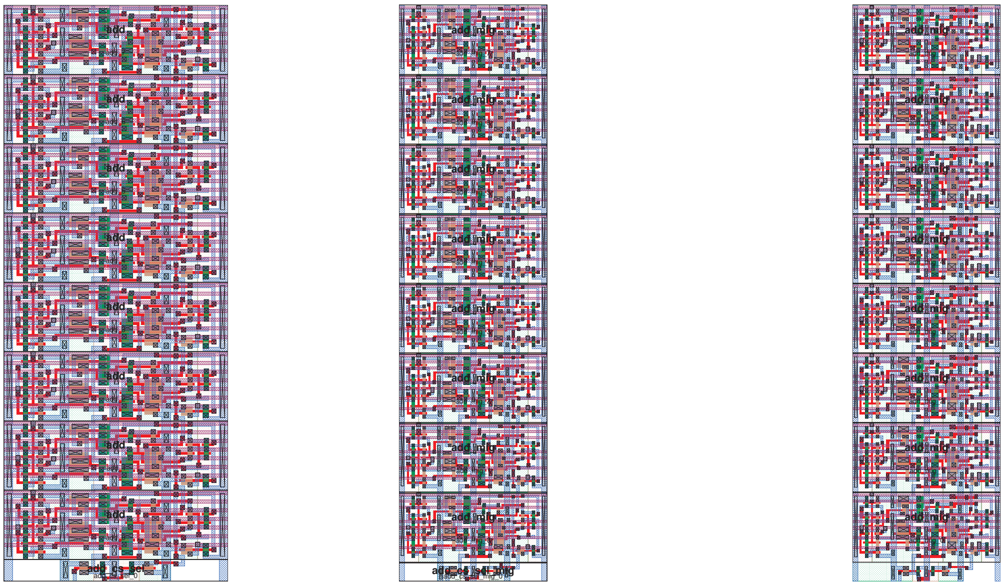
by merely 7.5% as compared to layout compaction using minimum area metric.

Our future work will extend towards the handling of complex design rules in modern processes, and the optimization of new manufacturability metrics.

References

[1] F.-L. Heng, Z. Chen, and G. E. Tellez, “A VLSI artwork legalization technique based on a new criterion of minimum layout perturbation,” in *International Symposium on Physical Design (ISPD)*, 1997, pp. 116–121.

[2] F. Fang and J. Zhu, “Automatic migration of datapath IP libraries,” in *Proceeding of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004.



(a) Original layout (b) Migrated layout in TSMC 0.25 μm (c) Migrated layout in TSMC 0.18μm

Fig. 16. Migration of 8-bit adder with GC.

TABLE I

RUN TIME RESULTS AND METRIC COMPARISON.

Library Name	Cell Name		# of tiles	# of DRC constraints 0.18μm / 0.25μm	ILP run time (s) 0.18μm / 0.25μm	DRC CG run time(s) 0.18μm / 0.25μm	GC layout change	MP layout change	
standard cell library	andf301		243	6208/6224	18.6/20.6	0.45/0.61	791	879	11%
	aof3201		463	14842/14878	107.4/110.5	1.16/1.17	1851	1918	4%
	blf00001		195	4181/4184	12.1/10.1	0.25/0.32	434	466	7%
	buff102		169	3550/2719	8.1/6.9	0.32/0.27	534	551	3%
	dfnf401		353	10643/10643	58.4/56	0.73/0.92	1145	1208	5%
	drif101		275	7636/7656	27.6/27.9	0.64/0.55	579	590	2%
	invf101		100	1583/1587	1.2/1.2	0.13/0.15	243	309	27%
	labf211		237	5906/5922	16.2/17	0.53/0.49	793	830	5%
	lrbf202		76	997/997	0.5/0.5	0.05/0.08	352	358	2%
	muxf201		337	10006/10034	40.5/49.5	0.81/0.74	1059	1137	7%
	nanf201		143	2707/2715	3.8/4.2	0.22/0.18	409	452	10%
	norf211		202	4699/4711	11.7/11.6	0.28/0.45	663	703	6%
	oaif2201		243	5951/5967	18.4/18.6	0.40/0.37	778	794	2%
	orf401		338	9947/9975	49.7/49.8	0.74/0.78	1798	1903	6%
	swcf020		154	3177/3185	5.0/5.2	0.24/0.33	418	483	16%
	xnof201		307	8892/8888	45.3/41.5	0.67/0.77	793	803	1%
	xorf201		303	8684/8660	41.5/37.3	0.70/0.66	732	742	1%
datapath libarary	adder	add	1079	36866/56127	1140/1110	149.5/134.9	2763	3044	10%
		add_cs_sel	167	3070/4288			269	309	13%
		add_cs_0	29	223/225			22	22	0%
	mux2	mux2	357	7745/11797	174/192	10.5/10.9	752	878	17%
		mux2_cs1	179	2203/4248			124	137	10%
		mux2_cs2	181	14574/14573			133	149	12%
		mux2_cs3	266	17512/17514			287	334	14%
		mux2_cs4	198	8518/11096			352	408	16%
	register	rf0	204	2748/2748	36/36	3.9/3.7	229	287	25%
		rf1	202	2756/2756			185	220	20%
		rf01_cs1	228	3517/3516			372	397	7%
		rf01_cs2	230	3584/3584			381	401	5%
		rf01_cs3	223	3464/3464			332	481	45%
	random logic	and2blp	300	5940/5940	198/198	6.6/6.4	552	587	6%
		nand2lp	240	4015/4015			294	304	3%
		nandand3lp	319	6706/6706			786	870	11%
		or2blp	273	5319/5319			384	411	7%
		xnor2lp	343	5319/5319			438	460	5%
		xor2lp	345	8003/8003			436	467	7%

[3] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "Automatic analog layout retargeting for new processes and device sizes," in *International Symposium in Circuit and Systems (ISCAS)*, 2003.

[4] F.-L. Heng, L. Liebmann, and J. Lund, "Application of automated design migration to alternating phase shift mask design," in *International Symposium on Physical Design (ISPD)*, 2001, pp. 116–121.

[5] Y. E. Cho, "A subjective review of compaction," in *Proceeding of the Design Automation Conference (DAC)*, 1985, pp. 396–404.

[6] D. G. Boyer, "Symbolic layout compaction review," in *Proceeding of the Design Automation Conference (DAC)*, 1988, pp. 383–389.

[7] N. Weste, "Virtual grid symbolic layout," in *Proceeding of the Design Automation Conference (DAC)*, 1981, pp. 225–233.

TABLE II
AREA RESULTS AND 2-D/1-D COMPARISON.

Cell Name	No. of tiles	(a) GC change X/Y	(b) GC layout change for soft constraint(X/Y)				(c) total GC change X/Y	(d) area				
			iter 1	iter 2	iter 3	iter 4		orig	MA	MP	GC	2D
cnt_top	117	10/48	10/21	0/7			10/28	3040	2400	4000	4025	3864
csainbuf	192	57/237	38/110	23/23	0/93		61/226	6264	3744	8388	8424	7956
csalsbeven	1053	759/3289	425/1709	34/68	167/464	50/285	676/2526	14848	27963	27963	28548	25920
csamsbodd	1180	920/4893	379/1708	84/35	166/310	53/392	682/2445	14848	32370	32370	32370	24644
invs	166	227/57	36/53	154/2	8/0		198/55	2560	2142	2688	2814	2814
mux2	343	224/347	21/216	10/121	40/0		71/337	3584	4644	4816	4988	4930
mux2_cs1	165	62/56	42/49	28/8	1/6		43/54	1680	1890	2030	2100	2065
mux2_cs2	167	61/66	41/54	28/13	2/6		43/60	1680	1995	2030	2100	2100
mux2_cs3	252	178/181	65/92	110/31			175/123	3080	3762	3828	3960	3599
mux2_cs4	354	197/197	86/109	5/22	4/42		100/197	4368	5369	5369	5551	5349
nand2lp	226	171/161	46/110	101/24			147/134	2944	3196	3243	3196	3012
noror2lp	265	265/185	131/60	22/38	22/24		175/122	3392	4032	4032	4032	3920
or2blp	259	226/185	38/99	153/34	15/60		206/193	3136	3876	3876	3876	3876
rf0	194	240/61	25/61	200/0	12/0		237/61	2688	2928	3072	3216	3016
scantspcr_cs1	280	127/442	92/222	23/93			115/315	3204	5040	5040	5152	4692
shcs1	219	113/286	64/108				64/108	2840	3538	4118	4118	3692
shl1bit	239	415/14	56/12	328/5	0/2		384/14	4544	3224	4615	4736	4736
shl1end	221	410/38	55/13	328/15	0/16		383/29	4544	3286	4615	4736	4736
shl1lsb	326	446/144	76/99	328/43	0/35		404/134	4544	4884	5254	5476	5024
tspcr	345	393/286	53/218	20/7	63/14		136/239	3392	4930	4930	4930	4731
Average									0.925	1.093	1.109	1.000

[8] A. E. Dunlop, "SLIM-the translation of symbolic layouts into mask data," in *Proceeding of the Design Automation Conference (DAC)*, 1980, pp. 595–602.

[9] R. C. Mcgarity and D. P. Siewiorek, "Experiments with the SLIM circuit compactor," in *Proceeding of the Design Automation Conference (DAC)*, 1983, pp. 740–746.

[10] M. Y. Hsueh and D. O. Pederson, "Computer-aided layout of LSI circuit building-blocks," University of California, Berkeley, Tech. Rep., 1979.

[11] Y. E. Cho, A. J. Korenjak, and D. E. Stockton, "FLOSS: An approach to automated layout for high-volume designs," in *Proceeding of the Design Automation Conference (DAC)*, 1977, pp. 138 – 141.

[12] P. T. Chapman and J. K. Clark, "The scan line approach to design rules checking: computational experiences," in *Proceeding of the Design Automation Conference (DAC)*, 1984, pp. 235–241.

[13] J. L. Burns and A. R. Newton, "Efficient constraint generation for hierachical compaction," in *Proceedings of the International Conference on Computer Design (ICCD)*, 1986, pp. 197–200.

[14] D. Marple, "A hierarchy preserving hierarchical compactor," in *Proceeding of the Design Automation Conference (DAC)*, 1990, pp. 375–381.

[15] C.-Y. Lo and R. Varadarajan, "An o($n^{1.5} logn$) 1-D compaction algorithm," in *Proceeding of the Design Automation Conference (DAC)*, 1990, pp. 382–387.

[16] M. Schiag, Y. Z. Liao, and C. K. Wong, "An algorithm for optimal two-dimentional compaction of VLSI layouts," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 1983, pp. 88–89.

[17] W. Wolf, R. Mathews, J. Newkirk, and R. Dutton, "Two-dimentional compaction strategies," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 1983, pp. 90–91.

[18] H. Shin, A. L. Sangiovanni-Vincentelli, and C. H. Sequin, "Two-dimensional compaction by zone refining," in *Proceeding of the Design Automation Conference (DAC)*, 1986, pp. 115–122.

[19] H. Shim and C.-Y. Lo, "An efficient two-dimensional layout compaction algorithm," in *Proceeding of the Design Automation Conference (DAC)*, 1989, pp. 290–295.

[20] R. C. Mosteller, "Monte carlo methods for 2-d compaction," California Institute of Technology, Tech. Rep., 1986.

[21] C.-K. Cheng, X. Deng, Y.-Z. Liao, and S.-Z. Yao, "Symbolic layout compaction under conditional design rules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 2, pp. 476–486, April 1992.

[22] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 87–100, 1984.

[23] T. Burd, "Very low power cell library," University of California, Berkeley, Tech. Rep., 1995.

[24] J. Zhu and W. S. Mong, "Specification of non-functional intellectual property components," in *Proceedings of the Design Automation and Test Conference in Europe (DATE)*, Munick, Germany, March 2002.

[25] "Mosis website," http://www.mosis.org.

[26] M. Berkelaar, "LP_SOLVE: Linear programming code," ftp://ftp.es.ele.tue.nl/pub/lp_solve/.

[27] M. A.Riepe and K. A.Sakallah, "The edge-based design rule model revisited," *ACM transactions on design automation of electronic system*, vol. 3, pp. 463–486, July 1998.

[28] E. R. Ca, "MOSIS scalable CMOS (SCMOS) design rules (revision 7.2) the MOSIS service USC/ISI," 4676 Admiralty Way, Marina del Rey, CA 90292-6695. [Online]. Available: citeceer.ist.psu.edu/484772.html

Jianwen Zhu (M'00) received the B.S degree in electrical engineering from the Tsinghua University, Beijing, China, the M.S and Ph.D. degree in computer science from the University of California, Irvine, USA, in 1993, 1996 and 1999 respectively. He is currently an assistant professor in the Department of Electrical and Computer Engineering, University of Toronto, Canada. He is the coauthor of the book SpecC: Specification Language and Methodology (Kluwer-Academic, 2000). His research interest includes hardware/software codesign, high-level synthesis for high-performance circuits and retargetable compilation.

Qianying Tang Qianying Tang is currently an undergraduate student in Electrical and Computer Engineering in University of Toronto, Toronto, Ont. and will receive her B.A.Sc. degree in May 2005. She was awarded the S. Sedra Outstanding Student Reward during her studies, and was supported by the NSERC Undergraduate Research Award for her research work. Her research interests are in areas of Computer aided Design for integrated circuit and in areas of Computational and Statistical Learning.