Optimal Slack-Driven Block Shaping Algorithm in Fixed-Outline Floorplanning *

Jackey Z. Yan
Placement Technology Group
Cadence Design Systems
San Jose, CA 95134 USA
zyan@cadence.com

Chris Chu
Department of ECE
Iowa State University
Ames, IA 50010 USA
cnchu@iastate.edu

ABSTRACT

This paper presents an efficient, scalable and optimal slack-driven shaping algorithm for soft blocks in non-slicing floorplan. The proposed algorithm is called *SDS*. Different from all previous approaches, *SDS* is specifically formulated for fixed-outline floorplanning. Given a fixed upper bound on the layout width, *SDS* minimizes the layout height by only shaping the soft blocks in the design. Iteratively, *SDS* shapes some soft blocks to minimize the layout height, with the guarantee that the layout width would not exceed the given upper bound. Rather than using some simple heuristic as in previous work, the amount of change on each block is determined by systematically distributing the global total amount of available slack to individual block. During the whole shaping process, the layout height is monotonically reducing, and eventually converges to an optimal solution. We also propose two optimality conditions to check the optimality of a shaping solution. To validate the efficiency and effectiveness of *SDS*, comprehensive experiments are conducted on MCNC and HB benchmarks. Compared with previous work, *SDS* is able to achieve the best experimental result with significantly faster runtime.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuits, Design Aids**]: Layout

General Terms

Algorithms, Design, Performance

Keywords

Block Shaping, Fixed-Outline Floorplan, Physical Design

1. INTRODUCTION

Floorplanning is a very crucial step in modern VLSI designs. A good floorplan solution has a positive impact on the placement, routing and even manufacturing. In floorplanning step, a design contains two types of blocks, hard and soft. A hard block is a circuit block with both area and aspect ratio [1] fixed, while a soft one has fixed area, yet flexible aspect ratio. Shaping such soft blocks plays an important role in determining the top-level spatial structure of a chip, because the shapes of blocks directly affect the packing quality and the area of a floorplan. However, due to the ever-increasing complexity of ICs, the problem of shaping soft blocks is not trivial.

1.1 Previous Work

In slicing floorplan, researchers proposed various soft-block shaping algorithms. Stockmeyer [1] proposed the shape curve representation used to capture different shapes of a subfloorplan. Based on the shape curve, it is straightforward to choose the floorplan solution with the minimum cost, e.g., minimum floorplan area. In [2], Zimmermann extended the shape curve representation by considering both slicing line directions when combining two blocks. Yan *et al.* [3] generalized the notion of slicing tree [4] and extended the shape curve operations. Consequently, one shape curve captures significantly more shaping and floorplan solutions.

Different from slicing floorplan, the problem of shaping soft blocks to optimize the floorplan area in non-slicing floorplan is much more complicated. Both Pan *et al.* [5] and Wang *et al.* [6] tried to extend the slicing tree and shape curve representations to handle non-slicing floorplan. But their extensions are limited to some specific non-slicing structures. Instead of using the shape curve, Kang *et al.* [7] adopted the bounded sliceline grid structure [8] and proposed a greedy heuristic algorithm to select different shapes for each soft block, so that total floorplan area was minimized. Moh *et al.* [9] formulated the shaping problem as a geometric programming and searched for the optimal floorplan area using standard convex optimization. Following the same framework as in [9], Murata *et al.* [10] improved the algorithm efficiency via reducing the number of variables and functions. But the algorithm still took a long time to find a good solution. In [11], Young *et al.* showed that the shaping problem for minimum floorplan area can be solved optimally by Lagrangian relaxation technique. Lin *et al.* [12] changed the problem objective to minimizing the half perimeter of a floorplan, and solved it optimally by the min-cost flow and trust region method.

All of the above shaping algorithms for non-slicing floorplan were targeting at classical floorplanning, i.e., minimizing the floorplan area. But, in the nanometer scale era classical floorplanning cannot satisfy the requirements of hierarchical design. In contrast, fixed-outline floorplanning [13] enabling the hierarchical framework is preferred by modern ASIC designs. In [14], Adya *et al.* introduced the notion of *slack* in floorplanning, and proposed a slack-based algorithm to shape the soft blocks. Such shaping algorithm was applied inside an annealing-based fixed-outline floorplanner. There are two problems with this shaping algorithm: 1) It is a simple greedy heuristic, in which each time every soft block is shaped to use up all its slack

*This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998.

ISPD'12, March 25–28, 2012, Napa, California, USA.

[1] The *aspect ratio* is defined as the ratio of the block height to the block width.

in one direction. Thus, the resulting solution has no optimality guarantee; 2) It is not formulated for fixed-outline floorplanning. The fixed-outline constraint is simply considered as a penalty term in the cost function of annealing. Therefore, in non-slicing floorplan it is necessary to design an efficient and optimal shaping algorithm that is specifically formulated for fixed-outline floorplanning.

1.2 Our Contributions

This work presents an efficient, scalable and optimal slack-driven shaping (*SDS*) algorithm for soft blocks in non-slicing floorplan. *SDS* is specifically formulated for fixed-outline floorplanning. Given a fixed upper bound on the layout width, *SDS* minimizes the layout height by only shaping the soft blocks in the design. If such upper bound is set as the width of a predefined fixed outline, *SDS* is capable of optimizing the area for fixed-outline floorplanning. As far as we know, none of previous work in non-slicing floorplan considers the fixed-outline constraint in the problem formulation. In *SDS*, soft blocks are shaped iteratively. At each iteration, we only shape some of the soft blocks to minimize the layout height, with the guarantee that the layout width would not exceed the given upper bound. The amount of change on each block is determined by systematically distributing the global total amount of available slack to individual block. During the whole shaping process, the layout height is monotonically reducing, and eventually converges to an optimal solution. Note that in [14] without a global slack distribution, all soft blocks are shaped greedily and independently by some simple heuristic. In their work, both the layout height and width are reduced in one shot (i.e., not iteratively) and the solution is stuck at a local minimum.

Essentially, we have three main contributions.

- **Basic Slack-Driven Shaping:** The basic slack-driven shaping algorithm is a very simple shaping technique. Iteratively, it identifies some soft blocks, and shapes them by a slack-based shaping scheme. The algorithm stops when there is no identified soft block. The runtime complexity in each iteration is linear time. The basic *SDS* can achieve an optimal layout height for most cases.
- **Optimality Conditions:** To check the optimality of the shaping solution returned by the basic *SDS*, two optimality conditions are proposed. We prove that if either one of the two conditions is satisfied, the solution returned by the basic *SDS* is optimal.
- **Slack-Driven Shaping (SDS):** Based on the basic *SDS* and the optimality conditions, we propose the slack-driven shaping algorithm. In *SDS*, a geometric programming method is applied to improve the non-optimal solution produced by the basic *SDS*. *SDS* always returns an optimal shaping solution.

To show the efficiency of *SDS*, we compare it with the two shaping algorithms in [11] and [12] on MCNC benchmarks. Even though both of them claim their algorithms can achieve the optimal solution, experimental results show that *SDS* consistently generates better solution on each circuit with significantly faster runtime. On average *SDS* is 253× and 33× faster than [11] and [12] respectively, to produce solutions of similar quality. We also run *SDS* on HB benchmarks. Experimental results show that on average after 6%, 10%, 22% and 47% of the total iterations, the layout height is within 10%, 5%, 1% and 0.1% difference from the optimal solution, respectively.

The rest of this paper is organized as follows. Section 2 describes the problem formulation. Section 3 introduces the basic slack-driven shaping algorithm. Section 4 discusses the optimality of a shaping solution and presents two optimality conditions. Section 5 describes the algorithm flow of *SDS*. Experimental results are presented in Section 6. Finally, this paper ends with a conclusion and the direction of future work.

2. PROBLEM FORMULATION

In the design, suppose we are given n blocks. Each block i ($1 \leq i \leq n$) has fixed area A_i. Let w_i and h_i denote the width and height of block i respectively. The range of w_i and h_i are given as $W_i^{min} \leq w_i \leq W_i^{max}$ and $H_i^{min} \leq h_i \leq H_i^{max}$. If block i is a hard block, then $W_i^{min} = W_i^{max}$ and $H_i^{min} = H_i^{max}$. Let x_i and y_i denote the x and y coordinates of the bottom-left corner of block i respectively. To model the geometric relationship among the blocks, we use the horizontal and vertical constraint graphs G_h and G_v, where the vertices represent the blocks and the edges between two vertices represent the non-overlapping constraints between the two corresponding blocks. In G_h, we add two dummy vertices 0 and $n+1$ that represent the left-most and right-most boundary of the layout respectively. Similarly, in G_v we add two dummy vertices 0 and $n+1$ that represent the bottom-most and top-most boundary of the layout respectively. The area of the dummy vertices is 0. We have $x_0 = 0$ and $y_0 = 0$. Vertices 0 and $n+1$ are defined as the source and the sink in the graphs respectively. Thus, in both G_h and G_v, we add one edge from the source to each vertex that does not have any incoming edge, and add one edge from each vertex that does not have any outgoing edge to the sink.

In our problem formulation, we assume the constraint graphs G_h and G_v are given. Given an upper bound on the layout width as W, we want to minimize the layout height y_{n+1} by only shaping the soft blocks in the design, such that the layout width $x_{n+1} \leq W$. Such problem can be mathematically formulated as follows:

PROBLEM 1. **Height Minimization with Fixed Upper-Bound Width**

$$
\begin{array}{lcl}
\text{Minimize} & y_{n+1} & \\
\text{subject to} & x_{n+1} \leq W & \\
& x_j \geq x_i + w_i, & \forall (i,j) \in G_h \\
& y_j \geq y_i + h_i, & \forall (i,j) \in G_v \\
& W_i^{min} \leq w_i \leq W_i^{max}, & 1 \leq i \leq n \\
& H_i^{min} \leq h_i \leq H_i^{max}, & 1 \leq i \leq n \\
& w_i h_i = A_i, & 1 \leq i \leq n \\
& x_0 = 0 & \\
& y_0 = 0 &
\end{array}
$$

It is clear that if W is set as the width of a predefined fixed outline, Problem 1 can be applied in fixed-outline floorplanning.

3. BASIC SLACK-DRIVEN SHAPING

In this section, we present the basic slack-driven shaping algorithm, which solves Problem 1 optimally for most cases.

First of all, we introduce some notations used in the discussion. Given the constraint graphs and the shape of the blocks, we can pack the blocks to four lines, i.e., the left (LL), right (RL), bottom (BL) and top (TL) lines. LL, RL, BL and TL are set as "$x = 0$", "$x = W$", "$y = 0$" and "$y = y_{n+1}$", respectively. Let Δ_{x_i} denote the difference of x_i when packing block i to RL and LL. Similarly, Δ_{y_i} denotes the difference of y_i when packing block i to TL and BL. For block i ($1 \leq i \leq n$), the horizontal slack s_i^h and vertical slack s_i^v are calculated as follows:

$$s_i^h = max(0, \Delta_{x_i}), \quad s_i^v = max(0, \Delta_{y_i})$$

In G_h, given any path [2] from the source to the sink, if for all blocks on this path, their horizontal slacks are equal to zero, then we define such path as a horizontal critical path (HCP). The length of one HCP is the summation of the width of blocks on this path. Similarly, we can define the vertical critical path (VCP) and the length of one VCP is the summation of the height of blocks on this path. Note that,

[2] By default, all paths in this paper are from the source to the sink in the constraint graph.

because we set RL as the "$x = W$" line, if $x_{n+1} < W$, then there is no HCP in G_h.

The algorithm flow of the basic *SDS* is simple and straightforward. The soft blocks are shaped iteratively. At each iteration, we apply the following two operations:

1. Shape the soft blocks on all VCPs by increasing the width and decreasing the height. This reduces the lengths of the VCPs.
2. Shape the soft blocks on all HCPs by decreasing the width and increasing the height. This reduces the lengths of the HCPs.

The purpose of the first operation is to minimize the layout height y_{n+1} by decreasing the lengths of all VCPs. As mentioned previously, if $x_{n+1} < W$ then there is no HCP. Thus, the second operation is applied only if $x_{n+1} = W$. This operation seems to be unnecessary, yet actually is critical for the proof of the optimality conditions. The purpose of this operation will be explained in Section 4. At each iteration, we first globally distribute the total amount of slack reduction to the soft blocks, and then locally shape each individual soft block on the critical paths based on the allocated amount of slack reduction. The algorithm stops when we cannot find any soft block to shape on the critical paths. During the whole shaping process, the layout height y_{n+1} is monotonically decreasing and thus the algorithm always converges.

In the following subsections, we first identify which soft blocks to be shaped (which we called *target soft blocks*) at each iteration. Secondly, we mathematically derive the shaping scheme on the target soft blocks. Finally, we present the algorithm flow of the basic *SDS*.

3.1 Target Soft Blocks

For a given shaping solution, the set of n blocks can be divided into the following seven disjoint subsets ($1 \le i \le n$).

$$\begin{cases} \text{Subset I} & = \{i \text{ is hard}\} \\ \text{Subset II} & = \{i \text{ is soft}\} \cap \{s_i^h \neq 0, s_i^v \neq 0\} \\ \text{Subset III} & = \{i \text{ is soft}\} \cap \{s_i^h = 0, s_i^v = 0\} \\ \text{Subset IV} & = \{i \text{ is soft}\} \cap \{s_i^h \neq 0, s_i^v = 0\} \cap \{w_i \neq W_i^{max}\} \\ \text{Subset V} & = \{i \text{ is soft}\} \cap \{s_i^h \neq 0, s_i^v = 0\} \cap \{w_i = W_i^{max}\} \\ \text{Subset VI} & = \{i \text{ is soft}\} \cap \{s_i^h = 0, s_i^v \neq 0\} \cap \{h_i \neq H_i^{max}\} \\ \text{Subset VII} & = \{i \text{ is soft}\} \cap \{s_i^h = 0, s_i^v \neq 0\} \cap \{h_i = H_i^{max}\} \end{cases}$$

Based on the definitions of critical paths, we have the following observations [3].

OBSERVATION 1. *If block $i \in$ subset II, then i is not on any HCP nor VCP.*

OBSERVATION 2. *If block $i \in$ subset III, then i is on both HCP and VCP, i.e., at the intersection of some HCP and some VCP.*

OBSERVATION 3. *If block $i \in$ subset IV or V, then i is on some VCP but not on any HCP.*

OBSERVATION 4. *If block $i \in$ subset VI or VII, then i is on some HCP but not on any VCP.*

As mentioned previously, y_{n+1} can be minimized by reducing the height of the soft blocks on the vertical critical paths, and such block-height reduction will result in a decrease on the horizontal slacks of those soft blocks. From the above observations, only soft blocks in subsets III, IV and V are on the vertical critical paths. However, for block $i \in$ subset III, $s_i^h = 0$, which means its horizontal slack cannot be further reduced. And for block $i \in$ subset V, $w_i = W_i^{max}$, which means its height cannot be further reduced. As a result, to minimize y_{n+1} we can only shape blocks in subset IV. Similarly, we conclude that whenever we need to reduce x_{n+1} we can only shape blocks in subset VI. For the hard blocks in subset I, they cannot be shaped anyway.

Therefore, the target soft blocks are the blocks in subsets IV and VI.

[3] Please refer to Theorem 1 in [14] for the proof of these observations.

3.2 Shaping Scheme

Let δ_i^h denote the amount of increase on w_i for block $i \in$ subset IV, and δ_i^v denote the amount of increase on h_i for block $i \in$ subset VI. In the remaining part of this subsection, we present the shaping scheme to shape the target soft block $i \in$ subset IV by setting δ_i^h. Similar shaping scheme is applied to shape the target soft block $i \in$ subset VI by setting δ_i^v. By default, all blocks mentioned in the following part are referring to the target soft blocks in subset IV.

We use "$i \in p$" to denote that block i is on a path p in G_h. Suppose the maximum horizontal slack over all blocks on p is s_{max}^p. Basically, s_{max}^p gives us a budget on the total amount of increase on the block width along this path. If $\sum_{i \in p} \delta_i^h > s_{max}^p$, then after shaping, we have $x_{n+1} > W$, which violates the constraint "$x_{n+1} \le W$". So we have to set δ_i^h accordingly, such that $\sum_{i \in p} \delta_i^h \le s_{max}^p$ for all p in G_h.

To determine the value of δ_i^h, we first define a distribution ratio α_i^p ($\alpha_i^p \ge 0$) for block $i \in p$. We assign the value of α_i^p, such that

$$\sum_{i \in p} \alpha_i^p = 1$$

LEMMA 1. *For any path p in G_h, we have*

$$\sum_{i \in p} \alpha_i^p s_i^h \le s_{max}^p$$

PROOF. Because $s_{max}^p = \text{MAX}_{i \in p}(s_i^h)$, this lemma can be proved as follows:

$$\sum_{i \in p} \alpha_i^p s_i^h \le \sum_{i \in p} \alpha_i^p s_{max}^p = s_{max}^p \sum_{i \in p} \alpha_i^p = s_{max}^p$$

□

Based on Lemma 1, for a single path p, it is obvious that if $\delta_i^h \le \alpha_i^p s_i^h$ ($i \in p$), then we can guarantee $\sum_{i \in p} \delta_i^h \le s_{max}^p$.

More generally, if there are multiple paths going through block i ($1 \le i \le n$), then δ_i^h needs to satisfy the following inequality:

$$\delta_i^h \le \alpha_i^p s_i^h, \forall p \in P_i^h \tag{1}$$

where P_i^h is the set of paths in G_h going through block i. Inequality 1 is equivalent to the following inequality.

$$\delta_i^h \le \underset{p \in P_i^h}{\text{MIN}}(\alpha_i^p) s_i^h \tag{2}$$

Essentially, Inequality 2 gives an upper bound on the amount of increase on w_i for block $i \in$ subset IV.

For block $i \in p$, the distribution ratio is set as follows:

$$\alpha_i^p = \begin{cases} 0 & i \text{ is the source or the sink} \\ \frac{W_i^{max} - w_i}{\sum_{k \in p}(W_k^{max} - w_k)} & \text{otherwise} \end{cases} \tag{3}$$

The insight is that if we allocate more slack reduction to the blocks that have potentially more room to be shaped, the algorithm will converge faster. And we allocate zero amount of slack reduction to the dummy blocks at the source and the sink in G_h. Based on Equation 3, Inequality 2 can be rewritten as follows ($1 \le i \le n$):

$$\delta_i^h \le \frac{(W_i^{max} - w_i) s_i^h}{\underset{p \in P_i^h}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k))} \tag{4}$$

From the above inequality, to calculate the upper bound of δ_i^h, we need to obtain the value of three terms, $(W_i^{max} - w_i)$, s_i^h and $\text{MAX}_{p \in P_i^h}(\sum_{k \in p}(W_k^{max} - w_k))$. The first term can be obtained in constant time. Using the longest path algorithm, s_i^h for all i can be calculated in linear time. A trivial approach to calculate the third term is via traversing each path in G_h. This takes exponential time, which is not practical. Therefore, we propose a dynamic programming (DP) based approach that only takes linear time to calculate the third term.

In G_h, suppose vertex i ($0 \leq i \leq n+1$) has in-coming edges coming from the vertices in the set V_i^{in}, and out-going edges going to the vertices in the set V_i^{out}. Let P_i^{in} denote the set of paths that start at the source and end at vertex i in G_h, and P_i^{out} denote the set of paths that start at vertex i and end at the sink in G_h. For the source of G_h, we have $V_0^{in} = \phi$ and $P_0^{in} = \phi$. For the sink of G_h, we have $V_{n+1}^{out} = \phi$ and $P_{n+1}^{out} = \phi$. We notice that $\text{MAX}_{p \in P_i^h}(\sum_{k \in p}(W_k^{max} - w_k))$ can be calculated recursively by the following equations.

$$\underset{p \in P_0^{in}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) = 0$$

$$\underset{p \in P_{n+1}^{out}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) = 0$$

$$\underset{p \in P_i^{in}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) = \underset{j \in V_i^{in}}{\text{MAX}}(\underset{p \in P_j^{in}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k))) + (W_i^{max} - w_i) \quad (5)$$

$$\underset{p \in P_i^{out}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) = \underset{j \in V_i^{out}}{\text{MAX}}(\underset{p \in P_j^{out}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) + (W_i^{max} - w_i) \quad (6)$$

$$\underset{p \in P_i^{h}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) = \underset{p \in P_i^{in}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) + \underset{p \in P_i^{out}}{\text{MAX}}(\sum_{k \in p}(W_k^{max} - w_k)) - (W_i^{max} - w_i) \quad (7)$$

Based on the equations above, the DP-based approach can be applied step by step as follows ($1 \leq i \leq n$):

1. We apply topological sort algorithm on G_h.
2. We scan the sorted vertices from the source to the sink, and calculate $\text{MAX}_{p \in P_i^{in}}(\sum_{k \in p}(W_k^{max} - w_k))$ by Equation 5.
3. We scan the sorted vertices from the sink to the source, and calculate $\text{MAX}_{p \in P_i^{out}}(\sum_{k \in p}(W_k^{max} - w_k))$ by Equation 6.
4. $\text{MIN}_{p \in P_i^h}(\sum_{k \in p}(W_k^{max} - w_k))$ is obtained by Equation 7.

It is clear that by the DP-based approach, the whole process of calculating the upper bound of δ_i^h for all i takes linear time.

3.3 Flow of Basic Slack-Driven Shaping

The algorithm flow of basic slack-driven shaping is shown in Figure 1. In this flow, for each block i in the design, we set its initial width $w_i = W_i^{min}$ ($1 \leq i \leq n$). Based on the input G_h, G_v and initial block shape, we can calculate an initial value of x_{n+1}. If such initial value is already bigger than W, then Problem 1 is not feasible.

At each iteration we set $\delta_j^v = \beta \times \text{MIN}_{p \in P_j^v}(\alpha_j^p) s_j^v$ for target soft block $j \in$ subset VI. By default, $\beta = 100\%$, which means we set δ_j^v exactly at its upper bound. One potential problem with this strategy is that the layout height y_{n+1} may remain the same, i.e., never decreasing. This is because after one iteration of shaping, the length of some non-critical vertical path increases, and consequently its length may become equivalent to the length of the VCP in the previous iteration. Accidentally, such scenario may keep cycling forever, and thus y_{n+1} would never decrease. This issue can be solved, as long as δ_j^v is set less than its upper bound. In this way, after one iteration of shaping we can guarantee that the length of the VCP will be shorter than the one in the previous iteration. Theoretically, any $\beta < 100\%$ can break the cycling scenario and guarantee the algorithm convergence. But because in *SDS* any amount of change that is less than 0.0001 would be masked by numerical error, we can actually calculate a lower bound of β, and obtain its range as follows.

$$\frac{0.01}{\text{MIN}_{p \in P_j^v}(\alpha_j^p) s_j^v}\% < \beta < 100\%$$

In the implementation, whenever we detect that y_{n+1} does not change for more than two iterations, we will set $\beta = 90\%$ for the next iteration. For δ_j^h, we always set it at its upper bound.

Because in each iteration the total increase on width or height of the target soft blocks would not exceed the budget, we can guarantee that the layout would not be outside of the four lines after shaping. As iteratively we set TL to the updated "$y = y_{n+1}$" line, y_{n+1} will be monotonically decreasing during the whole shaping process. Different from TL, because we set RL to the fixed "$x = W$" line, during the shaping process x_{n+1} may be bouncing i.e., sometimes increasing and sometimes decreasing, yet always no more than W. The shaping process stops when there is no target soft block.

Basic Slack-Driven Shaping
Input: $w_i = W_i^{min}$ ($\forall 1 \leq i \leq n$); G_h and G_v; upper-bound width W.
Output: optimized y_{n+1}, w_i and h_i.
Begin
1. Set LL, BL and RL to "$x = 0$", "$y = 0$" and "$x = W$".
2. Pack blocks to LL and use longest path algorithm to get x_{n+1}.
3. If $x_{n+1} > W$,
4. Return no feasible solution.
5. Else,
6. Repeat
7. Pack blocks to BL and use longest path algorithm to get y_{n+1}.
8. Set TL to "$y = y_{n+1}$".
9. Pack blocks to LL, RL and TL, respectively.
10. Calculate s_i^h and s_i^v.
11. Find target soft blocks.
12. If there are target soft blocks,
13. $\forall j \in$ subset IV, increase w_j by $\delta_j^h = \text{MIN}_{p \in P_j^h}(\alpha_j^p) s_j^h$;
14. $\forall j \in$ subset VI, increase h_j by $\delta_j^v = \beta \times \text{MIN}_{p \in P_j^v}(\alpha_j^p) s_j^v$.
15. Until there is no target soft block.
End

Figure 1: Flow of basic slack-driven shaping.

4. OPTIMALITY CONDITIONS

For most cases, in the basic *SDS* the layout height y_{n+1} will converge to an optimal solution of Problem 1. However, sometimes the solution may be non-optimal as the one shown in Figure 2-(a). The layout in Figure 2-(a) contains four soft blocks 1, 2, 3 and 4, where $A_i = 4$, $W_i^{min} = 1$ and $W_i^{max} = 4$ ($1 \leq i \leq 4$). The given upper-bound width $W = 5$. In the layout, $w_1 = w_3 = 4$ and $w_2 = w_4 = 1$. There is no target soft block on any one of the four critical paths (i.e., two HCPs and two VCPs), so the basic *SDS* returns $y_{n+1} = 5$. But the optimal layout height should be 3.2, when $w_1 = w_2 = w_3 = w_4 = 2.5$ as shown in Figure 2-(b). In this section, we will look into this issue and present the optimality conditions for the shaping solution returned by the basic *SDS*.

Let L represent a shaping solution generated by the basic *SDS* in Figure 1. All proof in this section are established based on the fact that the *only* remaining soft blocks that could be shaped to possibly

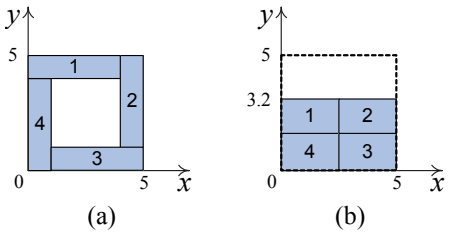


Figure 2: Example of a non-optimal solution from the basic *SDS*.

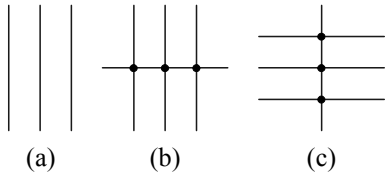


Figure 3: Examples of three optimal cases in L.

improve L are the ones in subset III. This is because L is the solution returned by the basic *SDS* and in L there is no soft block that belongs to subsets IV nor VI any more. This is also why we need apply the second shaping operation in the basic *SDS*. Its purpose is *not* reducing x_{n+1}, but eliminating the soft blocks in subset VI. From Observation 2, we know that any block in subset III is always at the intersection of some HCP and some VCP. Therefore, to improve L it is sufficient to just consider shaping the intersection soft blocks between the HCPs and VCPs.

Before we present the optimal conditions, we define two concepts.

- **Hard Critical Path:** If all intersection blocks on one critical path are hard blocks, then this path is a *hard* critical path.
- **Soft Critical Path:** A critical path, which is not hard, is a *soft* critical path.

LEMMA 2. *If there exists one hard VCP in L, then L is optimal.*

PROOF. Since all intersection blocks on this VCP are hard blocks, there is no soft block that can be shaped to possibly improve this VCP. Therefore, L is optimal. □

LEMMA 3. *If there exists at most one soft HCP or at most one soft VCP in L, then L is optimal.*

PROOF. As proved in Lemma 2, if there exists one hard VCP in L, then L is optimal. So in the following proof we assume there is no hard VCP in L. For any hard HCP, as all intersection blocks on it are hard blocks, we cannot change its length by shaping those intersection blocks anyway. So we can basically ignore all hard HCPs in this proof.

Suppose L is non-optimal. We should be able to identify some soft blocks and shape them to improve L. As mentioned previously, it is sufficient to just consider shaping the intersection soft blocks. If there is at most one soft HCP or at most one soft VCP, there are only three possible cases in L. (As we set TL as the "$y = y_{n+1}$" line, there is always at least one VCP in L.)

1. **There is no soft HCP, and there is one or multiple soft VCPs** (e.g., Figure 3-(a))
   In this case, L does not contain any intersection soft blocks.
2. **There is one soft HCP, and there is one or multiple soft VCPs** (e.g., Figure 3-(b))
   In this case, L has one or multiple intersection soft blocks. Given any one of such blocks, say i. To improve L, h_i has to be reduced. But this increases the length of the soft HCP, which violates "$x_{n+1} \leq W$" constraint. So, none of the blocks can be shaped to improve L.
3. **There is one or multiple soft HCPs, and there is one soft VCP** (e.g., Figure 3-(c))
   In this case, L has one or multiple intersection soft blocks. Given any one of such blocks, say i. Similarly, it can be proved that "$x_{n+1} \leq W$" constraint will be violated, if h_i is reduced. So, none of the blocks can be shaped to improve L.

As a result, for all the above cases, we cannot find any soft block that could be shaped to possibly improve L. This means our assumption is not correct. Therefore, L is optimal. □

5. FLOW OF SLACK-DRIVEN SHAPING

Using the conditions presented in Lemmas 2 and 3, we can determine the optimality of the output solution from the basic *SDS*. Therefore, based on the algorithm flow in Figure 1, we propose the slack-driven shaping algorithm shown in Figure 4. *SDS* always returns an optimal solution for Problem 1.

Slack-Driven Shaping
Input: $w_i = W_i^{min}$ ($\forall 1 \leq i \leq n$); G_h and G_v; upper-bound width W.
Output: optimal y_{n+1}, w_i and h_i.
Begin
Lines 1 – 14 are the same as the ones in Figure 1.
15. Else,
16. If Lemma 2 or 3 is satisfied,
17. L is optimal.
18. Else,
19. Improve L by a single step of geometric programming.
20. If no optimal solution is obtained,
21. Go to Line 7.
22. Else,
23. L is optimal.
24. Until L is optimal.
End

Figure 4: Flow of slack-driven shaping.

The differences between *SDS* and the basic version are starting from line 15 in Figure 4. When there is not target soft block, instead of terminating the algorithm, *SDS* will first check the optimality of L, and if it is not optimal, L will be improved via geometric programming. The algorithm stops when an optimal solution is obtained.

As mentioned previously, if the solution L generated by the basic *SDS* is not optimal, we only need to shape the intersection soft blocks to improve L. In this way, the problem now becomes shaping the intersection blocks to minimize the layout height y_{n+1} subject to layout width constraint "$x_{n+1} \leq W$". In other words, it is basically the same as Problem 1, except that we only need to shape a smaller number of soft blocks (i.e., the intersection soft blocks). This problem is a geometric program. It can be transformed into a convex problem and solved optimally by any convex optimization technique. However, considering the runtime, we don't need to rely on geometric programming to converge to an optimal solution. We just run one step of some iterative convex optimization technique (e.g., deepest descent) to improve L. Then we can go back to line 7, and applied the basic *SDS* again. It is clear that *SDS* always converges to the optimal solution because as long as the solution is not optimal, the layout height will be improved.

In modern VLSI designs, the usage of Intellectual Property (IP) and embedded memory blocks becomes more and more popular. As a result, a design usually contains tens or even hundreds of big hard

macros, i.e., hard blocks. Due to their big sizes, after applying the basic *SDS* most likely they are at the intersections of horizontal and vertical critical paths. Moreover, in our experiments we observe that there is always no more than one soft HCP or VCP in the solution returned by the basic *SDS*. Consequently, we never need to apply the geometric programming method in our experiments. Therefore, we believe that for most designs the basic slack-driven shaping algorithm is sufficient to achieve an optimal solution for Problem 1.

6. EXPERIMENTAL RESULTS

This section presents the experimental results. All experiments are run on a Linux server with AMD Opteron 2.59 GHz CPU and 16GB memory. We use two sets of benchmarks, MCNC [11] and HB [15]. For each circuit, the corresponding input G_h and G_v are provided by a floorplanner. The range of the aspect ratio for any soft block in the circuit is set to $[\frac{1}{3}, 3]$.

After the input data is read, *SDS* will set the initial width of each soft block at its minimum width. In *SDS*, if the amount of change on the width or height of any soft block is less than 0.0001, we would not shape such block because any change smaller than that would be masked by numerical error. Such numerical error, which is unavoidable, comes from the truncation of an infinite real number so as to make the computation possible and practical.

6.1 Experiments on MCNC Benchmarks

Using the MCNC benchmarks we compare *SDS* with the two shaping algorithms in [11] and [12]. All blocks in these circuits are soft blocks. The source code of [11] and [12] are obtained from the authors.

In fact, these three shaping algorithms *cannot* be directly compared, because their optimization objectives are all different:

- [11] is minimizing the layout area $x_{n+1}y_{n+1}$;
- [12] is minimizing the layout half perimeter $x_{n+1} + y_{n+1}$;
- *SDS* is minimizing the layout height y_{n+1}, s.t. $x_{n+1} \leq W$.

Still, to make some meaningful comparisons as best as we can, we setup the experiment in the following way.

- We conduct two groups of experiments: 1) *SDS v.s.* [11]; 2) *SDS v.s.* [12].
- As the circuit size are all very small, to do some meaningful comparison on the runtime, in each group we run both shaping algorithms 1000 times with the same input data.
- For group 1, we run [11] first, and use the returned final width from [11] as the input upper-bound width W for *SDS*. For group 2, similar procedure is applied.
- For groups 1 and 2, we compare the final results based on [11]'s and [12]'s objectives respectively.

Table 2 shows the results on group 1. The column "$ws(\%)$" gives the white space percentage over the total block area in the final layout. For all five circuits *SDS* achieves significantly better results on the floorplan area. On average, *SDS* achieves $394\times$ smaller white space and $23\times$ faster runtime than [11]. In the last column, we report the runtime *SDS* takes to converge to a solution that is better than [11]. To just get a slightly better solution than [11], on average *SDS* uses $253\times$ faster runtime. As pointed out by [12], [11] does not transform the problem into a convex problem before applying Lagrangian relaxation. Hence, algorithm [11] may not converge to an optimal solution.

Table 3 shows the results on group 2. The authors claims the shaping algorithm in [12] can find the optimal half perimeter on the floorplan layout. But, for all five circuits *SDS* gets consistently better half perimeter than [12], with on average $10\times$ faster runtime. Again, in the last column, we report the runtime *SDS* takes to converge to a solution that is better than [12]. To just get a slightly better solution than [12], on average *SDS* uses $33\times$ faster runtime. We believe algorithm [12] stops earlier, before it converges to an optimal solution.

Table 1: Comparison on runtime complexity.

Algorithm	Runtime Complexity
Young *et al.* [11]	$\mathcal{O}(m^3 + km^2)$
Lin *et al.* [12]	$\mathcal{O}(kn^2 mlog(nC))$
Basic *SDS*	$\mathcal{O}(km)$

(k is the total number of iterations, n is the total number of blocks in the design, m is the total number of edges in G_h and G_v, and C is the biggest input cost.)

From the runtime reported in Tables 2 and 3, it is clear that as the circuit size increases, *SDS* scales much better than both [11] and [12]. In Table 1, we list the runtime complexities among the three shaping algorithms. As in our experiments, it is never necessary to apply the geometric programming method in *SDS*, we list the runtime complexity of the basic *SDS* in Table 1. Obviously, the basic *SDS* has the best scalability.

6.2 Experiments on HB Benchmarks

This subsection presents the experimental results of *SDS* on HB benchmarks. As both algorithms [11] and [12] crashed on this set of circuits, we cannot compare *SDS* with them. The HB benchmarks contain both hard and soft blocks ranging from 500 to 2000 (see Table 4 for details).

For each test case, we set the upper-bound width W as the square root of 110% of the total block area in the corresponding circuit. Let Y denote the optimal y_{n+1} *SDS* converges to. The results are shown in Table 4. The "Convergence Time" column lists the total runtime of the whole convergence process. The "Total #.Iterations" column shows the total number of iteration *SDS* takes to converge to Y. For fixed-outline floorplanning, *SDS* can actually stop early as long as the solution is within the fixed outline. So in the subsequent four columns, we also report the number of iterations when $\frac{y_{n+1}-Y}{Y}$ starts to be less than 10%, 5%, 1% and 0.1%, respectively. The average total convergence time is 1.18 second. *SDS* takes average 1901 iterations to converge to Y. The four percentage numbers in the last row shows that on average after 6%, 10%, 22% and 47% of the total number of iterations, *SDS* converges to the layout height that is within 10%, 5%, 1% and 0.1% difference from Y, respectively. In order to show the convergence process more intuitively, we plot out the convergence graphs of y_{n+1} for four circuits in Figures 5(a)-5(d). In the figures, the four blue arrows point to the four points when y_{n+1} becomes less than 10%, 5%, 1% and 0.1% difference from Y, respectively.

Finally, we have four remarks on *SDS*.

1. As *SDS* sets the initial width of each soft block at its minimal width, such initial floorplan is actually considered as the *worse* start point for *SDS*. This means if any better initial shape is given, *SDS* will converge to Y even faster.
2. In our experiments, we never notice that the solution generated by the basic *SDS* contains more than one soft HCP or VCP. So if ignoring the numerical error mentioned previously, *SDS* obtains the optimal layout height for all circuits in the experiments simply by the basic *SDS*.
3. The experimental results show that after around $\frac{1}{5}$ of the total iterations, the difference between y_{n+1} and Y is already considered quite small, i.e., less than 1%. So in practice if it is not necessary to obtain an optimal solution, we can basically

Table 2: Comparison with [11] on MCNC Benchmarks († shows the total shaping time of 1000 runs and does not count I/O time).

Circuit	#. Soft Blocks	Young *et al.* [11] ws (%)	Final Width	Final Height	Shaping Time† (s)	*SDS* ws (%)	Final Width	Final Height	Upper-Bound Width W	Shaping Time† (s)	*SDS* stops when result is better than [11] ws (%)	Time† (s)
apte	9	4.66	195.088	258.647	0.12	**0.00**	195.0880	246.6147	195.0880	0.26	2.85	0.01
xerox	10	7.69	173.323	120.945	0.08	**0.01**	173.3229	111.6599	173.3230	0.23	6.46	0.01
hp	11	10.94	83.951	120.604	0.08	**1.70**	83.9509	109.2605	83.9510	0.10	7.96	0.02
ami33a	33	8.70	126.391	100.202	22.13	**0.44**	126.3909	91.7830	126.3910	**3.97**	8.67	0.28
ami49a	49	10.42	144.821	273.19	203.80	**1.11**	144.8210	247.4727	144.8210	**1.86**	9.74	0.20
Normalized		393.919			23.351	**1.000**				**1.000**	313.980	0.092

Table 3: Comparison with [12] on MCNC Benchmarks († shows the total shaping time of 1000 runs and does not count I/O time).

Circuit	#. Soft Blocks	Lin *et al.* [12] Half Perimeter	Final Width	Final Height	Shaping Time† (s)	*SDS* Half Perimeter	Final Width	Final Height	Upper-Bound Width W	Shaping Time† (s)	*SDS* stops when result is better than [12] Half Peri.	Time† (s)
apte	9	439.319	219.814	219.505	0.99	**439.3050**	219.8139	219.4911	219.8140	**0.59**	439.1794	0.01
xerox	10	278.502	138.034	140.468	1.24	**278.3197**	138.0339	140.2858	138.0340	**0.30**	278.4883	0.12
hp	11	190.3848	95.2213	95.1635	1.51	**190.2435**	95.2212	95.0223	95.2213	**0.17**	190.3826	0.10
ami33a	33	215.965	107.993	107.972	34.85	**215.7108**	107.9930	107.7178	107.9930	**1.45**	215.9577	0.46
ami49a	49	377.857	193.598	184.259	26.75	**377.5254**	193.5980	183.9274	193.5980	**2.20**	377.8242	0.44
Normalized		1.001			10.177	**1.000**				**1.000**	1.001	0.304

set a threshold value on the amount of change on y_{n+1} as the stopping criterion. For example, if the amount of change on y_{n+1} is less than 1% during the last 10 iterations, then *SDS* will stop.

4. Like all other shaping algorithms, *SDS* is *not* a floorplanning algorithm. To implement a fixed-outline floorplanner based on *SDS*, for example, we can simply integrate *SDS* into a similar annealing-based framework as the one in [14]. In each annealing loop, the input constraint graphs are sent to *SDS*, and *SDS* stops once the solution is within the fixed outline. The annealing process keeps refining the constraint graphs so as to optimize the various floorplanning objectives (e.g., wirelength, routability [16] [17], timing, etc.) in the cost function.

7. CONCLUSION AND FUTURE WORK

This work proposed an efficient, scalable and optimal slack-driven shaping algorithm for soft blocks in non-slicing floorplan. Unlike previous work, we formulate the problem in a way, such that it can be applied for fixed-outline floorplanning. For all cases in our experiments, the basic *SDS* is sufficient to obtain an optimal solution. Both the efficiency and effectiveness of *SDS* have been validated by comprehensive experimental results and rigorous theoretical analysis.

Due to the page limit, we have to reserve some problems on *SDS* as the motivation of future work, which includes: 1) To use the *duality gap* of Problem 1 as a better stopping criterion, because it indicates an upper-bound of the gap between the intermediate and optimal shaping solutions; 2) To propose a more scalable algorithm as a substitution of the geometric programing method in Figure 4; 3) To extend *SDS* to handle classical floorplanning. Also, because of the similarity between the *slack* in floorplanning and static timing analysis (STA), we believe *SDS* can be modified and applied on buffer/wire sizing for timing optimization.

Acknowledgment

The authors would like to thank Prof. H. Zhou from Northwestern University for providing us the source code of algorithms [11] and [12].

8. REFERENCES

[1] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57:91–101, May/June 1983.

[2] G. Zimmermann. A new area and shape function estimation technique for VLSI layouts. In *Proc. DAC*, pages 60–65, 1988.

[3] J. Z. Yan and C. Chu. DeFer: Deferred decision making enabled fixed-outline floorplanning algorithm. *IEEE Trans. on Computer-Aided Design*, 43(3):367–381, March 2010.

[4] R. H. J. M. Otten. Efficient floorplan optimization. In *Proc. ICCD*, pages 499–502, 1983.

[5] P. Pan and C. L. Liu. Area minimization for floorplans. *IEEE Trans. on Computer-Aided Design*, 14(1):129–132, January 1995.

[6] T. C. Wang and D. F. Wong. Optimal floorplan area optimization. *IEEE Trans. on Computer-Aided Design*, 11(8):992–1001, August 1992.

[7] M. Kang and W. W. M. Dai. General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure. In *Proc. ASP-DAC*, pages 265–270, 1997.

[8] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on BSG-structure and IC layout applications. In *Proc. ICCAD*, pages 484–491, 1996.

[9] T. S. Moh, T. S. Chang, and S. L. Hakimi. Globally optimal floorplanning for a layout problem. *IEEE Trans. on Circuits and Systems I*, 43:713–720, September 1996.

[10] H. Murata and E. S. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. In *Proc. ISPD*, pages 167–172, 1998.

[11] F. Y. Young, C. C. N. Chu, W. S. Luk, and Y. C. Wong. Handling soft modules in general non-slicing floorplan using Lagrangian relaxation. *IEEE Trans. on Computer-Aided Design*, 20(5):687–692, May 2001.

[12] C. Lin, H. Zhou, and C. Chu. A revisit to floorplan optimization by Lagrangian relaxation. In *Proc. ICCAD*, pages 164–171, 2006.

[13] A. B. Kahng. Classical floorplanning harmful? In *Proc. ISPD*, pages 207–213, 2000.

[14] S. N. Adya and I. L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. on VLSI Systems*, 11(6):1120–1135, December 2003.

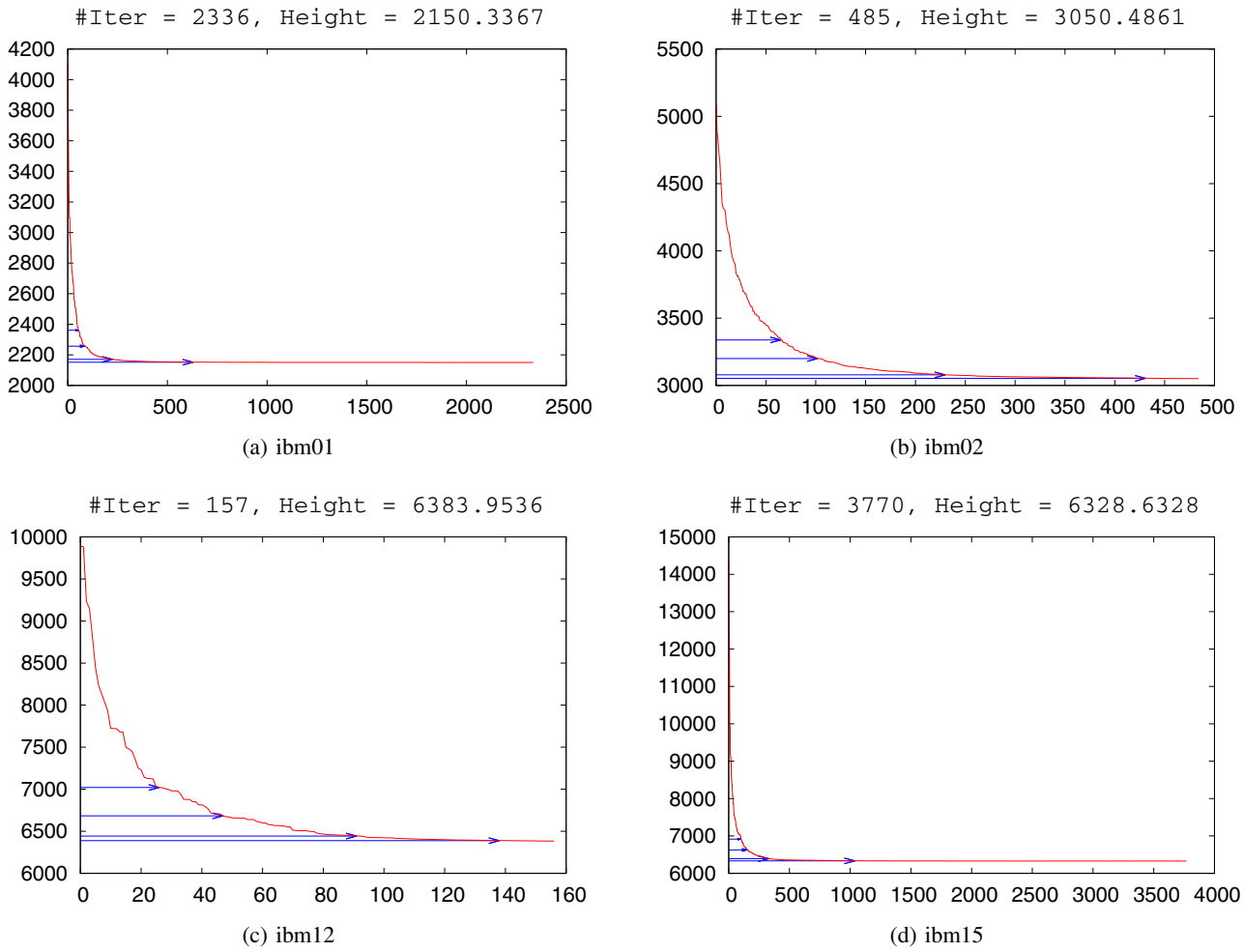
[15] J. Cong, M. Romesis, and J. R. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Proc. ASP-DAC*, pages 1119–1122, 2005.

[16] Y. Zhang and C. Chu. CROP: Fast and effective congestion refinement of placement. In *Proc. ICCAD*, pages 344–350, 2009.

[17] Y. Zhang and C. Chu. RegularRoute: An efficient detailed router with regular routing patterns. In *Proc. ISPD*, pages 45–52, 2011.

Table 4: Experimental Results of *SDS* on HB Benchmarks.

Circuit	#.Soft Blocks / #.Hard Blocks	Upper-Bound Width W	Final Width	Final Height (Y)	Convergence Time (s)	Total #.Iterations	#.Iterations when $\frac{y_{n+1}-Y}{Y}$ becomes $< 10\%$	$< 5\%$	$< 1\%$	$< 0.1\%$
ibm01	665 / 246	2161.9005	2161.9003	2150.3366	0.82	2336	54	85	225	629
ibm02	1200 / 271	3057.4816	3056.6026	3050.4862	0.40	485	65	102	230	431
ibm03	999 / 290	3298.2255	3298.2228	3305.6953	0.36	565	62	97	231	456
ibm04	1289 / 295	3204.7658	3204.7656	3179.9406	3.65	3564	53	87	271	1076
ibm05	564 / 0	2222.8426	2222.8424	2104.4136	0.29	1456	102	142	279	522
ibm06	571 / 178	3069.5289	3068.5232	2988.6851	0.14	500	58	105	265	419
ibm07	829 / 291	3615.5698	3615.5696	3599.6710	1.86	3966	63	114	269	1210
ibm08	968 / 301	3855.1451	3855.1449	3822.5919	0.42	690	75	111	232	545
ibm09	860 / 253	4401.0232	4401.0231	4317.0274	1.20	2512	50	82	234	687
ibm10	809 / 786	7247.6365	7246.7511	7221.0778	0.49	472	28	56	162	377
ibm11	1124 / 373	4844.2184	4844.2183	4820.8615	0.60	654	64	96	253	509
ibm12	582 / 651	6391.9946	6388.6978	6383.9537	0.10	157	26	47	91	138
ibm13	530 / 424	5262.6052	5262.6050	5204.0326	1.03	2695	52	78	244	753
ibm14	1021 / 614	5634.2142	5634.2140	5850.1577	2.88	2622	75	109	237	634
ibm15	1019 / 393	6353.8948	6353.8947	6328.6329	2.94	3770	100	152	331	1039
ibm16	633 / 458	7622.8724	7622.8723	7563.6297	0.95	2038	41	65	193	520
ibm17	682 / 760	6827.7756	6827.7754	6870.9049	1.78	2200	46	67	139	389
ibm18	658 / 285	6101.0694	6101.0692	6050.4116	1.35	3544	57	82	185	454
Average					**1.18**	**1901**	**5.9%**	**9.6%**	**22.3%**	**47.3%**



(a) ibm01 (b) ibm02

(c) ibm12 (d) ibm15

Figure 5: Layout-height convergence graphs for circuits ibm01, ibm02, ibm12 and ibm15. (*x-axis denotes the iteration number and y-axis denotes the layout height.*)