SDS: An Optimal Slack-Driven Block Shaping Algorithm for Fixed-Outline Floorplanning

Jackey Z. Yan and Chris Chu, *Fellow, IEEE*

***Abstract*—This paper presents an efficient, scalable, and optimal slack-driven shaping algorithm for soft blocks in nonslicing floorplan. The proposed algorithm is called SDS. 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 monotonically reduces and eventually converges to an optimal solution. Two optimality conditions are presented to check the optimality of a shaping solution for fixed-outline floorplanning. In practice, to terminate the process of convergence early, we propose two different stopping criteria. We also extend SDS to handle other floorplanning problems, e.g., classical floorplanning. To validate the efficiency and effectiveness of SDS, comprehensive experiments are conducted on MCNC and HB benchmarks. Compared with previous work, SDS achieves the best experimental result with a significantly faster runtime.**

***Index Terms*—Block shaping, fixed-outline floorplanning, very large scale integration physical design.**

I. Introduction

FLOORPLANNING is a very crucial step in modern very large scale integration (VLSI) designs. A good floorplan solution has a positive impact on the placement, routing, and even manufacturing. In the 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. But, due to the ever-increasing complexity of integrated circuits (ICs), the problem of shaping soft blocks is not trivial.

Manuscript received June 12, 2012; revised August 3, 2012 and September 22, 2012; accepted November 2, 2012. Date of current version January 18, 2013. This work was supported in part by the IBM Faculty Award and the NSF under Grant CCF-0540998. This paper was recommended by Associate Editor C.-K. Koh.

J. Z. Yan is with the Placement Technology Group, Cadence Design Systems, Inc., San Jose, CA 95134 USA (e-mail: zyan@cadence.com).

C. Chu is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010 USA (e-mail: cnchu@iastate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2228304

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

A. 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. Zimmermann [2] 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 nonslicing floorplan is much more complicated. Sutanthavibul *et al.* [5] formulated the shaping problem as a mixed integer linear program. Because it is quite time consuming to solve it optimally, the original problem was divided into multiple smaller subproblems each of which contained 10–12 blocks. As a result, each sub-problem was solved optimally, but the optimality of the solution on the original problem was not guaranteed. Both Pan *et al.* [6] and Wang *et al.* [7] tried to extend the slicing tree and shape curve representations to handle nonslicing floorplan. But their extensions are limited to some specific nonslicing structures. Instead of using the shape curve, Kang *et al.* [8] adopted the bounded sliceline grid structure [9] and proposed a greedy heuristic algorithm to select different shapes for each soft block, so that total floorplan area was minimized. Moh *et al.* [10] 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 [10], Murata *et al.* [11] 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. Young *et al.* [12] showed that the shaping problem for minimum floorplan area can be solved optimally by Lagrangian relaxation technique. Lin *et al.* [13] 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 nonslicing 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 [14] enabling the hierarchical framework is preferred by modern ASIC

designs. Adya *et al.* [15] 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 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 a penalty term in the cost function of annealing.

Considering the fixed-outline constraint, Lin *et al.* [16] determined the block shape using second-order cone programming. But in their problem formulation, the aspect ratio constraint on each block was ignored. They had to rely on a postprocessing step to re-shape the block in order to satisfy the aspect ratio constraint. So, in nonslicing floorplan it is necessary to design an efficient and optimal shaping algorithm that is specifically formulated for fixed-outline floorplanning and considers the aspect ratio constraints.

B. Our Contributions

This paper presents an efficient, scalable and optimal slack-driven shaping (SDS) algorithm[2] for soft blocks in nonslicing 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, while satisfying the aspect ratio constraint on each block. 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 nonslicing floorplan is able to solve this problem optimally. 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 [15] 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 five main contributions.

- *Basic Slack-Driven Shaping:* The basic SDS 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:* 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 nonoptimal solution produced by the basic SDS. SDS always returns an optimal shaping solution.
- *Stopping Criteria:* To early terminate the process of convergence in SDS, two stopping criteria are proposed. The first one is a simple heuristic based on the amount of change on the shaping solution for the last a few iterations. The second one is nonheuristic, and it guarantees that difference between the resulting and optimal shaping solutions is within the user-specified error margin.
- *Extension of SDS:* We extend SDS to handle two other floorplanning problems: 1) to minimize the layout width with the given layout height upper bound, and 2) to minimize the layout area, i.e., classical floorplanning.

To show the efficiency of SDS, we compare it with the two shaping algorithms in [12] and [13] on MCNC benchmarks. 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 [12] and [13], respectively, to produce solutions of similar quality. For fixed-outline floorplanning, 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. For classical floorplanning, using HB benchmarks we show that SDS can significantly reduce the layout area by just shaping the soft blocks.

The rest of this paper is organized as follows. Section II describes the problem formulation. Section III introduces the basic SDS algorithm. Section IV discusses the optimality of a shaping solution and presents two optimality conditions. Section V describes the algorithm flow of SDS. Section VI presents the stopping criteria. Section VII describes the extension of SDS on other floorplanning problems. Experimental results are presented in Section VIII. Finally, this paper ends with a conclusion and the direction of future work.

II. 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 nonoverlapping 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.

[2]A preliminary version of SDS was presented in [17].

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 $x_{n+1} \leq W$. This problem can be mathematically formulated as follows.

Problem 1: Height minimization with fixed upper-bound width.

$$\begin{aligned}
\text{Minimize} \quad & y_{n+1} \\
\text{subject to} \quad & x_{n+1} \leq W \\
& x_j \geq x_i + w_i, && \forall (i,j) \in G_h \\
& y_j \geq y_i + h_i, && \forall (i,j) \in G_v \\
& W_i^{\min} \leq w_i \leq W_i^{\max}, && 1 \leq i \leq n \\
& H_i^{\min} \leq h_i \leq H_i^{\max}, && 1 \leq i \leq n \\
& w_i h_i = A_i, && 1 \leq i \leq n \\
& x_0 = y_0 = 0,\ w_0 = h_0 = 0
\end{aligned}$$

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

III. Basic Slack-Driven Shaping

This section presents the basic SDS algorithm that solves Problem 1 optimally in our experiments.

First of all, we introduce some notations used in the discussion. Given the constraint graphs and the shape of the blocks, without violating the nonoverlapping constraints, 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 between the two layouts generated by packing blocks to RL and LL, respectively. Similarly, Δ_{y_i} denotes the difference of y_i between the two layouts generated by packing blocks to TL and BL, respectively. 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[3] 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, since 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 IV. 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 sections, 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.

A. Target Soft Blocks

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

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

Based on the definitions of critical paths, we have the following observations.[4]

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

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

[4]Please refer to [15, Theorem 1] for the proof of these observations.

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.

So the target soft blocks are the ones in subsets IV and VI.

B. 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 section, 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 "$x_{n+1} \leq W$". So we have to set δ_i^h accordingly, such that $\sum_{i\in p} \delta_i^h \leq s_{\max}^p$ for all p in G_h.

To determine the value of δ_i^h, we first define a distribution ratio α_i^p ($\alpha_i^p \geq 0$) 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

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

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

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

■

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

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

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

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

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

Essentially, (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:

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

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. We allocate zero amount of slack reduction to the dummy blocks at the source and the sink in G_h. Based on (3), (2) can be rewritten as follows ($1 \leq i \leq n$):

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

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 incoming edges coming from the vertices in the set V_i^{in}, and outgoing 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 $V_0^{\text{in}} = \phi$ and $P_0^{\text{in}} = \phi$. For the sink of G_h, we have $V_{n+1}^{\text{out}} = \phi$ and $P_{n+1}^{\text{out}} = \phi$. 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:

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

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

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

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

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

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 $\text{MAX}_{p\in P_i^{\text{in}}}(\sum_{k\in p}(W_k^{\max} - w_k))$ by (5).
3) We scan the sorted vertices from the sink to the source, and calculate $\text{MAX}_{p\in P_i^{\text{out}}}(\sum_{k\in p}(W_k^{\max} - w_k))$ by (6).
4) $\text{MAX}_{p\in P_i^h}(\sum_{k\in p}(W_k^{\max} - w_k))$ is obtained by (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.

Basic Slack-Driven Shaping
Input: $w_i = W_i^{min}$ ($\forall 1 \le i \le n$); 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

Fig. 1. Flow of basic slack-driven shaping.

C. Flow of Basic Slack-Driven Shaping

The algorithm flow of basic SDS is shown in Fig. 1. In this flow, for each block i in the design, we set its initial width $w_i = W_i^{\min}$ ($1 \le i \le n$). 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 noncritical 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*, as 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.

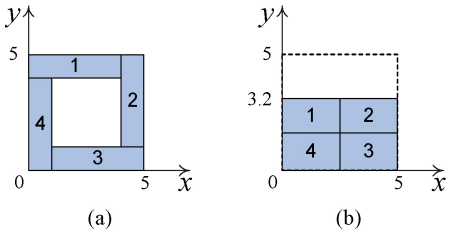

Fig. 2. Example of a nonoptimal solution from the basic SDS. (a) Nonoptimal solution. (b) Optimal solution.

IV. Optimality Conditions

Sometimes, in the basic SDS the layout height y_{n+1} may converge to a nonoptimal solution of Problem 1, as the one case shown in Fig. 2(a). The layout in Fig. 2(a) contains four soft blocks 1, 2, 3, and 4, where $A_i = 4$, $W_i^{\min} = 1$ and $W_i^{\max} = 4$ ($1 \le i \le 4$). 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 Fig. 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 Fig. 1. All proofs in this section are established based on the fact that the only remaining soft blocks that could be shaped to possibly 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. For the soft blocks in the other subsets (i.e., II, V and VII), in Section III we have already explained the reason why they cannot be shaped to improve any shaping solution. 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 such intersection soft blocks.

Before we present the optimality 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.

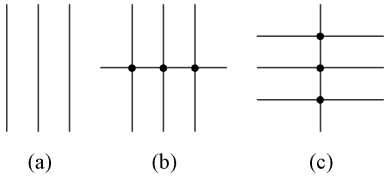

Fig. 3. Examples of three optimal cases in L.

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 nonoptimal. 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., Fig. 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., Fig. 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. Note that, if such constraint is violated, there is no way to reduce the length back to less than or equal to W, because all the blocks that are not intersection blocks on this soft HCP must either be hard blocks or soft blocks in subset VII. 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., Fig. 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. ■

Note that, in both Lemmas 2 and 3 we are proving that at certain point (i.e., when the optimality condition in the lemma is satisfied) there is absolutely nothing that can be done by any algorithm to possibly further improve the solution. This ensures that the shaping solution we obtained at this certain point is a global optimum, rather than a local minimum.

V. 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 Fig. 1, we propose the SDS algorithm shown in Fig. 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

Fig. 4. Flow of slack-driven shaping.

The differences between SDS and the basic version are starting from line 15 in Fig. 4. When there is no 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 do not 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 apply 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 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. We believe that for most designs the basic SDS algorithm is sufficient to achieve an optimal solution for Problem 1.

VI. Stopping Criteria

As shown in Fig. 4, SDS stops when the layout height becomes optimal. This is the default stopping criterion in SDS.

While considering other objectives, e.g., the runtime, it may not be necessary to wait until SDS achieves an optimal solution. Thus, to terminate the algorithm early, in this section we presents two stopping criteria that could be adopted in SDS.

A. One Simple Stopping Criterion

Because during the shaping process in SDS, the layout height is monotonically reducing and eventually converges to an optimal solution, one simple way to set a stopping criterion is to consider a threshold value on the amount of change on the layout height in the last a few iterations. For example, if the amount of change on the layout height is less than 1% during the last 10 iterations, then SDS will stop.

B. Stopping Criterion With Guaranteed Suboptimality Bound

The stopping criterion mentioned previously is a simple heuristic, in which we do not have any guarantee on the error bound between the resulting and optimal shaping solutions. In this section, we present a more sophisticated and nonheuristic stopping criterion. Given a user-specified error margin ϵ ($\epsilon > 0$), we can guarantee that when SDS stops under this criterion, the resulting solution is ϵ-suboptimal.

Let $\mathcal{LD}$ denote the Lagrange dual problem of Problem 1, d denote the value of the objective function from one feasible solution in $\mathcal{LD}$, and Y denote the optimal layout height in Problem 1. Based on the theory of duality [18], the value of the objective function from any feasible solution in $\mathcal{LD}$ yields a lower bound on Y. That is

$$d \leq Y. \tag{8}$$

For any intermediate layout height y' obtained by SDS, we can calculate the gap value defined as $y' - d$. Based on Inequality (8), if such gap value is smaller than ϵ, that is

$$y' - d \leq \epsilon \tag{9}$$

then we can guarantee $y' - Y \leq \epsilon$, which means the solution y' is ϵ-suboptimal.

In the Appendix, we mathematically derive the Lagrange dual problem $\mathcal{LD}$ and explain the calculation of d. So, at each shaping iteration in SDS, using Inequality (9) we can check if the current shaping solution is ϵ-suboptimal or not. If it is, SDS stops. Otherwise, the algorithm moves to the next iteration.

VII. Extension of SDS

In this section, we extend SDS to handle two other floorplanning problems: 1) to minimize the layout width with the given layout height upper bound, and 2) to minimize the layout area, i.e., classical floorplanning.

To solve these two floorplanning problems, we simply just need to change the way we set the right (RL) and top (TL) lines defined in Section III. The soft blocks are still shaped iteratively, and the shaping scheme remains the same as the one presented in Section III-B. The description on the different settings on RL and TL for various floorplanning problems is presented in Table I. In this table, the second row shows the setting we employed for solving Problem 1, while the third and fourth rows show the settings for solving the two extended problems, respectively.

TABLE I
Different Settings on RL and TL for Three Problems and Optimality of Solutions Achieved by SDS

Floorplanning Problem	Right Line RL	Top Line TL	Optimal ?
Minimize y_{n+1}, s.t. $x_{n+1} \leq W$	Set to "$x = W$" permanently	Set to "$y = y_{n+1}$" at each iteration	Yes
Minimize x_{n+1}, s.t. $h_{n+1} \leq H$	Set to "$x = x_{n+1}$" at each iteration	Set to "$y = H$" permanently	Yes
Minimize $x_{n+1}y_{n+1}$	Set to "$x = x_{n+1}$" at each iteration	Set to "$y = y_{n+1}$" at each iteration	No

W and H are the fixed upper bounds for layout width and height, respectively.

TABLE II
Comparison on Runtime Complexity

Algorithm	Runtime Complexity
Young *et al.* [12]	$\mathcal{O}(m^3 + km^2)$
Lin *et al.* [13]	$\mathcal{O}(kn^2 mlog(nC))$
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.

Because of the symmetrical nature of the two problems listed in the second and third rows in Table I, it is obvious that SDS can solve the problem in the third row optimally as well. Regarding the classical floorplanning problem, i.e., the one listed in the fourth row, SDS iteratively minimize both the layout width and height. Different from the fixed-outline floorplanning, both the layout width and height are converging during the whole shaping process. The optimality of the final solution depends on the initial layout, so SDS may not always achieve optimal layout area.

VIII. Experimental Results

This section presents the experimental results. All experiments were run on a Linux server with AMD Opteron 2.59 GHz CPU and 16 GB memory. We use two sets of benchmarks, MCNC [12] and HB [19]. 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 $[\frac{1}{3}, 3]$. 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.

A. Experiments on Fixed-Outline Floorplanning

For fixed-outline floorplanning, after the input data is read, SDS sets the initial width of each soft block to its minimum.

First, we present the experimental results on MCNC benchmarks. We compare SDS with the two shaping algorithms in [12] and [13]. All blocks in these circuits are soft blocks. The source code of [12] and [13] are obtained from the authors. In fact, these three shaping algorithms cannot be directly compared, because they have different optimization objectives:

TABLE III
COMPARISON WITH [12] ON MCNC BENCHMARKS

Circuit	#. Soft Blocks	Young *et al.* [12] 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 [12] 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

† shows the total shaping time of 1000 runs and does not count I/O time.

TABLE IV
COMPARISON WITH [13] ON MCNC BENCHMARKS

Circuit	#. Soft Blocks	Lin *et al.* [13] 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 [13] 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

† shows the total shaping time of 1000 runs and does not count I/O time.

TABLE V
EXPERIMENTAL RESULTS OF *SDS* ON HB BENCHMARKS FOR FIXED-OUTLINE FLOORPLANNING

Circuit	No. of Soft Blocks / No. of Hard Blocks	Upper-Bound Width W	Final Width	Final Height (Y)	Convergence Time (s)	Total No. of Iterations	No. of 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%**

- [12] minimizes the layout area $x_{n+1}y_{n+1}$;
- [13] minimizes the layout half perimeter $x_{n+1} + y_{n+1}$;
- SDS minimizes 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.

1) We conduct two groups of experiments: a) SDS versus [12], and b) SDS versus [13].
2) As the circuits 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.
3) For group 1, we run [12] first, and use the returned final width from [12] as the input upper bound width W for SDS. For group 2, similar procedure is applied.
4) For groups 1 and 2, we compare the final results based on [12]'s and [13]'s objectives, respectively.

We acknowledge that based on the above setup the runtime comparison may not be fair for [12] and [13], because SDS directly makes use of the resulting width value from [12] and [13] as W. However, as mentioned earlier, due to the different optimization objectives for these three algorithms, this is the best we can do. For fixed-outline floorplanning, W is always set as the width of the predefined fixed outline.

Table III 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 [12]. 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 takes $253\times$ faster

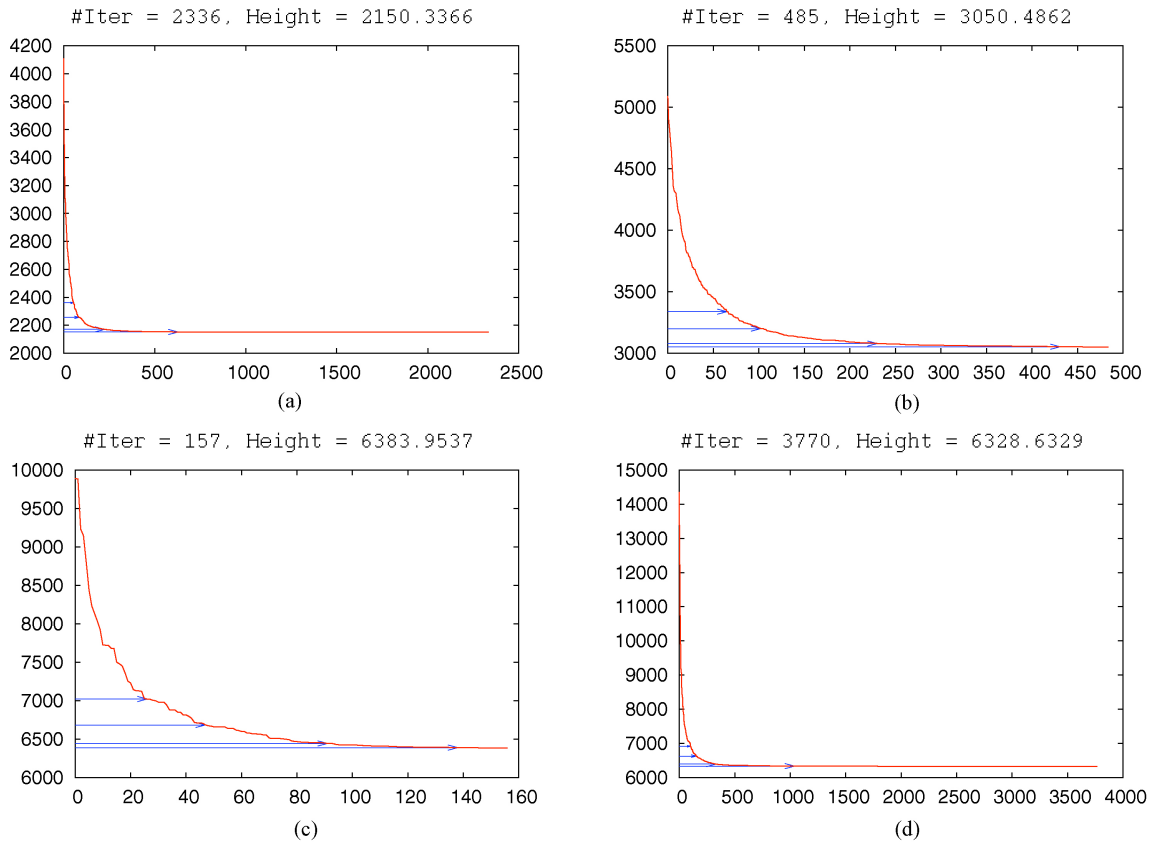

Fig. 5. Layout-height convergence graphs for circuits ibm01, ibm02, ibm12, and ibm15 in fixed-outline floorplanning. (x-axis denotes the iteration number and y-axis denotes the layout height.) (a) ibm01. (b) ibm02. (c) ibm12. (d) ibm15.

runtime. As pointed out by [13], [12] does not transform the problem into a convex problem before applying Lagrangian relaxation. Hence, it may not converge to an optimal solution.

Table IV shows the results on group 2. The authors claim the shaping algorithm in [13] can find the optimal half perimeter on the floorplan layout. But, for all five circuits SDS gets consistently better half perimeter than [13], with on average 10× faster runtime. Again, in the last column, we report the runtime SDS takes to converge to a solution that is better than [13]. To just get a slightly better solution than [13], on average SDS takes 33× faster runtime. We believe algorithm [13] stops earlier, before it converges to an optimal solution. The reason may be because of the numerical error.

From the runtime reported in Tables III and IV, it is clear that as the circuit size increases, SDS scales much better than both [12] and [13]. In Table II, 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 II. Obviously, the basic SDS has the best scalability.

Second, we present the experimental results of SDS on HB benchmarks. As both algorithms [12] and [13] 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 V 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 V. 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. 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. This shows that after around $\frac{1}{5}$ of the total iterations, the difference between y_{n+1} and Y is already quite small, i.e., less than 1%.

To show the convergence process more intuitively, we plot out the convergence graphs of the layout height (i.e., y_{n+1}) for four circuits in Fig. 5(a)–(d). In each figure, 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. For the same four circuits, in Fig. 6(a)–(d) we plot out the value of the layout width (i.e., x_{n+1+}) at each iteration. As we can see, while the layout height converges, the layout width keeps bouncing below the width upper bound. In Fig. 7(a) and (b),

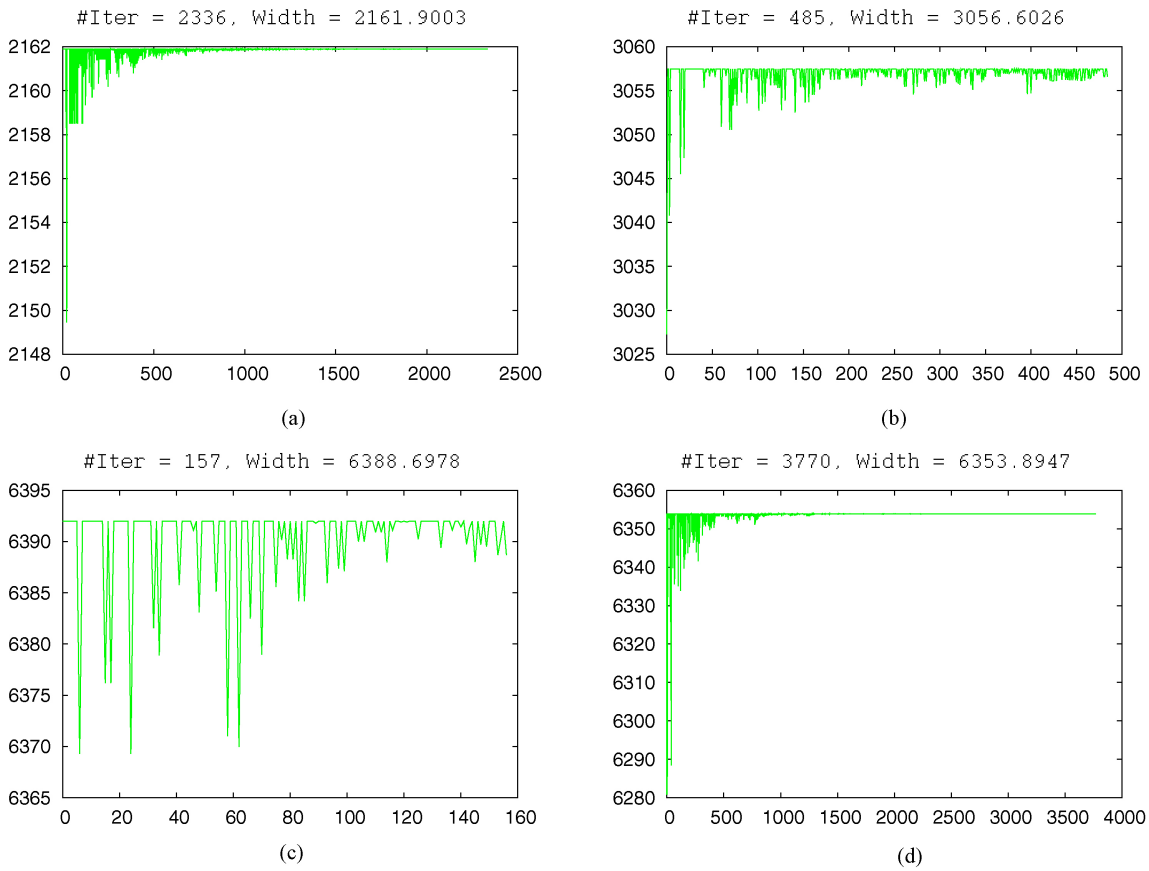

Fig. 6. Layout-width bouncing graphs for circuits ibm01, ibm02, ibm12, and ibm15 in fixed-outline floorplanning. (x-axis denotes the iteration number and y-axis denotes the layout width.) (a) ibm01. (b) ibm02. (c) ibm12. (d) ibm15.

we show the layouts of ibm01 before and after shaping by SDS for fixed-outline floorplanning.

B. Experiments on Classical Floorplanning

This section presents the experimental results of SDS on classical floorplanning. We use the HB benchmarks with the same input constraint graphs as the ones used in the experiments on fixed-outline floorplanning. For classical floorplanning, after the input data is read, SDS will set the initial shape of each soft block as a square.

The detailed experimental results are shown in Table VI. The columns "Initial ws (%)" and "Final ws (%)" give the white space percentages over the total block area in the initial and final layouts, respectively. Comparing these two columns, we can see that the area has been reduced significantly after shaping. In the last column, we also list the white space percentages from the final layouts obtained by SDS in the experiments on fixed-outline floorplanning. For 17 out of 18 circuits, the final layout area obtained in classical floorplanning is better than the one obtained in fixed-outline floorplanning. On the other hand, this also shows that the solution generated by SDS in classical floorplanning may not be optimal, because for ibm16 the resulting layout area in fixed-outline floorplanning is better. On average, the total convergence time over all circuits is 1.37 s, and it takes SDS 2063 iterations to converge.

In Fig. 8(a)–(d), we plot the corresponding convergence graphs of both the layout width and height for four circuits. Different from the ones shown in Figs. 5(a)–(d) and 6(a)–(d), for classical floorplanning both the layout width and height are monotonically reducing during the whole shaping process in SDS. In Fig. 9(a) and (b), we show the layouts of ibm01 before and after shaping by SDS for classical floorplanning (the fixed outlines are still plot as a reference).

C. Several Remarks

Finally, we have three remarks on SDS.

1) For the experiments on fixed-outline floorplanning, 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 simply by the basic SDS.
3) 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 [15]. In each annealing loop, the input constraint graphs are sent to SDS, and SDS stops once

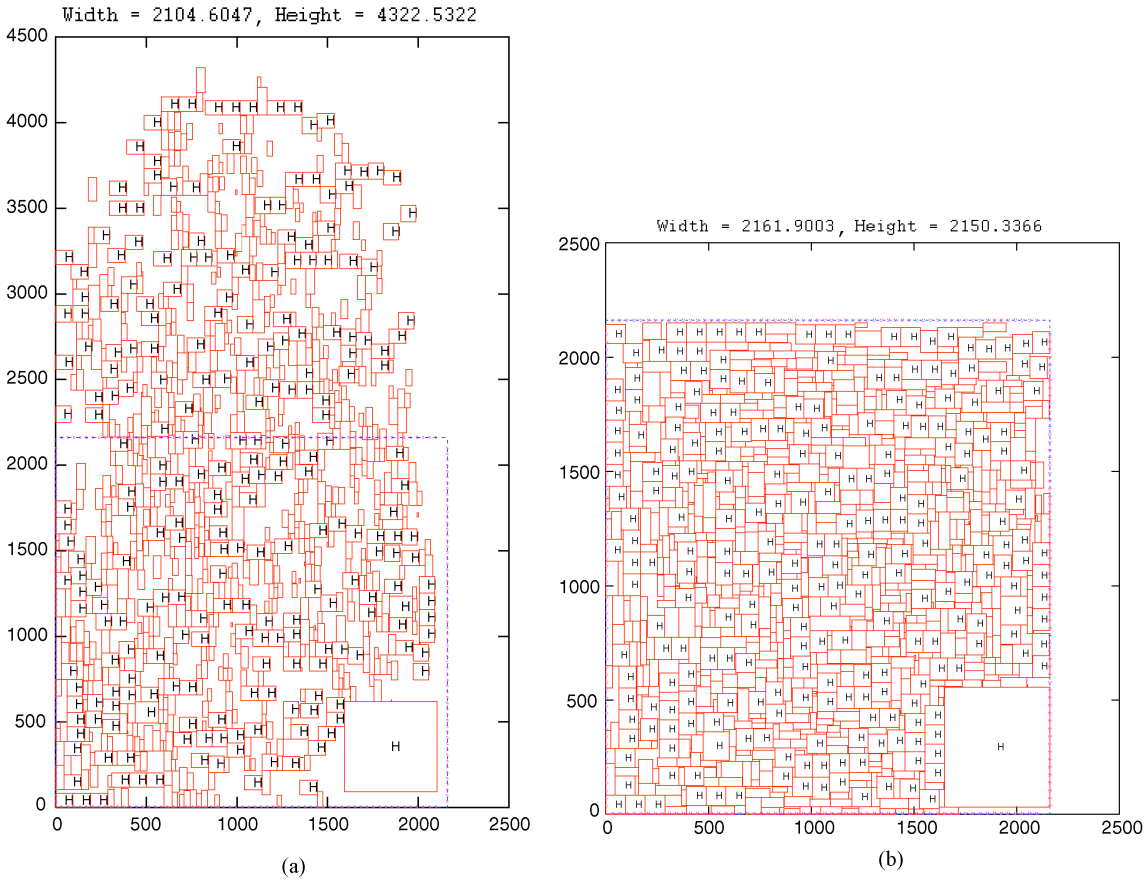

Fig. 7. Initial and final layouts obtained by SDS for circuit ibm01 in fixed-outline floorplanning. (The hard blocks are marked with "H," the blue line plots the fixed outline and the pink line plots the IO pads.) (a) Before shaping. (b) After shaping.

TABLE VI
EXPERIMENTAL RESULTS OF *SDS* ON HB BENCHMARKS FOR CLASSICAL FLOORPLANNING

Circuit	No. of Soft Blocks / No. of Hard Blocks	Initial ws (%)	Final ws (%)	Final Width	Final Height	Convergence Time (s)	Total No. of Iterations	Final ws (%) From Fixed-Outline Solution
ibm01	665 / 246	46.47	**8.61**	2127.1430	2175.7183	0.48	1396	9.02
ibm02	1200 / 271	37.88	**8.31**	3024.5961	3050.4862	0.77	958	9.27
ibm03	999 / 290	41.01	**9.47**	3289.6823	3305.6953	0.49	819	9.71
ibm04	1289 / 295	47.27	**8.79**	3198.0745	3186.5337	3.26	2736	8.80
ibm05	564 / 0	53.87	**4.09**	2155.9597	2162.4859	0.26	1236	4.41
ibm06	571 / 178	40.42	**6.38**	3030.5339	3005.3742	0.48	1738	7.02
ibm07	829 / 291	46.84	**8.57**	3538.6377	3656.3922	0.82	1797	9.10
ibm08	968 / 301	40.59	**8.48**	3844.7472	3822.5919	1.99	3423	8.73
ibm09	860 / 253	43.30	**7.72**	4432.7876	4285.0334	1.62	3520	7.74
ibm10	809 / 786	33.45	**8.98**	7228.2629	7225.5828	2.13	2266	9.16
ibm11	1124 / 373	44.66	**8.65**	4822.4148	4820.8615	3.57	4329	9.06
ibm12	582 / 651	39.39	**9.06**	6369.2802	6383.9537	0.19	331	9.34
ibm13	530 / 424	49.55	**8.47**	5268.7406	5197.3737	0.77	2061	8.48
ibm14	1021 / 614	56.96	**12.76**	5679.4810	5798.1802	2.37	2290	12.84
ibm15	1019 / 393	53.57	**8.74**	6254.8367	6400.5495	3.18	3936	9.14
ibm16	633 / 458	46.65	8.87	7545.9852	7646.9865	0.77	1709	**8.79**
ibm17	682 / 760	55.38	**9.96**	6950.4712	6741.1355	0.94	1256	10.07
ibm18	658 / 285	58.27	**8.47**	6205.6832	5930.3135	0.50	1324	8.74
Average						**1.37**	**2063**	

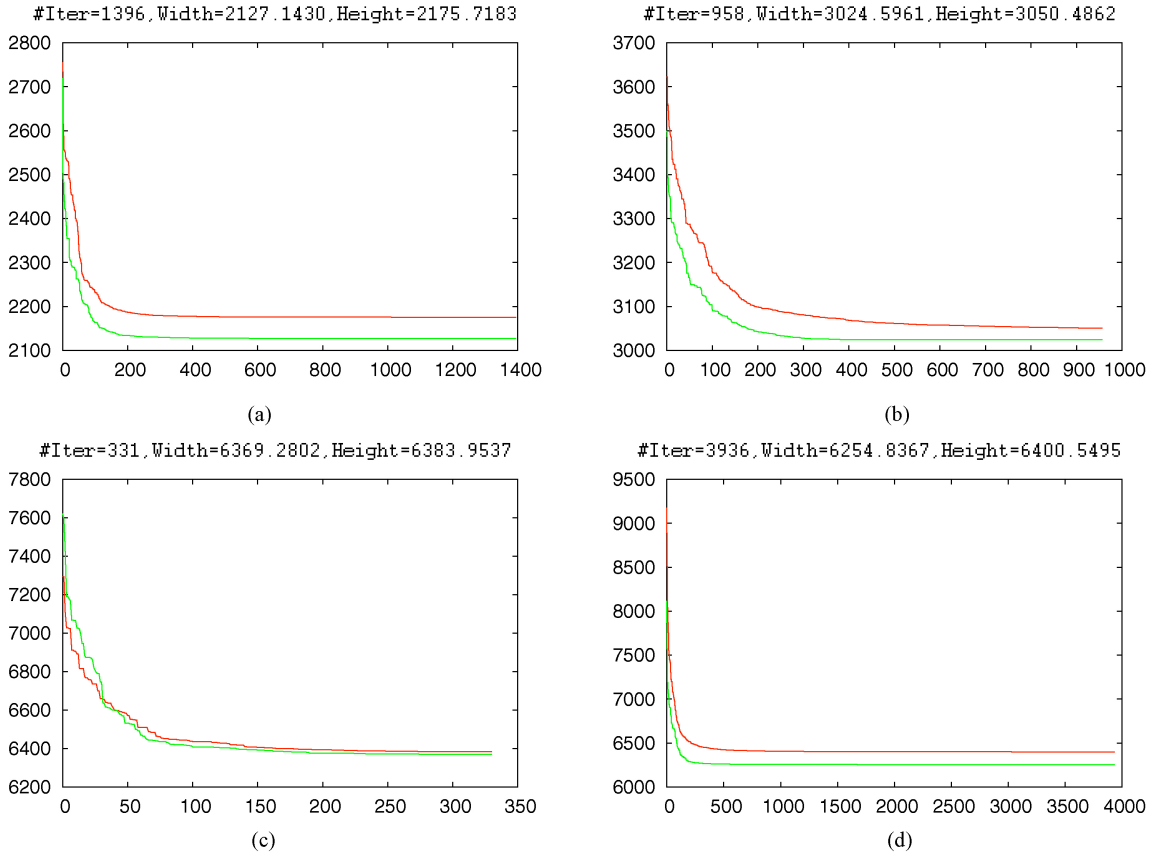

Fig. 8. Layout-width-height convergence graphs for circuits ibm01, ibm02, ibm12 and ibm15 in classical floorplanning. (x-axis denotes the iteration number, the green line plots the layout width and the red line plots the layout height.) (a) ibm01. (b) ibm02. (c) ibm12. (d) ibm15.

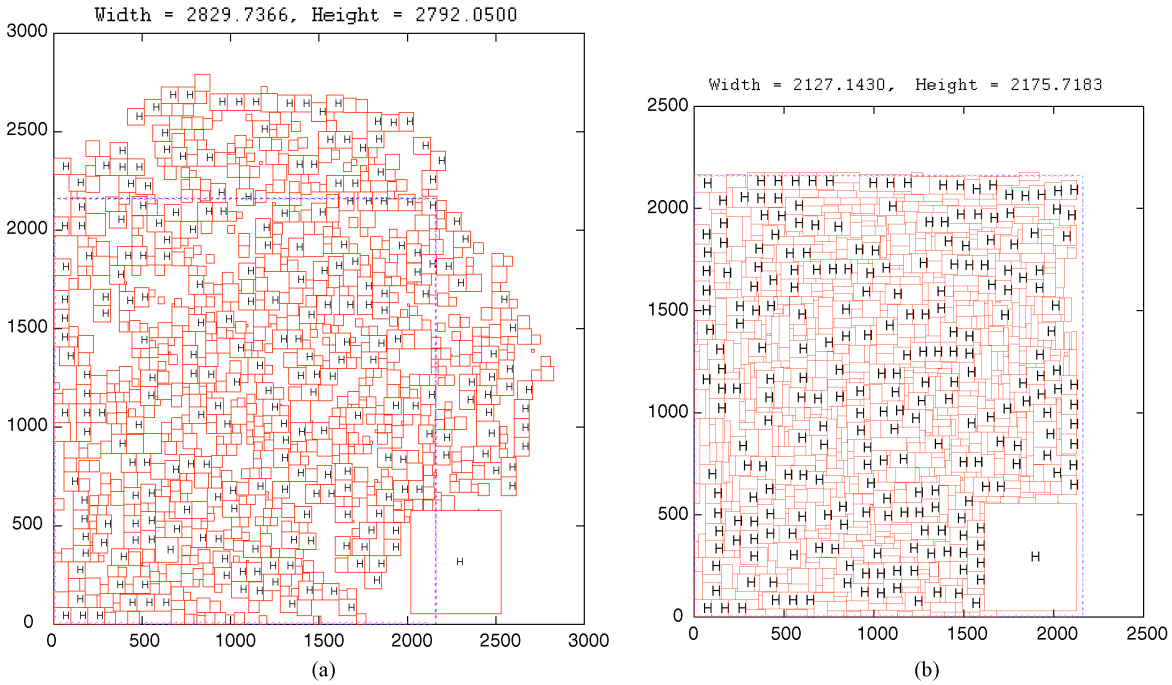

Fig. 9. Initial and final layouts obtained by SDS for circuit ibm01 in classical floorplanning. (The hard blocks are marked with "H," the blue line plots the fixed outline and the pink line plots the IO pads.) (a) Before shaping. (b) After shaping.

the solution is within the fixed outline. The annealing process keeps refining the constraint graphs to optimize the various floorplanning objectives (e.g., wirelength) in the cost function.

IX. Conclusion and Future Work

This paper proposed an efficient, scalable, and optimal SDS algorithm for soft blocks in nonslicing floorplan. We formulated the problem in such a way that it can be applied for fixed-outline floorplanning. We also extended SDS to solve the classical floorplanning. For all cases in our experiments, the basic SDS was sufficient to obtain an optimal solution in fixed-outline floorplanning. Both the efficiency and effectiveness of SDS were validated by comprehensive experimental results and rigorous theoretical analysis.

As the starting point of VLSI physical design, floorplanning plays a very critical role in both prototyping and block implementation. In order to consider multiple cross-domain factors (e.g., timing, routability, power, etc.) in the early stage of IC designs, the core engine of the floorplanner has to be very fast and high quality. The block shaping algorithm is a key component for the modern floorplanner to meet such requirements. This is because the shape of each circuit block determines the top-level spatial structure of a chip, and has direct impact on the circuit performance, e.g., power dissipation, timing closure and routability. Due to the ever-increasing complexity of ICs, the problem of shaping circuit blocks is very complicated. Previously, researchers proposed many sophisticated algorithms to conquer this problem. These algorithms take significant portion of the total runtime inside a floorplanner. However, SDS is very simple, as the core step of the shaping process inside SDS only contains two lines of codes. Yet, the algorithm is able to achieve an optimal solution with one to two orders of magnitude faster than previous state-of-the-art algorithms. If this paper is adopted and implemented in the modern physical optimization tools, it could potentially improve the circuit performance in various respects and significantly reduce the turnaround time for engineers designing new ICs.

In terms of future research direction, we will try to propose a more scalable algorithm as a substitution of the geometric programming method in Fig. 4. Also, because of the similarity between the slack in floorplanning and static timing analysis, we believe SDS can be modified and applied on buffer/wire sizing for timing optimization.

Appendix
Lagrangian Duality of Problem 1

In this section, following the similar procedure as in [12] and [13], we first derive the Lagrange dual problem of Problem 1. Then we explain how to calculate the value of the objective function from one set of dual feasible. To make the presentation easier, we remove dummy vertex 0 and its outgoing edges in G_h. For the vertices connecting to vertex 0 in G_h, we set their $x_i = 0$. The similar changes are made for G_v. As a result, Problem 1 can be rewritten as follows:

Problem 2:

$$\begin{aligned}
\text{Minimize} \quad & y_{n+1} \\
\text{subject to} \quad & x_{n+1} - W \le 0 \\
& x_i - x_j + w_i \le 0, \qquad \forall (i,j) \in G_h \\
& y_i - y_j + \tfrac{A_i}{w_i} \le 0, \qquad \forall (i,j) \in G_v \\
& W_i^{\min} \le w_i \le W_i^{\max}, \qquad 1 \le i \le n.
\end{aligned}$$

To obtain the Lagrangian subproblem of Problem 2, we introduce the nonnegative Lagrangian multiplier for the constraints in Problem 2. Let ϕ denote the multiplier for constraint $x_{n+1} - W \le 0$, $\lambda_{i,j}$ denote the multiplier for constraint $x_i - x_j + w_i \le 0$, and $\mu_{i,j}$ denote the multiplier for constraint $y_i - y_j + \frac{A_i}{w_i} \le 0$. So, we obtain the Lagrangian subproblem as follows.

Problem 3:

$$\begin{aligned}
\text{Minimize} \quad & y_{n+1} + \phi(x_{n+1} - W) \\
& + \sum_{\forall (i,j) \in G_h} \lambda_{i,j}(x_i - x_j + w_i) \\
& + \sum_{\forall (i,j) \in G_v} \mu_{i,j}(y_i - y_j + \frac{A_i}{w_i}) \\
\text{subject to} \quad & W_i^{\min} \le w_i \le W_i^{\max}, \qquad 1 \le i \le n.
\end{aligned}$$

Let $\mathcal{F}$ denote the objective function of Problem 3. $\mathcal{F}$ can be rewritten as follows:

$$\begin{aligned}
\mathcal{F} \;=\; & (\phi - \sum_{\forall (i,n+1) \in G_h} \lambda_{i,n+1}) x_{n+1} \\
& + (1 - \sum_{\forall (i,n+1) \in G_v} \mu_{i,n+1}) y_{n+1} \\
& + \sum_{i=1}^{n} (\sum_{\forall (i,j) \in G_h} \lambda_{i,j} - \sum_{\forall (j,i) \in G_h} \lambda_{j,i}) x_i \\
& + \sum_{i=1}^{n} (\sum_{\forall (i,j) \in G_v} \mu_{i,j} - \sum_{\forall (j,i) \in G_v} \mu_{j,i}) y_i \\
& + \sum_{i=1}^{n} (w_i \sum_{\forall (i,j) \in G_h} \lambda_{i,j} + \frac{A_i}{w_i} \sum_{\forall (i,j) \in G_v} \mu_{i,j}) \\
& - \phi W.
\end{aligned}$$

Because Problem 2 is a convex problem, the Kuhn–Tucker conditions of Problem 2 can be used to simplify $\mathcal{F}$ and the Lagrange dual problem of Problem 2. After applying the Kuhn–Tucker conditions, we obtain the Lagrange dual problem $\mathcal{LD}$ of Problem 2 follows:

$$\text{Maximize} \quad \sum_{i=1}^{n} (w_i \sum_{\forall (i,j) \in G_h} \lambda_{i,j} + \frac{A_i}{w_i} \sum_{\forall (i,j) \in G_v} \mu_{i,j}) - \phi W$$

$$\text{where} \quad w_i = \min\Big(W_i^{\max}, \max(W_i^{min}, \sqrt{\tfrac{A_i \sum_{\forall (i,j) \in G_v} \mu_{i,j}}{\sum_{\forall (i,j) \in G_h} \lambda_{i,j}}}) \Big) \tag{10}$$

$$\text{subject to} \quad \sum_{\forall (i,j) \in G_h} \lambda_{i,j} = \sum_{\forall (j,i) \in G_h} \lambda_{j,i} \tag{11}$$

$$\sum_{\forall (i,j) \in G_v} \mu_{i,j} = \sum_{\forall (j,i) \in G_v} \mu_{j,i} \tag{12}$$

$$\sum_{\forall (i,n+1) \in G_h} \lambda_{i,n+1} = \phi \tag{13}$$

$$\sum_{\forall(i,n+1)\in G_v} \mu_{i,n+1} = 1 \tag{14}$$

$$\lambda \geq 0, \mu \geq 0, \phi \geq 0. \tag{15}$$

In $\mathcal{LD}$, (10)–(14) are derived from the Kuhn–Tucker conditions.

From (11)–(15), it is not difficult to find one set of feasible Lagrange dual variables (λ, μ, ϕ) by linear time. Once (λ, μ, ϕ) are available, we can get the value d of the objective function in $\mathcal{LD}$. Note that, because any d yields a lower bound on the optimal layout height of Problem 1, it is not necessary to solve $\mathcal{LD}$ optimally and find the optimal d.

ACKNOWLEDGMENT

The authors would like to thank Prof. H. Zhou, Northwestern University, for providing them with the source code of algorithms [12] and [13]. They are also grateful to the anonymous reviewers from ICCAD, ISPD, and TCAD for their helpful suggestions and valuable comments on this work.

REFERENCES

[1] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Inform. Control*, vol. 57, pp. 91–101, May–Jun. 1983.
[2] G. Zimmermann, "A new area and shape function estimation technique for VLSI layouts," in *Proc. DAC*, 1988, pp. 60–65.
[3] J. Z. Yan and C. Chu, "Defer: Deferred decision making enabled fixed-outline floorplanning algorithm," *IEEE Trans. Comput.-Aided Des.*, vol. 43, no. 3, pp. 367–381, Mar. 2010.
[4] R. H. J. M. Otten, "Efficient floorplan optimization," in *Proc. ICCD*, 1983, pp. 499–502.
[5] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen, "An analytical approach to floorplan design and optimization," *IEEE Trans. Comput.-Aided Des.*, vol. 10, no. 6, pp. 761–769, Jun. 1991.
[6] P. Pan and C. L. Liu, "Area minimization for floorplans," *IEEE Trans. Comput.-Aided Des.*, vol. 14, no. 1, pp. 129–132, Jan. 1995.
[7] T. C. Wang and D. F. Wong, "Optimal floorplan area optimization," *IEEE Trans. Comput.-Aided Des.*, vol. 11, no. 8, pp. 992–1001, Aug. 1992.
[8] 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*, 1997, pp. 265–270.
[9] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proc. ICCAD*, 1996, pp. 484–491.
[10] T. S. Moh, T. S. Chang, and S. L. Hakimi, "Globally optimal floorplanning for a layout problem," *IEEE Trans. Circuits Syst. I*, vol. 43, no. 9, pp. 713–720, Sep. 1996.
[11] H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/pre-placed modules," in *Proc. ISPD*, 1998, pp. 167–172.
[12] 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. Comput.-Aided Design*, vol. 20, no. 5, pp. 687–692, May 2001.
[13] C. Lin, H. Zhou, and C. Chu, "A revisit to floorplan optimization by Lagrangian relaxation," in *Proc. ICCAD*, 2006, pp. 164–171.
[14] A. B. Kahng, "Classical floorplanning harmful," in *Proc. ISPD*, 2000, pp. 207–213.
[15] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.
[16] J.-M. Lin and Z.-X. Hung, "UFO: Unified convex optimization algorithms for fixed-outline floorplanning considering pre-placed modules," *IEEE Trans. Comput.-Aided Des.*, vol. 30, no. 7, pp. 1034–1044, Jul. 2011.
[17] J. Z. Yan and C. Chu, "Optimal slack-driven block shaping algorithm in fixed-outline floorplanning," in *Proc. ISPD*, 2012, pp. 179–186.
[18] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, MA: Cambridge Univ. Press, 2004.
[19] J. Cong, M. Romesis, and J. R. Shinnerl, "Fast floorplanning by look-ahead enabled recursive bipartitioning," in *Proc. ASP-DAC*, 2005, pp. 1119–1122.

Jackey Z. Yan received the B.S. degree in automation from the Huazhong University of Science and Technology, Wuhan, China, in 2006, and the Ph.D. degree in computer engineering from Iowa State University, Ames, in 2011.

Since November 2010, he has been a Senior Member of the Technical Staff with the Placement Technology Group, Cadence Design Systems, Inc., San Jose, CA. His current research interests include very large scale integration physical designs, specifically in algorithms for floorplanning and placement, and physical synthesis integrated system-on-a-chip designs.

Dr. Yan's work on fixed-outline floorplanning was nominated for the Best Paper Award at Design Automation Conference in 2008. His another work on hypergraph clustering for wirelength-driven placement was nominated for the Best Paper Award at the International Symposium on Physical Design (ISPD) in 2010. He was a recipient of the Best Paper Award at ISPD for his research on floorplan block shaping algorithm in 2012, the Graduate Research Excellence Award for Class of 2011 from Iowa State University, and the ICD Department Level Award from Cadence Design Systems, Inc., in 2011.

Chris Chu (M'99–F'12) received the B.S. degree in computer science from the University of Hong Kong, Pokfulam, Hong Kong, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Texas, Austin, in 1994 and 1999, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University, Ames. His area of expertise includes computer-aided design of very large scale integration physical design, and design and analysis of algorithms. His current research interests include performance-driven interconnect optimization and fast circuit floorplanning, placement, and routing algorithms.

Dr. Chu was a recipient of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Paper Award at 1999 for his research on performance-driven interconnect optimization. He was a recipient of another IEEE TCAD Best Paper Award at 2010 for his research on routing tree construction. He was a recipient of the International Symposium on Physical Design (ISPD) Best Paper Award in 2004 for his research on efficient placement algorithm, the ISPD Best Paper Award in 2012 for his research on floorplan block shaping algorithm, and the Bert Kay Best Dissertation Award for 1998 to 1999 from the Department of Computer Sciences, University of Texas, Austin.