Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing

Tung-Chieh Chen, *Student Member, IEEE*, and Yao-Wen Chang, *Member, IEEE*

***Abstract*—Unlike classical floorplanning that usually handles only block packing to minimize silicon area, modern very large scale integration (VLSI) floorplanning typically needs to pack blocks within a fixed die (outline), and additionally considers the packing with block positions and interconnect constraints. Floorplanning with bus planning is one of the most challenging modern floorplanning problems because it needs to consider the constraints with interconnect and block positions simultaneously. In this paper, the authors study two types of modern floorplanning problems: 1) fixed-outline floorplanning and 2) bus-driven floorplanning (BDF). This floorplanner uses B*-tree floorplan representation based on fast three-stage simulated annealing (SA) scheme called Fast-SA. For fixed-outline floorplanning, the authors present an adaptive Fast-SA that can dynamically change the weights in the cost function to optimize the wirelength under the outline constraint. Experimental results show that this floorplanner can achieve 100% success rates efficiently for fixed-outline floorplanning with various aspect ratios. For the BDF, the authors explore the feasibility conditions of the B*-tree with the bus constraints, and develop a BDF algorithm based on the conditions and Fast-SA. Experimental results show that this floorplanner obtains much smaller dead space for the floorplanning with hard/soft macro blocks, compared with the most recent work. In particular, this floorplanner is more efficient than the previous works.**

***Index Terms*—Floorplanning, physical design.**

I. Introduction

As THE DESIGN complexity increases dramatically, modern very large scale integration (VLSI) floorplanning incurs more sophisticated constraints with the die outline, interconnect planning, and block positions. As pointed out by Kahng in [10], modern VLSI design is based on a fixed-die (fixed-outline) floorplan, rather than a variable-die one. A floorplan with pure area minimization without any fixed-outline constraints may be useless because it cannot fit into the given outline. Unlike classical floorplanning that usually handles only block packing to minimize silicon area, therefore, modern floorplanning should be formulated as a fixed-outline floorplanning.

The fixed-outline floorplanning has been shown to be much more difficult than the outline-free floorplanning [2]. Based on the sequence pair (SP) representation [14], Adya and Markov [2], [3] first present new objective functions to drive simulated annealing (SA) and new types of moves that better guide local search for fixed-outline floorplanning. Lin *et al.* [13] apply evolutionary search to handle fixed-outline floorplanning based on the normalized polish expression [24].

Floorplanning with position constraints is also prevailing in modern floorplan designs. There are many types of position constraints in modern floorplanning, such as range, symmetry, alignment, and bus constraints. Among these position constraints, bus-driven floorplanning (BDF) is one of the most challenging modern floorplanning problems because it needs to consider the constraints with interconnect and block positions simultaneously. In particular, the interconnection on the chip becomes more congested as technology advances and, thus, bus routing becomes a challenging task. Since buses have different widths and go through multiple blocks, the positions of the blocks greatly affect the bus routing. To make the bus routing easier, we shall consider the bus planning earlier in the floorplanning stage [26].

Floorplanning with the alignment constraint is closely related to BDF. The alignment constraint is considered in [23] and [25]. For the constraint, the alignment blocks are required to be aligned in a row and abut one by one. However, blocks involved in a bus do not need to be placed adjacent to each other. Rafiq *et al.* [16], [17] proposed a BDF. The bus defined in their works is composed of wires connecting only two blocks, which is not general enough to capture real bus designs. The general BDF that allows a bus to connect multiple blocks is first studied in [26]. In this work, the buses are placed in the top two layers and go either horizontally or vertically in one layer. For this problem, Xiang *et al.* [26] proposed an algorithm based on the SP representation. Nevertheless, the SP representation incurs a larger solution space, and thus it is less efficient to find a high-quality solution. Note that the solution space for SP is $(n!)^2$, and the solution space for B*-tree that we used in this paper is only $O(n!2^{2n}/n^{1.5})$ [5].

We study in this paper two types of modern floorplanning problems: 1) fixed-outline floorplanning and 2) BDF. Our floorplanner uses the B*-tree floorplan representation [5] and is based on a fast three-stage SA scheme, called Fast-SA. The Fast-SA is significantly different from the existing SA schemes that try to speed up the annealing process, e.g., the well-known TimberWolf [19]–[21] that uses a two-stage technique

Manuscript received June 23, 2005; revised September 22, 2005. This work was supported in part by MediaTek Inc., by the National Science Council of Taiwan under Grants NSC 93-2215-E-002-009, NSC 93-2220-E-002-001, and NSC 93-2752-E-002-008-PAE, and by SpringSoft Inc. This paper was presented in part at the 2005 Association of Computing Machinery (ACM) International Symposium on Physical Design (ISPD'05), San Francisco, CA, April 2005. This paper was recommended by Associate Editor P. H. Madden.

T.-C. Chen is with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: tungchieh@ntu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: ywchang@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2006.870076

to control the temperature updating function to reduce the iterations. Our Fast-SA consists of three stages of temperature modification. Experimental results show that Fast-SA is suitable for block floorplanning; it is more efficient than the classical and TimberWolf SA to obtain high-quality floorplans.

For fixed-outline floorplanning, we present an adaptive Fast-SA that can dynamically change the weights in the cost function under the outline constraint. The adaptive Fast-SA controls the parameters of the cost function dynamically according to a set of the most recent floorplan solutions. Experimental results show that our method achieves a success rate of 100% for the fixed-outline floorplanning with a dead space of 10% and various aspect ratios, compared to the average success rates of 30.3%, 65.5%, and 99.4% obtained by genetic floorplan algorithm (GFA) [13], Parquet-4.5 [15] using SP, and Parquet-4.5 using B*-tree, respectively. Further, the wirelength with the fixed-outline constraint is reduced by 6% on average, compared with Parquet-4.5.

For BDF, we explore the feasibility conditions of the B*-tree with the bus constraints, and develop a BDF algorithm based on the conditions and Fast-SA. Compared with the most recent work by Xiang *et al.* [26], our method based on Fast-SA obtains much smaller dead space for the floorplanning with hard/soft blocks. In particular, our floorplanner is more efficient than the previous works.

The rest of this paper is organized as follows. Section II reviews the B*-tree floorplanning representation. Section III presents the Fast-SA scheme. Section IV copes with the fixed-outline floorplanning based on adaptive Fast-SA. Section V deals with BDF based on Fast-SA. The experimental results are reported in Section VI. Finally, we give the conclusion in Section VII.

II. B*-Tree Representation

A B*-tree [5] is an ordered binary tree for modeling nonslicing or slicing floorplans. Given an admissible placement [9] (in which no blocks can move left or down), we can construct a unique B*-tree in linear time to model the placement. Further, given a B*-tree, we can also obtain a legal placement by packing the blocks in amortized linear time with a contour structure [5].

Fig. 1 shows an admissible placement and its corresponding B*-tree. A B*-tree is an ordered binary tree with its root corresponding to the block on the bottom-left corner. Similar to the depth-first search (DFS) procedure, we construct a B*-tree T for an admissible placement in a recursive fashion. Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i be the set of blocks located on the right-hand side and adjacent to b_i. The left child of the node n_i corresponds to the lowest unvisited block in R_i. The right child of n_i represents the lowest block located above and with its x-coordinate equal to that of b_i.

Given a B*-tree T, its root represents the block on the bottom-left corner, and thus the coordinate of the block is $(x_{\rm root}, y_{\rm root}) = (0, 0)$. If node n_j is the left child of node n_i, block b_j is placed on the right-hand side and adjacent to block b_i; i.e., $x_j = x_i + w_i$. Otherwise, if node n_j is the right child of n_i, block b_j is placed above block b_i, with the x-coordinate of b_j equal to that of b_i; i.e., $x_j = x_i$. Therefore, given a B*-tree, the x-coordinates of all blocks can be determined by traversing the tree once in linear time. Further, each y-coordinate can be computed by a contour data structure in amortized constant time [5], making the overall evaluation an amortized linear-time process.

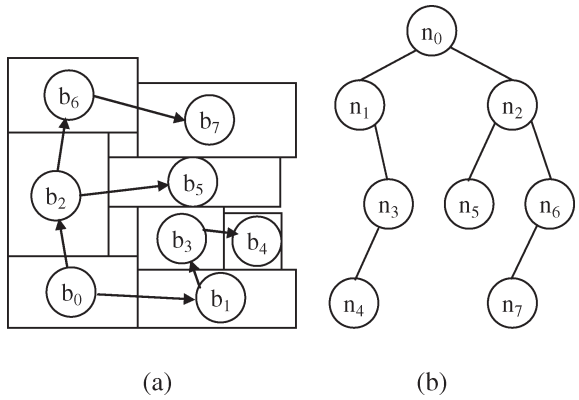

Fig. 1. (a) Admissible placement and (b) B*-tree representing placement.

III. Fast SA

SA [11] is widely used for floorplanning. It is an optimization scheme with nonzero probability for accepting inferior (uphill) solutions. The probability depends on the difference of the solution quality and the temperature. The probability is typically defined by

$$\text{Prob} = \min\left\{1, \mathrm{e}^{\frac{-\Delta C}{T}}\right\} \tag{1}$$

where ΔC is the difference of the cost of the neighboring state and that of the current state, and T is the current temperature. In the classical annealing schedule, the temperature is reduced by a fixed ratio λ (say, 0.85 as recommended by most previous works) for each iteration of annealing.

The excessive running time, however, is a significant drawback of the classical SA process. To reduce the running time of SA for searching for the desired solutions more efficiently, several annealing schemes of controlling the temperature changes during the annealing process have been proposed. The annealing schedule used in TimberWolf [19]–[21] is probably the most successful scheme reported in the literature. It increases λ gradually from its lowest value (0.8) to its highest value (approximately 0.95), and then gradually decreases λ back to its lowest value. (See Fig. 2(a) and (b) for the respective temperature changes for classical SA and TimberWolf SA as the search time goes by.)

We propose a Fast-SA scheme. The motivation is that we try to reduce the number of uphill moves in the beginning. We observe that it is not efficient and effective to accept too many uphill moves in the beginning since most of the solutions are inferior. We resort to greedy search to find a local optimal faster. Starting with the local optimal, we then switch to normal SA. By doing so, we can save time for searching for good

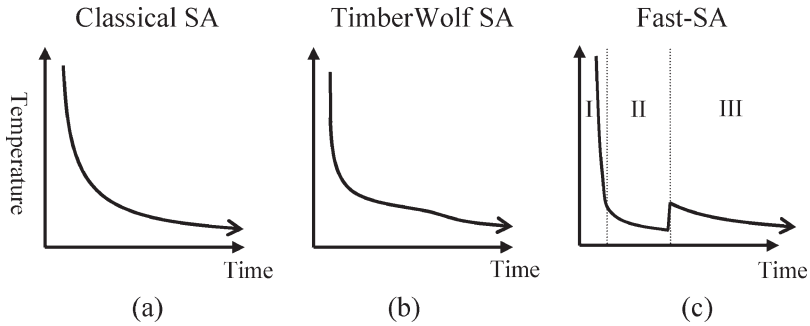

Fig. 2. Temperature versus search time for (a) classical SA, (b) TimberWolf SA, and (c) Fast-SA. Temperature of TimberWolf SA drops faster than that of classical SA at beginning and ending iterations, but slower in middle iterations. Fast-SA consists of three stages.

solutions. To implement the above ideas, our Fast-SA consists of three stages:

1) high-temperature random search stage;
2) pseudogreedy local-search stage;
3) hill-climbing search stage.

At the first stage, the temperature is set to a very large value. According to (1), the probability to accept an inferior solution approaches one. This can avoid getting trapped in a local optimal in the very first. At the second stage, we let the temperature approach zero to accept only a small number of inferior solutions. At the third stage, the temperature is raised to facilitate hill climbing to search for better solutions. The temperature reduces gradually, and very likely it finally converges to a desirable solution.

Since the new SA scheme saves many iterations to explore the solution space, it could devote more time to finding better solutions in the hill-climbing stage. This makes annealing much more efficient and effective. To implement the annealing scheme, we derive the temperature T updating function of the Fast-SA by

$$T_n = \begin{cases} \frac{\Delta_{\text{avg}}}{\ln P} & n = 1 \\ \frac{T_1 \langle \Delta_{\text{cost}} \rangle}{nc} & 2 \le n \le k \\ \frac{T_1 \langle \Delta_{\text{cost}} \rangle}{n} & n > k. \end{cases} \tag{2}$$

Here, n is the number of iterations, Δ_{avg} is the average uphill cost, P is the initial probability to accept uphill solutions, $\langle \Delta_{\text{cost}} \rangle$ is the average cost change (new cost $-$ old cost) for the current temperature, and c and k are user-specified parameters. At the first iteration, the temperature is set according to the given initial accepting probability P and the average uphill cost Δ_{avg}. Since P is usually set close to one, therefore, it performs random search to find a good solution. Then, it enters the pseudogreedy local-search stage until the kth iteration. Here, c is a user-defined parameter to control how low the temperature is in the second stage. We usually choose a large c to make $T \to 0$ so that it only accepts good solutions to perform pseudogreedy searches. After k iterations, the temperature jumps up to further improve the solution quality. The value of $\langle \Delta_{\text{cost}} \rangle$ affects the reduction rate of the temperature. If the cost of a neighboring solution changes significantly, $\langle \Delta_{\text{cost}} \rangle$ is larger and, thus, the temperature reduces slower. In contrast, if $\langle \Delta_{\text{cost}} \rangle$ is smaller, it implies that the cost of the neighboring solution only changes a little; for this case, we reduce the temperature more to reduce the number of iterations. Since the cost function is normalized to one, therefore, $\langle \Delta_{\text{cost}} \rangle < 1$, and it ensures the decreasing temperature. The behavior of the temperature changes is illustrated in Fig. 2(c). The number of iterations in the second stage can be determined by the problem size. The smaller the problem size, the smaller the k value. In our cases, we set $c = 100$ and $k = 7$ for floorplanning problems. Note that the initial temperature for the Fast-SA is the same as that for the classical SA, i.e., $T_1 = \Delta_{\text{avg}} / \ln P$. The initial temperature T_1 needs to be kept high to avoid getting trapped in a local minimum in the very beginning.

In this paper, we use the B*-tree representation to model a floorplan. Each B*-tree corresponds to a floorplan. Therefore, the solution space consists of all B*-trees with the given nodes (blocks). To find a neighboring solution, we perturb a B*-tree to get another B*-tree by the following operations.

1) Op1: Rotate a block.
2) Op2: Move a node/block to another place.
3) Op3: Swap two nodes/blocks.
4) Op4: Resize a soft block.

For Op1, we rotate a block for a B*-tree node, which does not affect the B*-tree structure. For Op2, we delete a node and move it to another place in the B*-tree. For Op3, we swap two nodes in the B*-tree. For Op4, we adjust the aspect ratio of a soft block. The soft-block adjustment algorithm is described in the experimental results. After packing for a B*-tree, we obtain a new floorplan. Whether or not we take the new solution depends on the current temperature and the cost function. The cost function is defined based on problem requirements. For example, we may adopt the following cost function to optimize the wirelength and the area of the floorplan:

$$\text{Cost} = \alpha \frac{A}{A_{\text{norm}}} + (1 - \alpha) \frac{W}{W_{\text{norm}}} \tag{3}$$

where A is the current area, A_{norm} is the average area, W is the current wirelength, W_{norm} is the average wirelength, and α controls the weights for area and wirelength. To calculate Δ_{avg} [in (2)], A_{norm} and W_{norm} [in (3)], we perturb the B*-tree for n times to obtain another n floorplans before the SA process starts. Δ_{avg} is the average of all positive (uphill)

cost change for these n perturbations. A_{norm} and W_{norm} are the average area and wirelength of these floorplans, respectively. The value n is proportional to the problem size (the number of modules).

IV. Fixed-Outline Floorplanning

In this section, we present an adaptive Fast-SA scheme that can dynamically change the weights for simultaneous chip area and wirelength optimization under the fixed-outline constraint.

A. Fixed-Outline Constraints

For a collection of blocks with the total area A and the given maximum percent of dead space Γ, we construct a fixed outline with the aspect ratio R^*, i.e., height/width. The height H^* and width W^* of the outline are defined by [2]

$$H^* = \sqrt{(1+\Gamma)AR^*}, \quad W^* = \sqrt{\frac{(1+\Gamma)A}{R^*}}. \tag{4}$$

B. Algorithm Overview

We use our Fast-SA to search for a desired solution. We initialize a B*-tree as a complete binary tree, and perturb a B*-tree to another by the operations described in Section III. For some blocks, they only have one feasible orientation to fit into the fixed outline. We mark all such blocks as nonrotatable blocks and set their orientations before performing perturbations. For Op1, we can only choose a rotatable block. Since we intend to minimize the wirelength/area of the floorplan, we always record the floorplan of the minimum wirelength/area during SA. After the temperature cools down enough, we terminate the SA process and report the best floorplan.

The researchers in [3] directly penalize the floorplan solution with the amounts of its height and width violating the outline constraint by adding $\max\{H - H^*, 0\} + \max\{W - W^*, 0\}$ to their cost function. We observe that such a cost function might not lead to high-quality floorplanning results. The reasons are twofold: 1) If we assign a large weight for the penalty, it is very easy to get trapped in the local minimum when the first feasible solution is found and 2) If we give a small weight for the penalty, in contrast, it might not be easy to obtain a feasible floorplan that satisfies the outline constraint. Thus, it is desirable to derive a new cost function for a fixed-outline floorplanning to remedy these problems.

In addition to the wirelength/area objective, we add an aspect-ratio penalty to the cost function. The idea is that if the aspect ratio of the floorplan is similar to that of the outline, and the dead space of the floorplan is smaller than the maximum percentage of dead space Γ, then the floorplan can fit into the outline. Assume that the current aspect ratio of the floorplan is R. We define the cost function Φ for a floorplan solution F by

$$\Phi(F) = \alpha A + \beta W + (1 - \alpha - \beta)(R - R^*)^2 \tag{5}$$

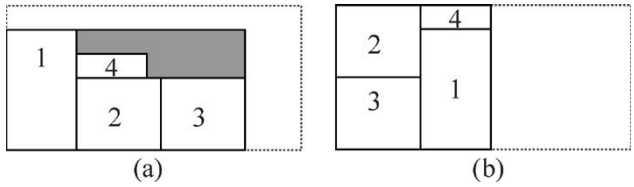

Fig. 3. Two feasible floorplans. Dotted line is fixed outline, and fixed-outline aspect ratio R^* is 0.5. Floorplan aspect ratio R is (a) 0.5 and (b) 1.

where A is the current floorplan area, W is the current wirelength, R is the current floorplan aspect ratio, R^* is the desired floorplan aspect ratio, and α, β are user-defined parameters. As the example shown in Fig. 3, the best aspect ratio of the floorplan in the fixed outline may not be the same as that of the outline. In this case, we shall decrease the weight of aspect-ratio penalty to concentrate more on the floorplan wirelength/area optimization. (We will discuss the relationship of weight values in reporting the experimental results.) Since it is not easy to determine the weight of the wirelength/area and the weight of the aspect-ratio penalty, we show in the following how to adaptively control the weight for different floorplans.

C. Adaptive SA

We focus on area optimization ($\beta = 0$) with the fixed-outline constraint for easier presentation; the technique readily applies to wirelength optimization as well. Since R^* and Γ are user-specified parameters, the weights for the area and the aspect ratio should be determined by the given values. It is not easy to determine the best α, and it is not efficient to try every α value in the cost function. Therefore, we use an adaptive method to control α according to n most recent floorplans found. The area weight α is defined by

$$\alpha = \alpha_{\text{base}} + (1 - \alpha_{\text{base}}) \left(\frac{n_{\text{feasible}}}{n} \right) \tag{6}$$

where n_{feasible} is the number of feasible solutions in n most recent floorplan solutions, and α_{base} $(0 < \alpha_{\text{base}} < 1)$ is determined by the user, say $\alpha_{\text{base}} = 0.5$. The range of α is from α_{base} to 1.0. The value of α cannot be larger than 1.0 since the weight of the aspect-ratio penalty is $1 - \alpha$. The more feasible floorplans found, the smaller weight of the aspect ratio is. The experimental results for using adaptive SA are reported in Section IV-B.

D. Algorithm

Fig. 4 summarizes our algorithm. First, we mark all nonrotatable blocks, set their orientations, and initialize a B*-tree with input blocks as a complete binary tree. Then, we start with the adaptive Fast-SA process. After each perturbation, we perform packing and evaluate the B*-tree cost. Whether or not we accept the new B*-tree is determined by (1). If the floorplan is better than the current best one, we record it as the best floorplan. Then, we update the temperature T and update the

Algorithm: Fix-Outline Floorplan(F)
Input: A set of blocks and a fixed outline.
Output: A floorplan within the outline.
1 Mark all non-rotatable blocks and set their orientations;
2 Initialize a B*-tree with input blocks;
3 // Start the adaptive Fast-SA process;
4 $T \leftarrow$ Initial temperature;
5 **do**
6 Perturb the B*-tree;
7 Pack macro blocks;
8 Evaluate the B*-tree cost;
9 Decide if we should accept the new B*-tree.
10 Modify the weights in the cost function;
11 Update T;
12 **until** converged or cooling down;
13 **return** the best solution.

Fig. 4. Adaptive SA for fixed-outline floorplanning.

weights in the cost function. The weights in the cost function are updated dynamically (adaptively), instead of keeping the weight as a constant. This process continues to the end of SA, and the best solution is reported. The time complexity for each perturbation and evaluation of the fixed-outline floorplanning is the same as the packing time of one B*-tree, i.e., $O(n)$, where n is number of macro blocks.

V. BDF

In this section, we explore the feasibility conditions of the B*-tree with the bus constraints and develop a BDF algorithm based on the conditions and Fast-SA.

A. BDF Formulation

We consider a chip with multiple metal layers, and buses are assigned on the top two layers. The orientation of buses is either horizontal or vertical. The problem of BDF is defined as follows [26].

Given n rectangular macro blocks $B = \{b_i | i = 1, \ldots, n\}$ and m buses $U = \{u_i | i = 1, \ldots, m\}$, each bus u_i has a width t_i and goes through a set of blocks B_i, where $B_i \subseteq B$ and $|B_i| = k_i$. Decide the positions of macro blocks and buses such that there is no overlap between any two blocks or between any two horizontal (vertical) buses, and bus u_i goes through all of its k_i blocks. At the same time, the chip area and the bus area are minimized.

For convenience, let $\langle g, t, \{b_1, \ldots, b_k\}\rangle$ represent a bus u, where $g \in \{H, V\}$ is the orientation, t is the bus width, and b_i, $i = 1, \ldots, k$, are the blocks that the bus goes through. In short, a bus is represented by $\{b_1, \ldots, b_k\}$. Fig. 5 shows a feasible horizontal bus.

B. B*-Tree Properties for Bus Constraints

The blocks that a bus goes through must locate in an alignment range, i.e., the vertical or horizontal overlap of the blocks has to be larger than the bus width. For a B*-tree, the left child n_j of the node n_i represents the lowest adjacent block b_j, which is right to the block b_i (i.e., $x_j = x_i + w_i$). Therefore, the blocks have horizontal relationships in a left-skewed subtree.

Property 1: In a B*-tree, the nodes in the left-skewed subtree may satisfy a horizontal bus constraint.

Blocks are compacted to the bottom and left after packing. Therefore, the blocks associated with a left-skewed subtree of a B*-tree may be aligned together if no block falls down during packing. We introduce dummy blocks to solve the falling-down problem. In Fig. 6(a), the blocks b_2 and b_4 are displaced because they fall down during packing. We add dummy blocks right below the displaced blocks. The dummy blocks have the same x-coordinates as the displaced blocks, and the widths are also the same. In Fig. 6(b), we adjust the heights of the dummy blocks to shift the displaced blocks to satisfy the bus constraint. After adjusting the heights of the dummy blocks, we can guarantee that the blocks are feasible with the horizontal bus constraint. The height Δ_i of the dummy block D_i can be computed by

$$\Delta_i = \begin{cases} (y_{\min} + t) - (y_i + h_i), & \text{if } (y_{\min} + t) > (y_i + h_i) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where $x_i(y_i)$ is the $x(y)$-coordinate of block b_i, and $y_{\min} = \max\{y_i | i = 1, 2, \ldots, k\}$ for bus $\{b_1, \ldots, b_k\}$. Fig. 7 shows an example of a feasible horizontal bus by inserting dummy blocks D_5 and D_6.

Property 2: By inserting dummy blocks of appropriate heights, we can guarantee the feasibility of a horizontal bus with blocks whose corresponding B*-tree nodes are in the left-skewed subtree.

For a B*-tree, the right child n_j of the node n_i represents the closest upper block b_j, which has the same x-coordinate as the block b_i (i.e., $x_j = x_i$). Therefore, the blocks in the right-skewed subtree are aligned with the x-coordinate. Assume the minimum width of the macro blocks that the bus goes through is larger than the bus width. The structure forms a vertical bus. In the example shown in Fig. 8, the nodes n_3 and n_5 are in the right-skewed subtree of n_0, so the blocks b_0, b_3, and b_5 satisfy the vertical bus constraint.

Property 3: In a B*-tree, the nodes in the right-skewed subtree can guarantee the feasibility of a vertical bus.

Note that the vertical bus is not constrained to be at the right subtree of the node corresponding to the lowest module among the set. According to property 3, the nodes in the right-skewed subtree can guarantee the feasibility of a vertical bus, but it is not always true vice versa.

C. Twisted-Bus Structure

Consider two buses simultaneously, we cannot always fix the horizontal bus constraints by inserting dummy blocks. As the example shown in Fig. 9, two buses are considered: $u = \{b_0, b_3\}$ and $v = \{b_2, b_6\}$. We can add the dummy block D_0 (D_2) below b_0 (b_2) to satisfy the horizontal bus u (v). However, we cannot satisfy two horizontal bus constraints at the same time since two buses are twisted. The idea to discard B*-trees with twisted-bus structures is to reduce the solution space and make the solution searching more efficient. Note that it is impossible to fix a twisted-bus structure by

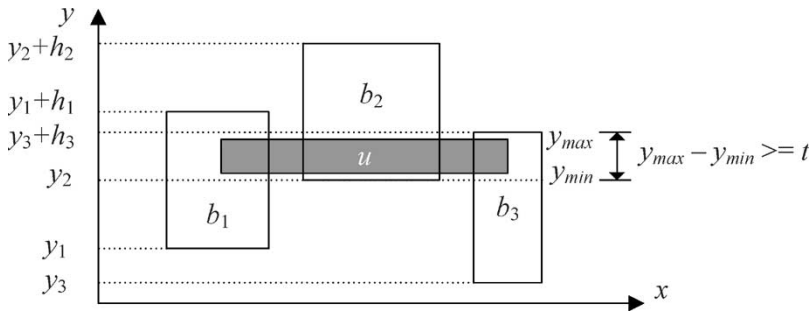

Fig. 5. Feasible horizontal bus $u = \langle H, t, \{b_1, b_2, b_3\}\rangle$.

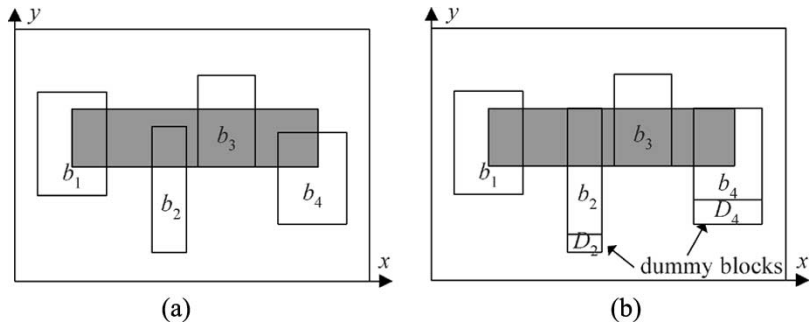

Fig. 6. (a) Infeasible floorplan since block-overlap range is less than bus width t and (b) inserting dummy blocks, bus $\langle H, t, \{b_1, b_2, b_3, b_4\}\rangle$ is satisfied.

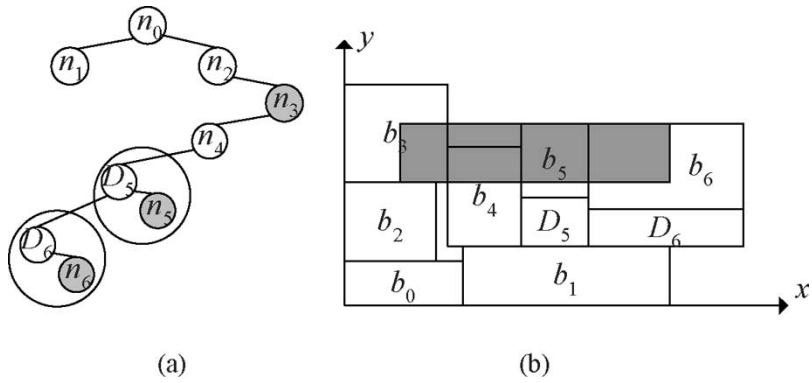

Fig. 7. (a) B*-tree with left-skewed subtree after inserting dummy nodes and (b) corresponding feasible horizontal bus $\langle H, t, \{b_3, b_5, b_6\}\rangle$.

inserting dummy blocks. Discarding such a configuration will not remove any feasible solutions. We directly examine the twisted-bus structure by checking the B*-tree nodes. Consider two buses u and v. If one node of bus u is in the right-skewed subtree of one node of bus v, and one node of bus v is in the right-skewed subtree of one node of bus u, then it will incur a twisted-bus structure. Therefore, we shall discard a B*-tree with such an infeasible tree topology during solution perturbation. Note that not all potential twisted-bus structures are checked through the aforementioned procedure. Fig. 9 shows a twisted-bus structure, where n_3 is in the right-skewed subtree of n_2, and n_6 is in the right-skewed subtree of n_0.

D. Bus Overlapping

When multiple buses are considered, we need to avoid overlaps between buses. For example, in Fig. 10, two horizontal buses are to be assigned. The buses $u = \{b_0, b_4\}$ ($v = \{b_2, b_3\}$) are feasible when we consider only one bus. However, the vertical space is not large enough for fitting two buses. In this case, we compute the minimum shifting distance for the block b_2, and insert a dummy block D_2 right below b_2. Thus, the two buses can be assigned at the same time by inserting the dummy block. In our implementation, we check the buses one by one using the order in the benchmark. When one bus is examined, we also allocate the space for the bus according to the bus width and the block positions. If we find the space of one bus overlapping with another bus, we will let the new bus be on top of the other and insert dummy blocks to avoid overlaps.

E. Fixed I/O Ports

Sometimes buses are connected to I/O ports that are fixed on the boundary. If the I/O ports to which the bus connects are at

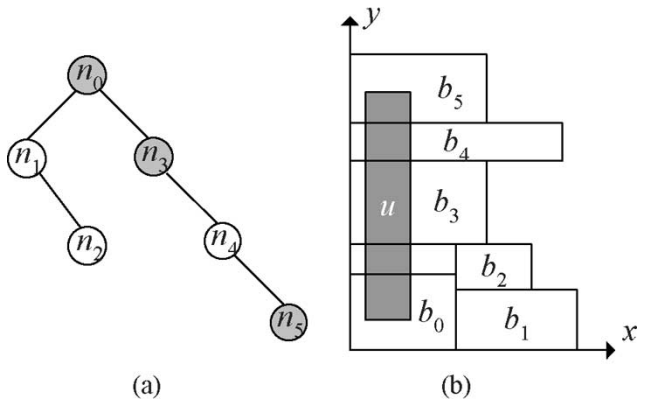

Fig. 8. (a) Right-skewed subtree and (b) corresponding feasible vertical bus $u = \langle V, t, \{b_0, b_4, b_5\}\rangle$.

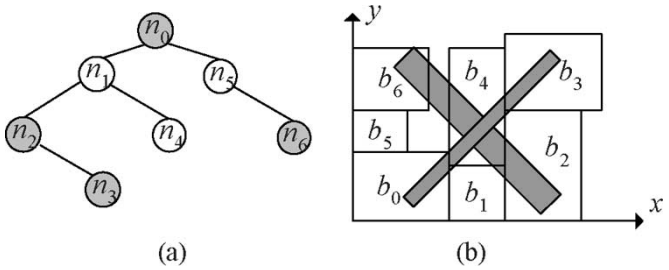

Fig. 9. Infeasible floorplan for two buses $u = \{b_0, b_3\}$ and $v = \{b_2, b_6\}$. (a) Twisted-bus structure, where n_3 is in right subtree of n_2, and n_6 is in right subtree of n_0. (b) Corresponding floorplan. Two twisted buses cannot be satisfied simultaneously by inserting dummy blocks.

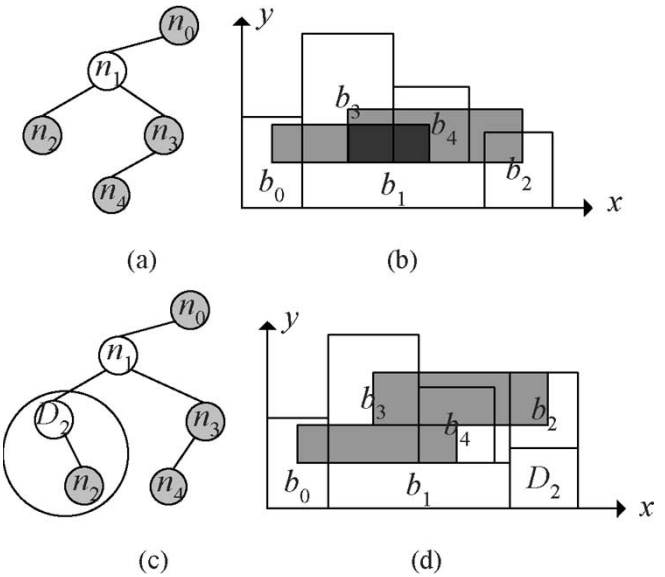

Fig. 10. Two horizontal buses $u = \{b_0, b_4\}$ and $v = \{b_2, b_3\}$. (a) Two buses overlap. (b) By inserting dummy block, we can get feasible floorplan without bus overlapping.

the top/bottom side of the chip, only the vertical bus may be feasible. Similarly, if the I/O ports to which the bus connected are at the left/right side of the chip, only the horizontal bus may be feasible. Thus, the directions of the buses can be fixed, and we do not need to check the directions when deciding the bus locations. To avoid the block-falling-down problem with fixed I/O ports, we need to set the heights of the dummy blocks considering the positions of fixed I/O ports. We can directly set $y_{\min}$ in (7) to the y-coordinate value of the fixed I/O port to which the bus connects. By doing so, it will try to align the blocks with the fixed I/O port to make the horizontal bus feasible.

Fig. 11 shows an example of inserting dummy blocks, considering a fixed I/O port F. The heights for the dummy blocks D_3, D_5, and D_6 are $(y_F + t) - (y_3 + h_3)$, $(y_F + t) - (y_5 + h_5)$, and $(y_F + t) - (y_6 + h_6)$, respectively, where y_i is the y-coordinate of block i, and t is the bus width. The bus $\{b_3, b_5, b_6, F\}$ is feasible after inserting dummy blocks.

F. Algorithm

Our BDF algorithm applies Fast-SA based on the B*-tree representation. Fig. 12 summarizes our algorithm. First, we initialize B*-tree as a complete binary tree, and start with the Fast-SA process. After each perturbation and nondummy block packing, we check if there exists a "twisted-bus structure" in the B*-tree. If any, we simply discard the current solution and perturb the B*-tree again. This checking can save time to find feasible solutions. If there is no twisted-bus structure in the B*-tree, we insert the dummy blocks to the appropriate nodes to fix the horizontal bus constraints and bus overlapping. After adjusting the heights of the dummy blocks, we pack the B*-tree again. Then, we decide the bus locations so that there is no overlap between buses. After adjusting the heights of the dummy blocks and repacking the floorplan, some buses still may not be feasible. We refer to these buses as unassigned buses.

Since the objective function of BDF is to satisfy all bus constraints so that the chip area and the total bus area are minimized, we define the cost function Ψ for a floorplan solution F with the set of buses U as follows:

$$\Psi(F, U) = \alpha A + \beta B + \gamma M \tag{8}$$

where A is the chip area, B is the bus area, M is the number of unassigned buses, and α, β, and γ are user-specified parameters.

In the SA process, we record the floorplan solution with the most number of feasible buses and the lowest cost. After the SA process stops, we report the lowest cost with the least number of unassigned buses. Thus, we can find the desired floorplan with the most feasible buses.

Suppose we are given m buses and n blocks. According to Fig. 12, the combination of the pseudocode from line 4 to line 13 makes one perturbation and evaluation of the B*-tree. Line 4 takes $O(1)$ time for perturbation, and line 5 takes $O(n)$ time for B*-tree packing. In line 6, the time complexities for fixing horizontal bus constraints and bus overlap checking are $O(mn)$ and $O(m^2n)$, respectively. Line 9 takes $O(n)$ time for packing, line 10 takes $O(m^2n)$ time for deciding the bus locations, and lines 11–13 take $O(1)$ time. Thus, the total time complexity is $O(m^2n)$.

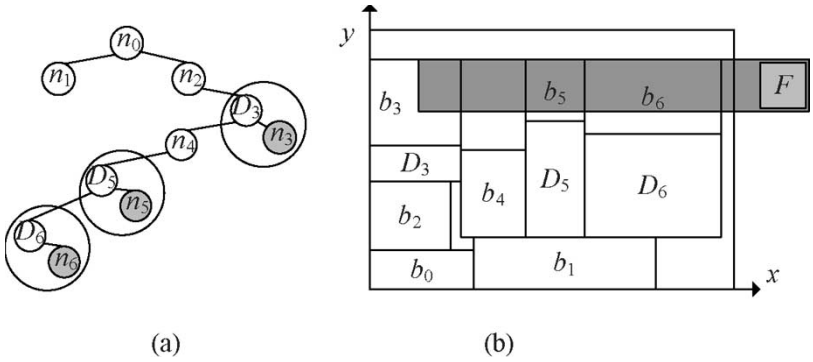

Fig. 11. Horizontal bus connects to fixed I/O port F, $u = \{b_3, b_5, b_6, F\}$. (a) B*-tree after inserting dummy nodes and (b) corresponding feasible horizontal bus.

Algorithm: Bus-Driven Floorplanning Using B*-trees
Input: A set of blocks and a set of bus constraints.
Output: A floorplan satisfying bus constraints with chip area and the total bus area being minimized.

```
1  Initialize a B*-tree for the input blocks;
2  // Perform the Fast-SA process;
3  do
4      Perturb the B*-tree;
5      Pack macro blocks without dummy blocks;
6      if there exists a "twisted-bus structure" in the B*-tree
7          then restart the do-loop;
8      Adjust the heights of dummy blocks to fix horizontal
           bus constraints and fix bus-overlapping;
9      Pack macro blocks with dummy blocks;
10     Decide bus locations;
11     Evaluate the floorplan cost;
12     Decide if we should accept the new B*-tree.
13     Update the temperature;
14 until converged or cooling down;
15 return the best solution.
```

Fig. 12. BDF algorithm.

G. Extension to Multibend Buses

In our formulation, we only allow the bus with 0 bend. The cost of a 0-bend bus is smaller than a multibend bus if their lengths are the same since vias are needed for a multibend bus, and bus routing becomes harder. However, when the number of blocks that a bus goes through is large, allowing multibend buses may lead to better solutions [12].

To extend our method to handle multibend BDF, we can first divide the blocks of a bus into different groups. These groups form different segments of the bus. If we allow k bends for a bus, then we have $k+1$ groups for a bus. We add a perturbation (see line 4 in Fig. 12) to change the current group of a block for a bus. During the SA, the groups of blocks are decided. If there are not any blocks in a group for a bus, the bend number is decreased by one. Therefore, the bend of a bus is also decided during SA. Note that i-bend buses are also allowed in j-bend BDF, where $i \leq j$. After deciding the locations for each segment (see line 10 in Fig. 12), a new step is needed to connect all segments to form a bus. We can simply extend the length of segments to connect each other. Fig. 13 shows an example of connecting segments. In Fig. 13(a), bus u is divided into two groups $\{b_0, b_5\}$ and $\{b_3, b_4\}$. The two groups form two

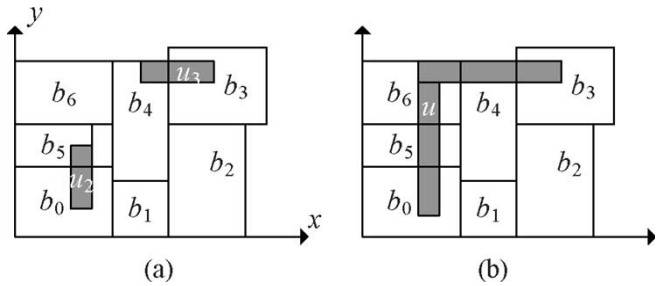

Fig. 13. (a) Two segments u_1 and u_2 for bus u. (b) After connecting segments, we obtain one-bend bus u.

TABLE I
DEAD SPACE AND CPU TIME FOR DIFFERENT SA SCHEMES USING GSRC FLOORPLAN BENCHMARK SUITE. WS IS DEAD SPACE (%)

	Classical SA		TimberWolf SA		Fast-SA (k=1)		Fast-SA (k=7)	
	WS (%)	Time (sec)	WS (%)	Time (sec)	WS (%)	Time (sec)	WS (%)	Time (sec)
n100	5.1	127	5.0	85	5.0	16	4.9	5.5
n200	5.0	537	4.9	406	5.1	59	5.1	37
n300	5.3	1293	5.1	1102	5.2	292	5.3	95
Comp.		1.0X		1.3X		5.3X		17.1X

segments, u_1 and u_2, of the bus. We stretch the segments to connect each other to form the 1-bend bus u.

The cost function Ψ' for a floorplan solution F with the set of buses U for multibend BDF is defined as

$$\Psi'(F, U) = \alpha A + \beta B + \gamma M + \delta N + \epsilon L \tag{9}$$

where A is the chip area, B is the bus area, M is the number of unassigned buses, N is the number of unassigned segments, L is the number of the bends, and α, β, γ, δ, and ϵ are user-specified parameters.

VI. EXPERIMENTAL RESULTS

We conducted extensive experiments to justify the effectiveness and efficiency of the Fast-SA scheme, our fixed-outline floorplanning algorithm, and our BDF algorithm. Our floorplanning package is available at http://eda.ee.ntu.edu.tw/research.htm.

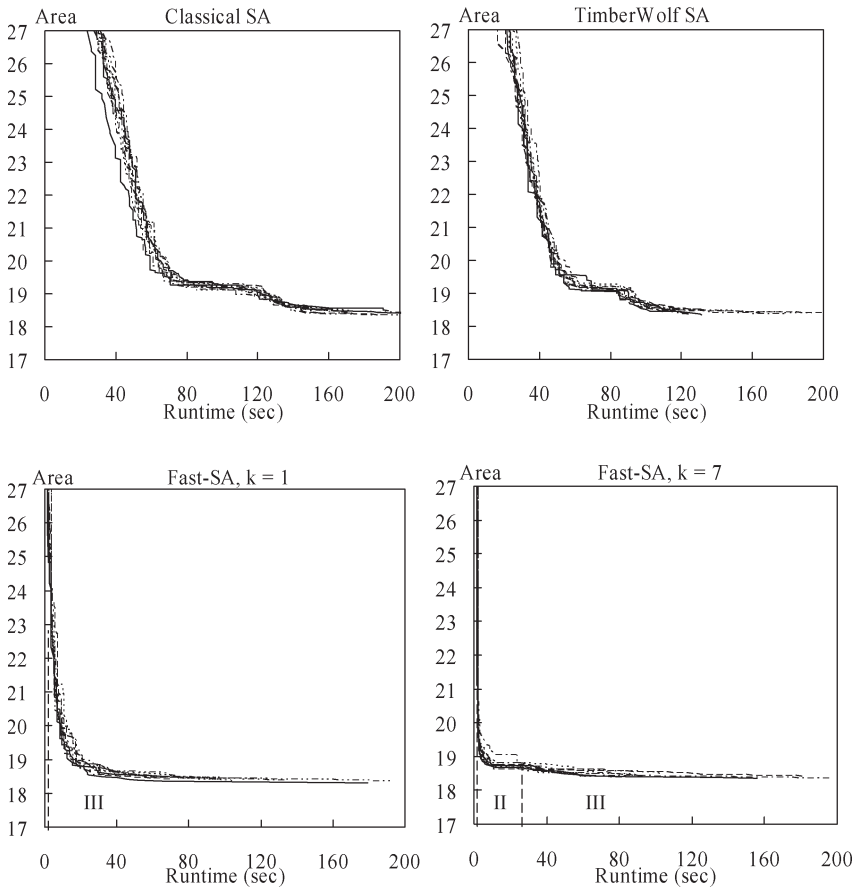

Fig. 14. Comparison for stability and convergence among classical SA, TimberWolf SA, Fast-SA with $k = 1$, and Fast-SA with $k = 7$. Different stages for Fast-SA are divided using vertical dotted lines (note that initial area is 69.76 mm^2, same for four SA schemes, and we focus on results with area smaller than 27 mm^2 to examine the effect more closely).

A. Convergence and Stability for Fast-SA

To test the efficiency of Fast-SA, we experimented on the three largest circuits in the GSRC floorplan benchmark suite [8], n100, n200, and n300 (which contain 100, 200, and 300 blocks, respectively). We implemented the classical SA, TimberWolf SA, and Fast-SA in the C++ programming language on an Intel Pentium 4 1.6-GHz PC with 256-MB memory. All of the SA algorithms are based on the B*-tree floorplan representation and the same initial temperature. The initial probabilities of accepting an uphill move are all set to 0.9. The only difference is the annealing schedule. For classical SA, the updating function for temperature T is

$$T_{\text{new}} = \lambda T_{\text{old}}, \qquad 0 < \lambda < 1. \tag{10}$$

The value of λ for classical SA is set to a fixed value 0.85 [18]. The annealing schedule of Fast-SA was described in Section III. For TimberWolf SA [19], [20] the value of λ is gradually increased from its lowest value to its highest one, and is then gradually decreased back to its lowest value. We set the lowest λ to 0.8, and set the highest λ to 0.95, the same as that used in [19]. It should be noted that the latest TimberWolf has made many improvements for its cost function, move selections, etc. In this paper, however, we focus on the "cooling schedule" of SA. In TimberWolfMC SA [21], the cooling schedule is determined experimentally: the value of λ is increased from 0.85 to 0.92, and is then decreased back to 0.80. By doing so, it can reduce the iterations in the high temperature and the low temperature, and increase the iterations in the middle temperature for SA. The cooling schedule is slightly changed in TimberWolf7.1 SA [22]; however, the methodology remains the same.

Table I compares the running times of the three different SA schemes based on comparable solution quality. We list the times to achieve the similar solution quality (say, around 5% dead space in this experiment) for the three annealing schemes. For the first Fast-SA, we set $k = 1$ to remove the greedy local-search stage. We reduced the running time in the high-temperature stage (stage 1), and spent more time in the hill-climbing stage (stage 3). This scheme achieved 5.3× speedup to generate comparable solutions, compared to the classical SA. For the second Fast-SA, we set $k = 7$ to perform six iterations of greedy local search; therefore, the convergence speed is even higher. The third stage of Fast-SA can avoid getting trapped in a local minimum in the second stage of Fast-SA. On the average, Fast-SA achieved a 12× speedup in finding the floorplan solutions of comparable areas.

Fig. 14 compares the convergence speed and the stability of the three SA schemes. For each SA scheme, the area is plotted

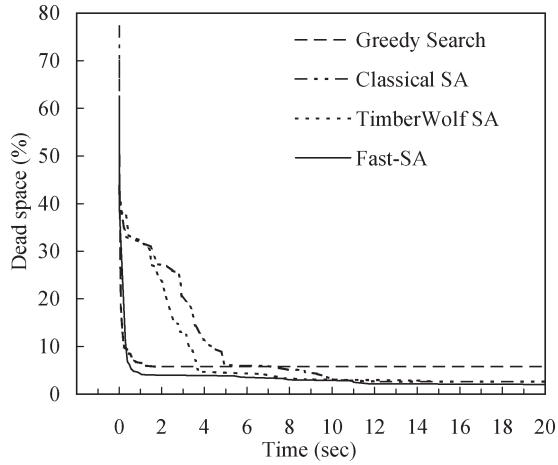

Fig. 15. Dead space versus run time among greedy search, classical SA, TimberWolf SA, and Fast-SA. Respective final dead spaces are 5.76%, 2.62%, 2.13%, and 2.00% for greedy search, classical SA, TimberWolf SA, and Fast-SA, respectively.

TABLE II
DEAD SPACES AND RUNTIMES FOR DIFFERENT SA SCHEMES (NA: NOT AVAILABLE)

Dead space	Greedy Runtime (sec)	Classical SA Runtime (sec)	TimberWolf SA Runtime (sec)	Fast-SA Runtime (sec)
10%	0.234	4.375	3.28	0.359
8%	0.562	4.859	3.58	0.375
6%	1.266	5.729	3.66	0.531
5%	NA	8.692	3.70	0.625
4%	NA	9.656	7.65	1.406

as a function of running time. We ran the n100 circuit for ten times for each SA scheme. The different stages for Fast-SA are divided using vertical dotted lines. As shown in the figure, the convergence speed of Fast-SA with the greedy local-search stage is much faster than Fast-SA without the greedy local-search stage, and Fast-SA without the greedy local-search stage is much faster than the classical SA. The TimberWolf SA is better than the classical SA but worse than Fast-SA. Note that the initial area is 69.76 mm^2, same for the three SA schemes. To view the convergence more clearly, we only plotted the results with the area smaller than 27 mm^2.

To compare greedy search, classical SA, TimberWolf SA, and Fast-SA in more detail, we performed an experiment for these annealing schemes on the Microelectronics Center of North Carolina (MCNC) benchmark ami49. In Fig. 15, the dead space is plotted as a function of the running time. As shown in Fig. 15 and Table II, the convergence speed of the greedy search is the fastest; it took only 0.234 s to find a floorplan solution of less than 10% dead space. However, the final solution for greedy search has a dead space of 5.76%. The classical (TimberWolf) SA method can further improve the solution quality until the dead space equals 2.62% (2.13%). Since Fast-SA combines the pseudogreedy local-search stage and the hill-climbing stage, its convergence speed is much faster than that of classical SA. Fast-SA only spent 0.625 s to obtain a floorplan solution of 5% dead space while classical SA needed 8.687 s. Fast-SA spent more iterations to find better floorplan solutions with dead spaces under 5%. Fast-SA achieved 2.00% dead space at last while classical SA only achieved 2.62%. Based on the results, the greedy search is not suitable for handling the floorplanning problems if the solution quality is a major concern, and Fast-SA is the best choice for the floorplanning problem addressed here (it achieved 13.9× speedup over classical SA for finding a floorplan solution of less than 5% dead space for this case).

TABLE III
SUCCESS PROBABILITY AND AVERAGE DEAD SPACE USING CONSTANT α AND ADAPTIVE α ON n100, $\Gamma = 10\%$

α		$R^* = 1$ Success prob.	$R^* = 1$ Average dead space	$R^* = 2$ Success prob.	$R^* = 2$ Average dead space
Constant α	1.0	0%	NA	0%	NA
	0.9	26%	6.85%	22%	7.47%
	0.8	67%	6.89%	100%	6.54%
	0.7	71%	6.87%	100%	6.67%
	0.6	81%	6.46%	100%	6.81%
	0.5	100%	6.66%	100%	6.87%
	0.4	100%	7.04%	100%	7.17%
	0.3	100%	7.28%	100%	7.35%
	0.2	89%	7.96%	78%	8.45%
	0.1	0%	NA	0%	NA
Adaptive α		96%	6.19%	100%	5.89%

B. Fixed-Outline Floorplanning

Table III compares the nonadaptive scheme and the adaptive scheme for fixed-outline floorplanning with area optimization alone (i.e., $\beta = 0$ in the cost function). We set the fixed-outline aspect ratio $R^* = 1$ and 2, and the maximum percentage of dead space $\Gamma = 10\%$ for this table.

For the nonadaptive scheme, we chose ten αs between zero and one. When α is below 0.3, the success probability decreases because the weight for area optimization is so small that the dead spaces of the resulting floorplans often exceed 10%. When α increases, the success probability becomes higher (100%) because the weight for area optimization is larger and, thus, the dead space decreases. However, the success probability decreases again if α is too large. The reason is that the aspect ratio of the final floorplan is far from the given outline and, thus, we cannot obtain a feasible solution efficiently. It also shows that for different R^*s, the optimal α is also different. Note that when $\alpha = 1$, it becomes classical outline-free floorplanning optimization. We found that it is harder to find a feasible solution by using the classical-floorplanning scheme.

For adaptive SA, we set α_{base} to 0.5 and used 500 most recent floorplans found to determine α dynamically, i.e., $n = 500$. From Table III, the average dead space using adaptive α is less than that using a constant α. As the results show that adaptive SA can achieve higher success probability and superior solution quality simultaneously.

Success Rates for Fixed-Outline Floorplanning: To test the effectiveness of our fixed-outline floorplanning algorithm, we set the maximum percentages of dead space Γ to 15%

TABLE IV
AVERAGE SUCCESS RATES AND RUNTIMES FOR GFA, PARQUET, AND OUR ALGORITHM UNDER DIFFERENT ASPECT RATIOS ON n100

dead space (Γ)	success rates/ average runtime			
	GFA [17]	Parquet-4.5(SP) [19]	Parquet-4.5(B*-tree) [19]	Ours
10%	30.3%/ 44.5s	65.5%/ 9.3s	99.4%/ 4.7s	100.0%/ 4.5s
15%	86.7%/ 22.4s	99.4%/ 9.4s	99.4%/ 2.8s	100.0%/ 2.5s

TABLE V
COMPARISON OF WIRELENGTH UNDER FIXED-OUTLINE CONSTRAINT FOR n100, n200, AND n300 WITH ASPECT RATIO $R^* = 1, 2, 3, 4$

Circuit	Aspect Ratio R^*	Parquet-4.5 (SP)		Ours	
		Wire (mm)	Time (sec)	Wire (mm)	Time (sec)
n100	1	33.56	30	32.06	26
	2	35.44	30	34.39	24
	3	35.48	30	34.23	27
	4	36.89	29	32.74	27
n200	1	63.55	175	58.33	150
	2	62.76	173	59.84	156
	3	63.70	180	61.55	156
	4	66.31	176	63.72	171
n300	1	76.05	399	71.00	363
	2	77.60	386	74.22	358
	3	81.67	391	79.56	371
	4	88.58	382	82.18	370
Comparison		1.06	1.11	1.00	1.00

and 10%. The expected aspect ratios R^* of the floorplans are chosen from the range [1], [3] with interval 0.5. Experiments were performed on a 1.6-GHz Intel Pentium 4 PC using the GSRC benchmark circuit n100. The results were averaged for 50 runs for each aspect ratio. We compared with GFA [13] and Parquet-4.5 [15] based on the same platform. We have tested Parquet-4.5 with two floorplan representations, SP, and B*-tree. Table IV lists the average success rates for GFA, Parquet-4.5 using SP, Parquet-4.5 using B*-tree, and ours. Our method obtained 100% success rates of fitting into the given fixed outlines for all runs with dead space $\Gamma = 15\%$ and $\Gamma = 10\%$. In contrast, the success rates when $\Gamma = 10\%$ for GFA, Parquet using SP, and Parquet using B*-tree were 30.3%, 65.5%, and 99.4%, respectively. The dramatic differences reveal the effectiveness of our approach. Also, our method required the least running time on average.

Wirelength Optimization for Fixed-Outline Floorplanning: Since the chip outline is given, the fixed-outline floorplanning problem should focus more on wirelength. Since GFA does not have wirelength-optimization mode, we cannot compare with GFA. We used the default wirelength-optimization parameters for Parquet. We compared with Parquet using SP representation since Parquet obtains better results with the SP-representation mode [4]. For fair comparison, the I/O pads for all circuits are fixed at the given coordinates in the benchmark. Table V shows the best wirelength for Parquet and our program. We used the GSRC benchmarks n100, n200, and n300, which contain 885, 1585, and 1893 nets, respectively, and reported the best results among ten runs. All the results listed in Table V can fit into the given outline (i.e., feasible solutions); thus, the dead spaces are not reported here. Our program can obtain better floorplan solutions than Parquet for all test cases; our program, on the average, can reduce the wirelength by about 6% using 11% less running time compared with Parquet.

All the above results show the efficiency and effectiveness of Fast-SA. Our method results in very stable and high-quality floorplan solutions. Fig. 16 shows the resulting floorplans for n100 with various aspect ratios.

C. BDF

We also performed experiments on BDF. The benchmarks are provided by Xiang *et al.* [26]; they are modified from the MCNC benchmark suite. The number of bus constraints ranges from 5 to 18. Each bus needs to go through 2–7 blocks according to the constraints. Our platform is a 2.8-GHz Intel Pentium 4 PC while the work [26] is on a 2.4-GHz Xeon PC; the speed difference between the two machines is marginal. The work [26] only reported dead spaces for the set of benchmarks. For fair comparisons, we optimized the same cost metric with area optimization alone.

We also implemented the soft-block resizing algorithm. The soft-macro-block adjustment is based on a simple, yet effective approach presented in [7]. Given M blocks, we assume that block b's bottom-left coordinate is $(b.x1, b.y1)$ and its top-right coordinate is $(b.x2, b.y2)$. Each soft block has four candidates for the block dimensions (i.e., shapes). The candidates are defined as follows:

1) $R_b = e.x_2 - b.x_1$, where $e.x_2 = \min\{g.x_2 | g.x_2 > b.x_2, g \in M\}$;
2) $L_b = d.x_2 - b.x_1$, where $d.x_2 = \max\{g.x_2 | g.x_2 < b.x_2, g \in M\}$;
3) $T_b = a.y_2 - b.y_1$, where $a.y_2 = \min\{g.y_2 | g.y_2 > b.y_2, g \in M\}$;
4) $B_b = c.y_2 - b.y_1$, where $c.y_2 = \max\{g.y_2 | g.y_2 < b.y_2, g \in M\}$.

After the candidates of the block shapes are determined, we may change the shape of a soft block b_i by choosing one of the following five choices during SA:

1) change the width of block b_i to R_i;
2) change the width of block b_i to L_i;
3) change the height of block b_i to T_i;
4) change the height of block b_i to B_i;
5) change the aspect ratio of block b_i to a random value in the range of the given soft aspect-ratio constraint.

Fig. 17 shows an example of resizing a soft block. Block b_3 has four shape candidates, R_3, L_3, T_3, and b_3. If we stretch

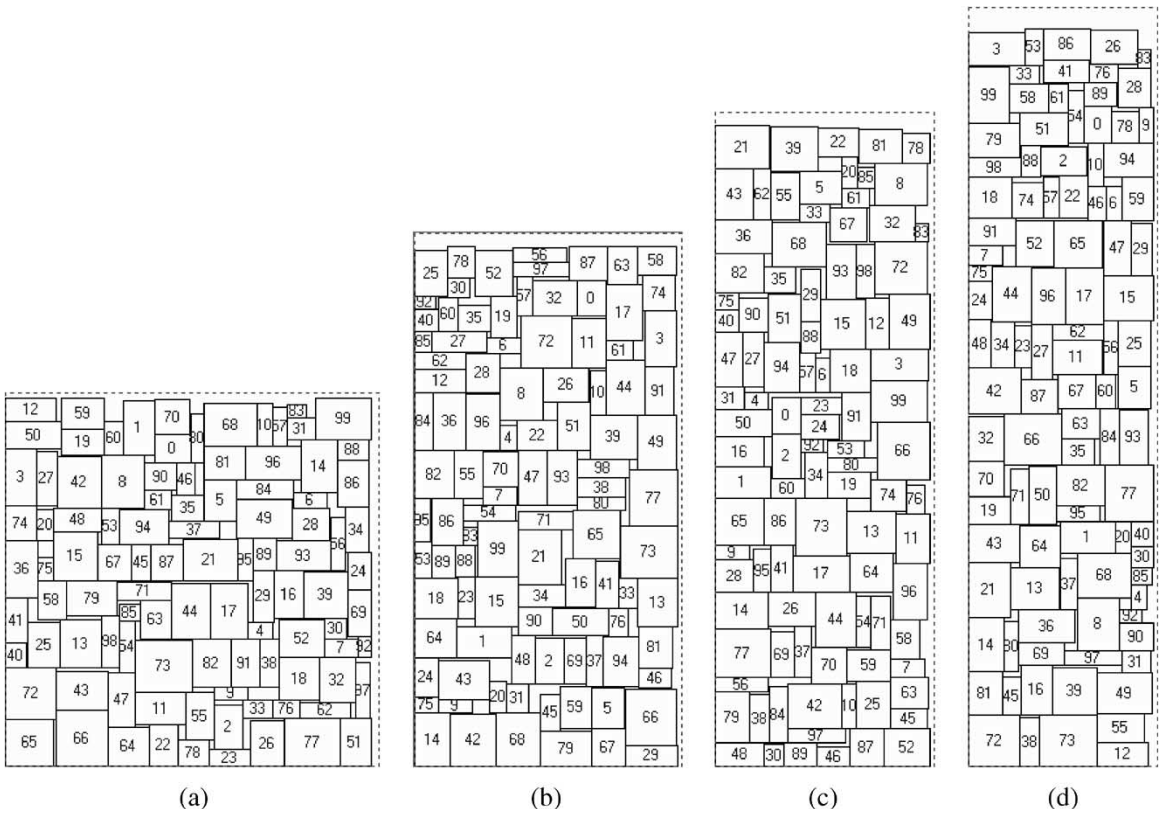

Fig. 16. GSRC n100 floorplan results with $\Gamma = 10\%$, and desired aspect ratios R^*s are (a) 1, (b) 2, (c) 3, and (d) 4. Dead spaces are (a) 5.57%, (b) 5.06%, (c) 5.03%, and (d) 4.70%. Dotted line is fixed outline.

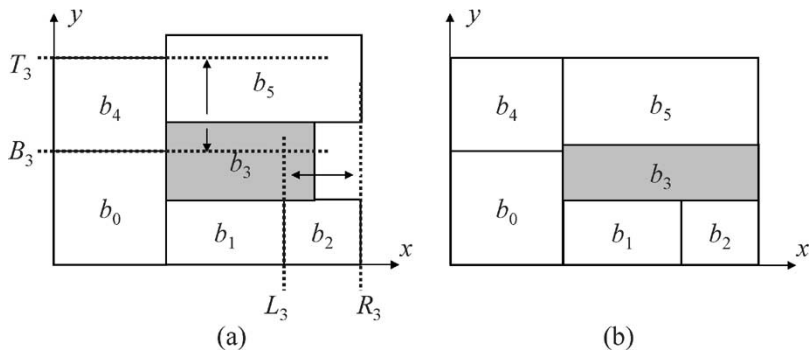

Fig. 17. Soft-block resizing example. (a) There are four shape candidates for b_3 for resizing. (b) Stretch right boundary of b_3 to R_3, and resulting floorplan becomes more compact.

the right boundary of b_3 to R_3, it can generate a more compact floorplan.

The block shapes could be changed to obtain a more compact floorplan during SA. Table VI summarizes the BDF results for the method used in [26], ours using the classical SA and Fast-SA. Ours using classical SA outperforms the method used in [26]. Using Fast-SA, we obtained the smallest dead space in the shortest average running time. For hard blocks, our average dead space (using Fast-SA) is 4.38% while the work in [26] is 5.51%. We only needed 26 s on average while that in [26] required 104 s. For soft blocks, since the previous work [26] resizes the blocks from the existing solutions, 2 s are added to the average runtime (106 s in total). Our method performs resizing and floorplanning at the same time; therefore, the runtimes are longer than hard-block floorplanning alone (but is still much faster than the work in [26]). The average runtime is 47 s, and the average dead space is 0.41%, compared to 0.91% required by the previous work. In short, our algorithm can obtain better bus-driven floorplan solutions for all test cases in shorter running times. Figs. 18 and 19 show the resulting floorplans for ami33-2 and ami49-3, respectively.

VII. Conclusion

We have proposed algorithms for the modern floorplanning problems with fixed-outline and bus constraints, based on our new Fast-SA scheme and the B*-tree representation. Experimental results have shown that our Fast-SA leads to faster and more stable convergence to the desired floorplan solutions. For fixed-outline floorplanning, the new cost function

TABLE VI
BDF RESULTS. RESULTS OF SP ON 2.4-GHZ INTEL XEON PC, WHILE OURS ARE ON 2.8-GHZ INTEL PENTIUM 4 PC

Block type	Circuit	Block	Bus	SP [30]		Ours (Classical SA)		Ours (Fast-SA)	
				Time (sec)	Dead space	Time (sec)	Dead space	Time (sec)	Dead space
Hard	apte	9	5	11	4.11%	3	1.77%	2	1.59%
	xerox	10	6	12	3.88%	7	3.85%	5	3.85%
	hp	11	14	28	5.02%	21	4.07%	20	4.47%
	ami33-1	33	8	61	6.02%	63	6.95%	19	5.69%
	ami33-2	33	18	81	6.10%	78	4.93%	22	3.87%
	ami49-1	49	9	98	5.42%	56	4.82%	28	5.34%
	ami49-2	49	12	278	6.09%	76	6.90%	43	5.45%
	ami49-3	49	15	265	7.40%	72	6.94%	66	4.74%
	Average			104	5.51%	45	4.90%	26	4.38%
	Ratio			1.00	1.00	0.44	0.91	0.25	0.79
Soft	apte	9	5	12	0.72%	6	0.61%	3	0.02%
	xerox	10	6	13	0.95%	6	0.24%	6	0.10%
	hp	11	14	28	0.62%	23	0.00%	11	0.03%
	ami33-1	33	8	62	0.94%	53	0.84%	30	0.33%
	ami33-2	33	18	86	1.27%	83	0.74%	73	0.73%
	ami49-1	49	9	101	0.85%	109	0.35%	58	0.51%
	ami49-2	49	12	281	0.84%	223	0.78%	112	0.67%
	ami49-3	49	15	268	1.09%	125	1.02%	81	0.92%
	Average			106	0.91%	79	0.57%	47	0.41%
	Ratio			1.00	1.00	0.74	0.63	0.44	0.45

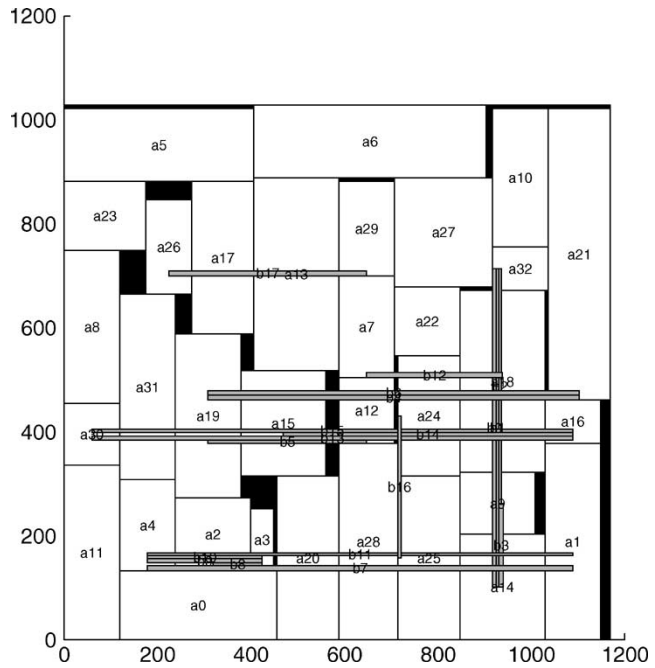

Fig. 18. Resulting packing of ami33-2 with hard blocks. There are 33 blocks and 18 buses. Buses are {9, 14, 32}, {14, 32}, {9, 32}, {9, 14}, {12, 19, 21}, {12, 19}, {19, 21}, {1, 2, 3, 4}, {2, 3}, {3, 4}, {2, 3, 4}, {1, 2, 4}, {7, 18}, {15, 16, 30}, {15, 16}, {16, 30}, {24, 25}, and {26, 29}.

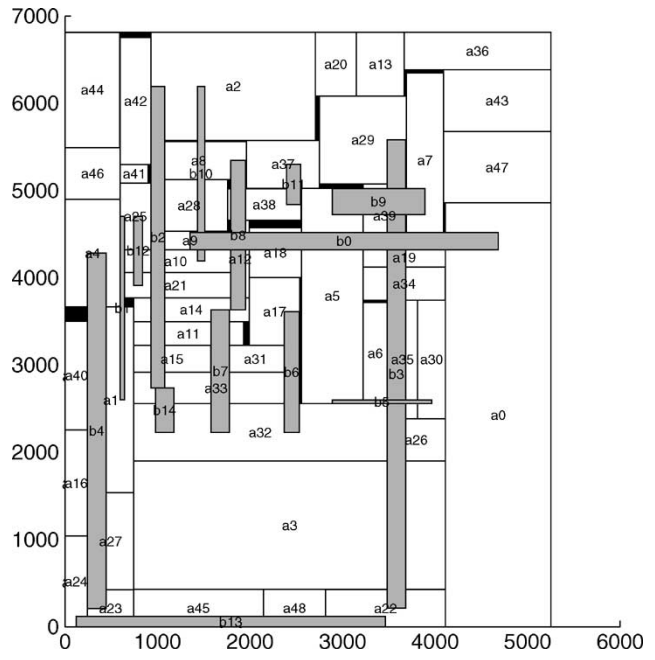

Fig. 19. Resulting packing of ami49-3 with soft-block adjustment. There are 49 blocks and 15 buses. Buses are {0, 5, 9, 12, 18}, {1, 10, 21, 25}, {2, 28, 33}, {3, 19, 22, 26, 29, 34}, {4, 23, 27}, {5, 35, 30, 6}, {32, 31, 17}, {11, 14, 15, 32, 33}, {12, 8, 14}, {5, 7, 39}, {2, 8, 9, 10}, {37, 38}, {10, 21, 25}, {22, 23, 24}, and {32, 33}.

considering the aspect-ratio penalty drives SA more efficiently to find floorplans inside the given chip outline. For BDF, we have demonstrated that the dummy blocks can fix the vertical block-falling-down problem during the packing of a B*-tree. Thus, the dummy-block inserting technique for B*-tree can be effectively used to solve position constraints, especially for modern floorplanning problems. The experimental results on both fixed-outline floorplanning and the BDF have shown the efficiency and effectiveness of our floorplanning algorithms; for those applications, our results outperform the related recent works by large margins.

We should note that the proposed three-stage fast annealing scheme readily applies to other optimization problems. Though beyond the scope of this paper, we have also implemented the scheme to handle the classical two-way bipartitioning problem. Our results show significant speedups over the classical SA scheme and better solution quality. Research along this direction is ongoing.

ACKNOWLEDGMENT

The authors would like to thank Prof. I. Markov, S. N. Adya, Dr. H. Xiang, X. Tang, M. D. F. Wong, Prof. D.-S. Chen, C. T. Lin, and Y. W. Wang for providing the authors with benchmarks and/or programs for the comparative studies.

REFERENCES

[1] E. H. L. Aarts and P. J. M. van Laarhoven, "A new polynomial time cooling schedule," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1985, pp. 206–208.

[2] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. IEEE Int. Conf. Computer Design*, Austin, TX, 2001, pp. 328–334.

[3] ——, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[4] H. H. Chan, S. N. Adya, and I. L. Markov, "Are floorplan representations important in digital design?," in *Proc. ACM Int. Symp. Physical Design*, San Francisco, CA, Apr. 2005, pp. 129–136.

[5] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proc. ACM/IEEE Design Automation Conf.*, Los Angeles, CA, Jun. 2000, pp. 458–463.

[6] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on fast simulated annealing," in *Proc. ACM Int. Symp. Physical Design*, San Francisco, CA, Apr. 2005, pp. 104–112.

[7] J.-C. Chi and M. C. Chi, "A block placement algorithm for VLSI circuits," *Chung Yuan J.*, vol. 31, no. 1, pp. 69–75, Mar. 2003.

[8] *GSRC Floorplan Benchmark Suite*. [Online]. Available: http://www.cse.ucsc.edu/research/surf/GSRC/GSRCbench.html

[9] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplans and its applications," in *Proc. ACM/IEEE Design Automation Conf.*, New Orleans, LA, Jun. 1999, pp. 268–273.

[10] A. B. Kahng, "Classical floorplanning harmful?," in *Proc. ACM Int. Symp. Physical Design*, San Diego, CA, Apr. 2000, pp. 207–213.

[11] S. Kirpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[12] J. H. Y. Law and E. F. Y. Young, "Multi-bend bus driven floorplanning," in *Proc. ACM Int. Symp. Physical Design*, San Francisco, CA, Apr. 2005, pp. 113–120.

[13] C.-T. Lin, D.-S. Chen, and Y.-W. Wang, "Robust fixed-outline floorplanning through evolutionary search," in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2004, pp. 42–44.

[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1995, pp. 472–479.

[15] *Parquet: Fixed-Outline Floorplanner*. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/parquet/

[16] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Bus-based integrated floorplanning," in *Proc. IEEE Int. Symp. Circuits and Systems*, Phoenix-Scottsdale, AZ, May 2002, pp. 875–878.

[17] ——, "Integrated floorplanning with buffer/channel insertion for bus-based microprocessor designs," in *Proc. ACM Int. Symp. Physical Design*, Del Mar, CA, Apr. 2002, pp. 875–878.

[18] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. Singapore: World Scientific, 1999.

[19] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SSC-20, no. 2, pp. 510–522, Apr. 1985.

[20] ——, "TimberWolf3.2: A new standard cell placement and global routing package," in *Proc. IEEE Design Automation Conf.*, Las Vegas, NV, 1986, pp. 432–439.

[21] C. Sechen, "Chip-planning, placement, and global routing of macro/custom cell integrated circuits using simulated annealing," in *Proc. IEEE Design Automation Conf.*, Anaheim, CA, 1988, pp. 73–81.

[22] W. Swartz, "Automatic layout of analog and digital mixed macro/standard cell integrated circuits," M.S. thesis, Dept. Elect. Eng., Yale Univ., New Haven, CT, 1993. [Online]. Available: http://www.internetcad.com/thesis.htm

[23] X. Tang and D. F. Wong, "Floorplanning with alignment and performance constraints," in *Proc. ACM/IEEE Design Automation Conf.*, New Orleans, LA, Jun. 2002, pp. 848–853.

[24] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design*. Boston, MA: Kluwer, 1988.

[25] M.-C. Wu and Y.-W. Chang, "Placement with alignment and performance constraints using the B*-tree representation," in *Proc. IEEE Int. Conf. Computer Design*, San Jose, CA, Sep. 2004, pp. 568–571.

[26] H. Xiang, X. Tang, and M. D. F. Wong, "Bus-driven floorplanning," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2003, pp. 66–73.

Tung-Chieh Chen (S'04) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 2003. He is currently working toward the Ph.D. degree at the Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan, R.O.C.

His current research interests include very large scale integration (VLSI) floorplanning and placement.

Yao-Wen Chang (S'94–A'96–M'96) received the B.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1988, and the M.S. and Ph.D. degrees from the University of Texas, Austin in 1993 and 1996, respectively, all in computer science.

Currently, he is a Professor of the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University. He is currently also a Visiting Professor at Waseda University, Japan. He was with IBM T.J. Watson Research Center, Yorktown Heights, NY, in summer 1994. From 1996 to 2001, he was on the faculty of National Chiao Tung University, Taiwan. His current research interests include VLSI physical design, design for manufacturing, and FPGA.

Dr. Chang received the Best Paper Award at ICCD-1995 and six Best Paper Nominations from DAC-2005, 2004 ACM TODAES, ASP-DAC-2003, ICCAD-2002, ICCD-2001, and DAC-2000. He has received many awards for distinguished research, such as the 2005 First-Class Principal Investigator Award and the 2004 Mr. Wu Ta You Memorial Award from the National Science Council of Taiwan, and for excellent teaching from National Taiwan University and National Chiao Tung University. He currently serves on the technical program committees of a few important conferences on VLSI design automation and circuit design, including APCCAS, ASP-DAC (topic chair), DATE, ICCAD, ICCD, ISPD, SOCC, and VLSI-DAT. He is currently the Chair of the Electronic Design Automation (EDA) Consortium of the Ministry of Education, Taiwan, a member of board of governors of Taiwan IC Design Society, and a member of the IEEE Circuits and Systems Society, ACM, and ACM/SIGDA.