Modern Floorplanning Based on Fast Simulated Annealing [*]

Tung-Chieh Chen
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei 106, Taiwan

Yao-Wen Chang
Graduate Institute of Electronics Engineering
and Department of Electrical Engineering
National Taiwan University
Taipei 106, Taiwan

ABSTRACT

Unlike classical floorplanning that usually handles only block packing to minimize silicon area, modern 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. We study in this paper two types of modern floorplanning problems: (1) fixed-outline floorplanning and (2) bus-driven floorplanning. Our floorplanner uses the B*-tree floorplan representation and is based on a fast three-stage simulated annealing scheme, called Fast-SA. For fixed-outline floorplanning, we present an adaptive Fast-SA that can dynamically change the weights in the cost function to optimize wirelength under the outline constraint. Experimental results show that our floorplanner can achieve almost 100% success rates efficiently for fixed-outline floorplanning with various aspect ratios, compared to 10%–90% success rates obtained by the most recent works. For the bus-driven floorplanning, we explore the feasibility conditions of the B*-tree with the bus constraints and develop a bus-driven floorplanning algorithm based on the conditions and Fast-SA. Experimental results show that our floorplanner on the average reduces 20% (55%) dead space for the floorplanning with hard (soft) macro blocks, compared with the most recent work. In particular, our floorplanner is more efficient than the previous works.

Categories and Subject Descriptors: B.7.2 [Integrated Circuits]: Design Aids

General Terms: Algorithms, Experimentation, Performance

Keywords: Floorplanning, Simulated Annealing

1. INTRODUCTION

As the design complexity increases dramatically, modern 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 representation [14], Adya and Markov [2, 4] first presents new objective functions to drive simulated annealing and new types of moves that better guide local search for fixed-outline floorplanning. Lin et al. [13] applies evolutionary search to handle fixed-outline floorplanning based on the normalized polish expression [20].

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, 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 interconnect 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 bus routing. To make bus routing easier, we shall consider the bus planning earlier in the floorplanning stage [22].

Floorplanning with the alignment constraint is closely related to bus-driven floorplanning. The alignment constraint is considered in [19] and [21]. 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 bus-driven floorplanning. The bus defined in their works is composed of wires connecting only two blocks, which is not general for real bus designs. The general BDF that allows a bus to connect multiple blocks is first studied in [22]. In the 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. [22] proposed an algorithm based on the sequences pair (SP) representation. Nevertheless, the SP representation incurs a larger solution space, and thus it is less efficient to find a high-quality solution.

We study in this paper two types of modern floorplanning problems: (1) fixed-outline floorplanning and (2) bus-driven floorplanning. Our floorplanner uses the B*-tree floorplan representation [5] and is based on a fast three-stage simulated annealing scheme, called Fast-SA. The Fast-SA is sig-

[*]This work was partially supported by SpringSoft, Inc. and National Science Council of Taiwan under Grant No's. NSC 93-2215-E-002-009, NSC-93-2220-E-002-001, and NSC 93-2752-E-002-008-PAE. Emails: tungchieh@ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw

nificantly different from existing simulated annealing schemes that try to speed up the annealing process, e.g., the well-known TimberWolf [8] that uses a two-stage technique 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 achieves an average 12X speedup over both of 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 an average success rate of 100% (99.7%) for the fixed-outline floorplanning with a dead space of 15% (10%) and various aspect ratios, compared to the average success rates of 78% and 85% obtained by Parquet-2.1 [15] and [13], respectively.

For the bus-driven floorplanning, 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. [22], our method on the average reduces 20% (50%) dead space for the floorplanning with hard (soft) blocks. In particular, our floorplanner is more efficient than the previous works.

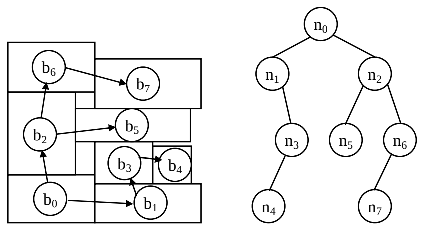
The remainder of this paper is organized as follows. Section 2 reviews the B*-tree floorplan representation. Section 3 presents the Fast-SA scheme. Section 4 copes with the fixed-outline floorplanning based on the adaptive Fast-SA. Section 5 deals with BDF based on Fast-SA. The experimental results are reported in Section 6. Finally, we give conclusions in Section 7.

2. REVIEW OF THE B*-TREE REPRESENTATION

A B*-tree [5] is an ordered binary tree for modelling non-slicing 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].

Figure 1 shows an admissible placement and its corresponding B*-tree. A B*-tree is an ordered binary tree whose root corresponds to the block on the bottom-left corner. Similar to the 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 denotes 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 block in R_i that is unvisited. 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 $(x_{root}, y_{root}) = (0, 0)$. 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.



(a) (b)

Figure 1: (a) An admissible placement. (b) The B*-tree representing the placement.

3. FAST SIMULATED ANNEALING SCHEME

Simulated annealing (SA) [11] is widely used for floorplanning. It is an optimization scheme with non-zero 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 $\min\{1, e^{-\Delta C/T}\}$, 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 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 [8] 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 Figure 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 Simulated Annealing (Fast-SA) process to integrate the random search with hill climbing more efficiently. Unlike the classical SA [11] and the TimberWolf SA [8], the annealing process consists of three stages: (1) The high-temperature random search stage, (2) the pseudo-greedy local search stage, and (3) the hill-climbing search stage. At the first stage, we let temperature $T \rightarrow \infty$ so that the probability of accepting an inferior solution approaches 1. The process is like a random search to find the best solution. At the second stage, we let $T \rightarrow 0$. Since the temperature is very low, we only accept a very small number of inferior solutions, which is like a greedy local search. We call this process the pseudo-greedy local search stage. The third stage is the hill-climbing search stage. The temperature raises again to facilitate the hill climbing. Thus, it can escape from the local minimum and search for better solutions. The temperature reduces gradually, and very likely it finally converges to a globally optimal solution.

Since the new simulated annealing 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 the annealing much more efficient and effective. To implement the annealing scheme, we define the temperature T of the Fast-SA by the following equations:

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

Here, n is the number of iterations, Δ_{avg} is the average up-hill cost, P is the initial probability to accept up-hill so-

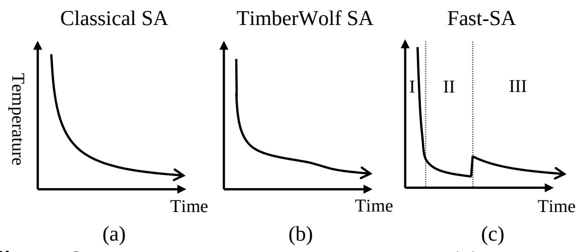


Figure 2: Temperature vs. search time for (a) classical simulated annealing, (b) TimberWolf SA, and (c) Fast-SA. The temperature of TimberWolf SA drops faster than that of classical SA at the beginning and the ending iterations, but slower in the middle iterations. The Fast-SA consists of three stages.

lutions, $\langle\Delta_{cost}\rangle$ 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. Since P is usually set close to 1, so it performs random search to find a good solution. Then, it enters the pseudo-greedy 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 $T \rightarrow 0$ so that it only accepts good solutions to perform pseudo-greedy searches. After k iterations, the temperature jumps up to further improve the solution quality. The value of $\langle\Delta_{cost}\rangle$ affects the reduction rate of the temperature. If the cost of a neighboring solution changes significantly, $\langle\Delta_{cost}\rangle$ is larger and thus the temperature reduces slower. In contrast, if $\langle\Delta_{cost}\rangle$ 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 1, so $\langle\Delta_{cost}\rangle < 1$, and it ensures the decreasing temperature. The behavior of the temperature changes is illustrated in Figure 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., $T_1 = \Delta_{avg}/\ln P$. The initial temperature T_1 needs to be kept high to avoid getting bogged 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:

- Op1: Rotate a block.
- Op2: Move a node/block to another place.
- Op3: Swap two nodes/blocks.
- 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 a floorplan:

$$Cost = \alpha\frac{A}{A_{norm}} + (1-\alpha)\frac{W}{W_{norm}}, \tag{2}$$

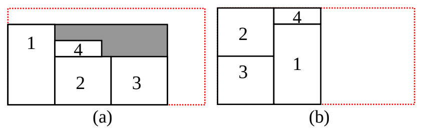


Figure 3: Two feasible floorplans. The dotted line is the fixed outline, and the fixed-outline aspect ratio, R^*, is 0.5. The floorplan aspect ratio R is (a) 0.5, and (b) 1.

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.

4. 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.

4.1 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. Since a floorplanner can change the orientations of individual blocks, we choose $R^* \geq 1$. The height H^* and width W^* of the outline is defined by the following equations [2]:

$$H^* = \sqrt{(1+\Gamma)AR^*}, \quad W^* = \sqrt{(1+\Gamma)A/R^*}. \tag{3}$$

4.2 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 3. For some blocks, they only have one feasible orientation to fit into the fixed outline. We mark all such blocks as non-rotatable 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 simulated annealing. After the temperature cools down enough, we terminate the simulated annealing process and report the best floorplan.

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 the following equation:

$$\Phi(F) = \alpha A + \beta W + (1-\alpha-\beta)(R-R^*)^2 \tag{4}$$

where A is the current floorplan area, W is the current wirelength, R is the current floorplan aspect ratio, R^* is desired floorplan aspect ratio, and α, β are user-defined parameters. As the example shown in Figure 3, the best aspect ratio of the floorplan in the fixed outline is not 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.

```
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 modules;
8       Evaluate the B*-tree cost;
9       Modify the weights in the objective function;
10      Update $T$;
11  until converged or cooling down;
12  return the best solution.
```

Figure 4: The adaptive simulated annealing for fixed-outline floorplanning.

4.3 Adaptive Simulated Annealing

We focus on area optimization 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. So we use an adaptive method to control α according to n most recent floorplans found. The area weight α is defined by the following equation:

$$\alpha = \alpha_{base} + \left(1 - \frac{n_{feasible}}{n}\right) \quad (5)$$

where $n_{feasible}$ is the number of feasible solutions in n most recent floorplan solutions, and α_{base} is determined by the user, say $\alpha_{base} = 0.5$. Once α is determined, the weight of the aspect-ratio penalty is also determined. The experimental results are reported in Section 6.2.

4.4 Algorithm

Figure 4 summarizes our algorithm. First, we mark all non-rotatable 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. If the floorplan is better than the current best one, we record it as the best floorplan. Then, we decrease the temperature T and update the weights in the objective function. This process continues to the end of simulated annealing, and the best solution is reported.

5. BUS-DRIVEN FLOORPLANNING

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.

5.1 Bus-Driven Floorplanning 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 bus-driven floorplanning (BDF) is defined as follows [22]:

Given n rectangular macro blocks $B = \{b_i | i = 1, ..., n\}$ and m buses $U = \{u_i | i = 1, ..., m\}$, 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.

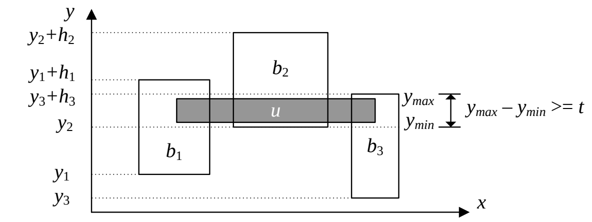


Figure 5: A feasible horizontal bus $u = < H, t, \{b_1, b_2, b_3\} >$.

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

5.2 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. $x_j = x_i + w_i$). So, the blocks has horizontal relationships in a left-skewed sub-tree.

PROPERTY 1. *In a B*-tree, the nodes in a left-skewed sub-tree may satisfy a horizontal bus constraint.*

Blocks are compacted to the bottom and left after packing. So the blocks associated with a left-skewed sub-tree 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 Figure 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 Figure 6 (b), we adjust the heights of dummy blocks to shift the displaced blocks to satisfy the bus constraint. After adjusting the heights of 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 the following equation:

$$\Delta_i = \begin{cases} (y_{min} + t) - (y_i + h_i) & \text{if } (y_{min} + t) > (y_i + h_i) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where x_i (y_i) is the $x(y)$-coordinate of block b_i, and $y_{min} = \max\{y_i | i = 1, 2, ..., k\}$ for a bus $\{b_1, ..., b_k\}$. Figure 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 a left-skewed sub-tree.*

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 sub-tree are aligned with the x-coordinate. Assume the minimum width of the modules that the bus goes through is larger than the bus width. The structure forms a vertical bus. In the example shown in Figure 8, the nodes n_3 and n_5 is in the right-skewed sub-tree 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 a right-skewed sub-tree can guarantee the feasibility of a vertical bus.*

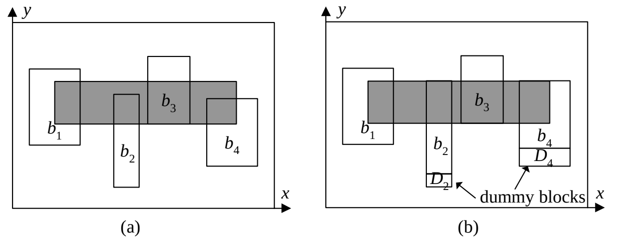


Figure 6: (a) An infeasible floorplan since the block-overlap range is less than the bus width t. Inserting dummy blocks, the bus $< H, t, \{b_1, b_2, b_3, b_4\} >$ is satisfied.

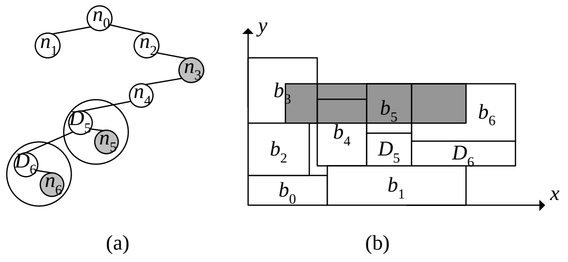


Figure 7: (a) The B*-tree with a left-skewed sub-tree after inserting dummy nodes. (b) The corresponding feasible horizontal bus $< H, t, \{b_3, b_5, b_6\} >$.

5.3 The Twisted-Bus Structure

Consider two buses simultaneously, we cannot always fix the horizontal bus constraints by inserting dummy blocks. As the example shown in Figure 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. In a B*-tree, if the nodes of two buses are both in the two right-skewed sub-trees, it incurs a twisted-bus structure and cannot be solved by inserting dummy blocks. Therefore, we shall discard a B*-tree with such an infeasible tree topology during solution perturbation. Figure 9 shows a twisted-bus structure where n_3 is in the right-skewed sub-tree of n_2, and n_6 is in the right-skewed sub-tree of n_0.

5.4 Bus-Overlapping

When multiple buses are considered, we need to avoid overlaps between buses. For example, in Figure 10, two horizontal buses are to be assigned. The buses $u = \{b_0, b_4\}$ ($v = \{b_2, b_3\}$) is 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 b_2, and insert a dummy block D_2 right below b_2. Thus, the two buses can be assigned at the same time

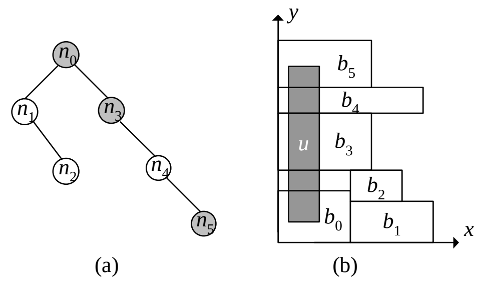


Figure 8: (a) The B*-tree with a right-skewed sub-tree. (b) The corresponding feasible vertical bus $u =< V, t, \{b_0, b_4, b_5\} >$.

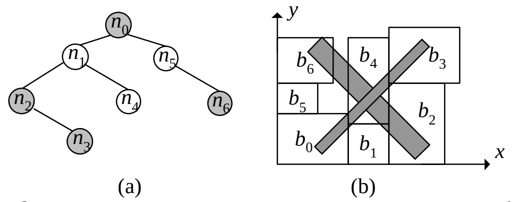


Figure 9: An infeasible floorplan for two buses, $u = \{b_0, b_3\}$ and $v = \{b_2, b_6\}$. (a) A twisted-bus structure where n_3 is in the right sub-tree of n_2, and n_6 is in the right sub-tree of n_0. (b) The corresponding floorplan. The two twisted-bus cannot be satisfied simultaneously by inserting dummy blocks.

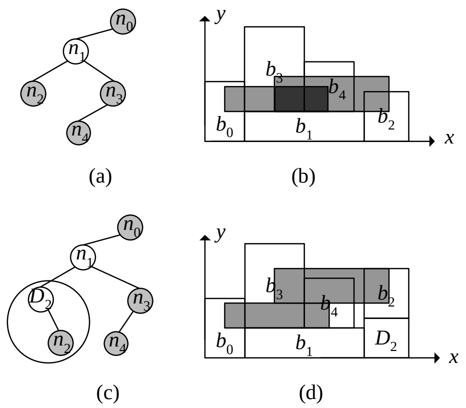


Figure 10: Two horizontal buses, $u = \{b_0, b_4\}$ and $v = \{b_2, b_3\}$. (a) Two buses overlap. (b) By inserting a dummy block, we can get a feasible floorplan without bus-overlapping.

by inserting the dummy block.

5.5 Algorithm

Our bus-driven floorplanning algorithm applies Fast-SA based on the B*-tree representation. Since the objective function of bus-driven floorplanning 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:

$$\Psi(F, U) = \alpha A + \beta B + \gamma M \tag{7}$$

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.

Figure 11 summarizes our algorithm. First, we initialize the B*-tree as a complete binary tree and start with the Fast-SA process. After each perturbation and non-dummy 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. After adjusting the heights of dummy blocks, we pack the B*-tree again. Then, we perform bus-planning to determine the locations of buses. We also check bus overlapping so that no two buses overlap. During the floorplan evaluation, the cost can be determined by the chip area A, bus area B of feasible buses, and the number of unassigned buses M. In the simulated annealing process, we record the floorplan solution with the most number of feasible buses and the lowest cost. After the simulated annealing 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.

```
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 modules 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;
9       Pack modules with dummy blocks;
10      Decide bus locations;
11      Evaluate the floorplan cost;
12  until converged or cooling down;
13  return the best solution.
```

Figure 11: The bus-driven floorplanning algorithm.

6. 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.

6.1 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 [7], 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 1.6GHz PC with 256 MB memory. All of the simulated annealing algorithms are based on the B*-tree floorplan representation and the same initial temperature. The initial probability 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 given below:

$$T_{new} = \lambda T_{old}, \ \ 0 < \lambda < 1. \tag{8}$$

The value of λ for classical SA is set to a fixed value 0.85 [18]. For TimberWolf SA [8], 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 annealing schedule of Fast-SA was described in the Section 3.

Table 1 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% deadspace in this experiments) 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 5.3X 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; so the convergence speed is even higher. The 3rd stage of Fast-SA can avoid getting bogged in a local minimum in the 2nd stage of Fast-SA. On the average, Fast-SA achieved a 12X speedup in finding floorplan solutions of comparable areas.

Figure 12 compares the convergence speed and stability of the three SA schemes. For each SA scheme, the area is plotted as a function of running time. We ran the n100 circuit for 10 times for each SA scheme. 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 MCNC benchmark ami49. In Figure 13, the dead space is plotted as a function of the running time. As shown in Figure 13 and Table 2, the convergence speed of the greedy search is the fastest; it took only 0.234 seconds to find a floorplan solution of less than 10% dead space. However, the final solution for greedy search has dead space of 5.76%. The classical (TimberWolf) simulated annealing method can further improve the solution quality until the dead space equals 2.62% (2.13%). Since Fast-SA combines the pseudo-greedy local search stage and the hill-climbing stage, its convergence speed is much faster than that of classical SA. The Fast-SA only spent 0.625 seconds to obtain a floorplan solution of 5% dead space while classical SA needed 8.687 seconds. The Fast-SA spent more iterations to find better floorplan solutions with dead spaces under 5%. The 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

Table 1: Dead space and CPU time for different simulated annealing schemes using the GSRC floorplan benchmark suite. WS is the deadspace (%).

	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

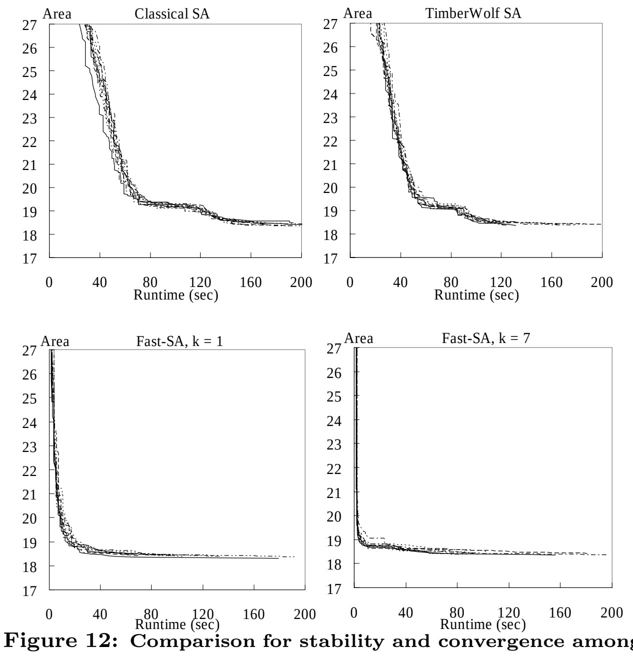


Figure 12: Comparison for stability and convergence among classical SA, TimberWolf SA, Fast-SA with k=1, and Fast-SA with k=7. (Note that the initial area is 69.76 mm^2, same for the four SA schemes. We focus on the results with area smaller than 27 mm^2 to examine the effect more closely.)

Table 2: Dead spaces and runtimes for different simulated annealing 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

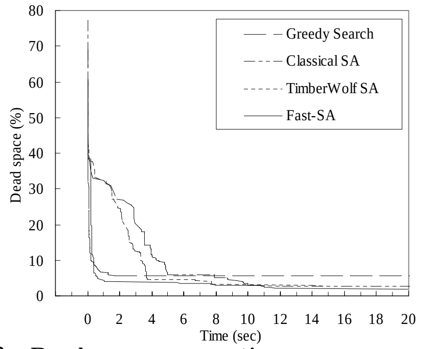


Figure 13: Dead space vs. runtime among greedy search, classical SA, TimberWolf SA, and Fast-SA. The respective final dead spaces are 5.76%, 2.62%, 2.13%, and 2.00% for greedy search, classical SA, TimberWolf SA, and Fast-SA.

a major concern, and Fast-SA is the best choice for the floorplanning problem addressed here (it achieved 13.9X speedup over classical SA for finding a floorplan solution of less than 5% dead space for this case).

6.2 Fixed-Outline Floorplanning

Table 3 compares the non-adaptive scheme and the adaptive scheme for the fixed-outline floorplanning with area optimization alone (i.e., $\beta = 0$ in the objective 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 non-adaptive scheme, we chose 10 α's between 0 and 1. 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 a classical outline-free floorplanning problem. We found that it is harder to find a feasible solution by using the classical floorplanning scheme.

For adaptive simulated annealing, we set α_{base} to 0.5 and used 500 most recent floorplans found to determine α dynamically, i.e., $n = 500$. From Table 3, the average dead space by using adaptive α is less than that by using a constant α. As the results show that adaptive simulated annealing can achieve higher success probability and superior solution quality simultaneously.

To test the effectiveness of our fixed-outline floorplanning algorithm, we set the maximum percentages of dead space Γ to 15% and 10%. The expected aspect ratios R^* of the floorplans are chosen from the range [1, 4]. Experiments were performed on a 1.6GHz Intel Pentium 4 PC using the GSRC benchmark circuit n100. The results were averaged for 50

Table 3: Success probability and average dead space using constant α and adaptive α on n100, $\Gamma = 10\%$.

		$R^* = 1$		$R^* = 2$	
α		Success prob.	Average dead space	Success prob.	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%

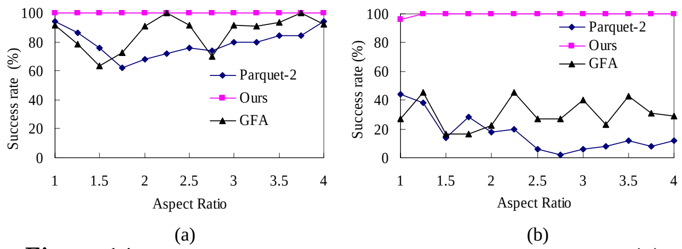


Figure 14: Success probability v.s. aspect ratio on n100, (a) $\Gamma = 15$, and (b) $\Gamma = 10\%$.

runs for each aspect ratio. We compared with Parquet-2 [15] and GFA [13] based on the same platform. The Parquet-2 and the GFA programs are provided by the authors of [15] and [13], respectively.

Figure 14 plots the success probability of satisfying the fixed-outline constraints vs. the desired aspect ratio of the fixed outline with $\Gamma = 15\%$ and $\Gamma = 10\%$, respectively. Note that we set the maximum running time to 100 seconds and used the *default* SA parameters given in Parquet-2. When $\Gamma = 15\%$, our method always achieved 100% success rates of fitting into the given fixed outlines while the success rates for Parquet-2 and GFA were about 60%–90%. When $\Gamma = 10\%$, our algorithm still achieved 100% success rates, except for the case with aspect ratio equal to one (for which the success rate is 96%). In contrast, the success rates for Parquet-2 and GFA were consistently under 50%. The dramatic differences reveal the effectiveness of our approach.

Table 4 lists the average success rates, average dead spaces, and average runtimes for Parquet-2, GFA, and Fast-SA for n100, $\Gamma = 10\%$ under different aspect ratios in [1, 4]. The average success rate for Fast-SA is much higher than Parquet-2 and GFA (99.7% vs. 16.6% and 30.3%, respectively). The average dead space for Fast-SA is the smallest (5.79% vs. 7.32% and 6.26%), and Fast-SA used the least time on average (27.6 sec vs. 40.2 sec and 44.5 sec).

The fixed-outline floorplanning problem should emphasize more on wirelength. Since GFA does not have wirelength minimization mode, so we compared our program with Parquet-2.1. We modified the stopping criterion of Parquet-2.1 to search for more solutions after finding the

Table 4: The average success rates, dead space, and runtime for Parquet-2, GFA, and Fast-SA under different aspect ratios (n100, $\Gamma = 10\%$).

n100, $\Gamma = 10\%$	Parquet-2	GFA	Fast-SA
Avg. success rate	16.6%	30.3%	99.7%
Avg. dead space	7.32%	6.26%	5.79%
Avg. dead space ratio	1.26	1.08	1.00
Avg. runtime (sec)	40.2	44.5	27.6
Avg. runtime ratio	1.46	1.61	1.00

Table 5: The comparison of the wirelength under the fixed-outline constraint for ami33 and ami49 with $R^* = 1, 2, 3, 4$.

Circuit	Aspect Ratio R^*	Parquet-2.1		Ours	
		Wire (mm)	Time (sec)	Wire (mm)	Time (sec)
ami33	1	64.6	23	46.3	16
	2	65.9	24	48.9	11
	3	80.9	23	67.7	15
	4	72.7	24	61.4	14
Average		71.0	24	56.1	14
Comparison		1.27	1.71	1.00	1.00
ami49	1	753	25	752	17
	2	792	25	739	18
	3	964	25	858	18
	4	989	25	787	20
Average		875	25	784	18
Comparison		1.12	1.39	1.00	1.00

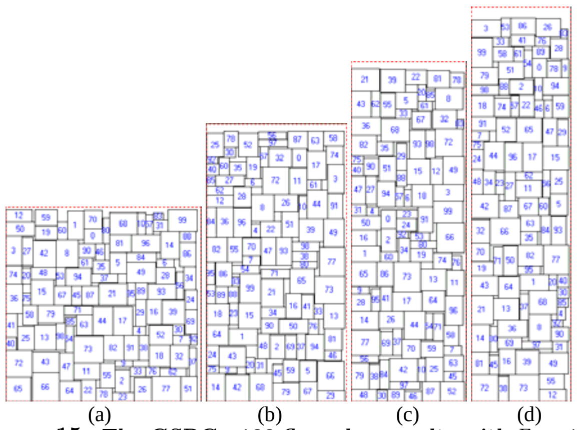


Figure 15: The GSRC n100 floorplan results with $\Gamma = 10\%$ and the desired aspect ratios R^*'s are (a) 1, (b) 2, (c) 3, (d) 4. The dead spaces are (a) 5.57%, (b) 5.06%, (c) 5.03%, and (d) 4.70%. The dotted line is the fixed outline.

first solution within the bounding box. The maximum runtime was set to 30 seconds, and we used the default wirelength optimization parameters for Parquet-2.1. Table 5 shows the best wirelength for Parquet-2.1 and our program. We used the MCNC benchmark ami33 and ami49 which contain 123 and 408 nets respectively, and reported the best results among 10 runs. All the results listed in Table 5 can fit into the given outline (i.e. feasible solutions). Our program can obtain better floorplan solutions than Parquet-2.1 for all test cases in shorter running times; our program, on the average, can reduce wirelength by about 20% and runtime by about 55%, compared to Parquet-2.1.

The above results all show the efficiency and effectiveness of Fast-SA. Our method results in very stable and high-quality floorplan solutions. Figure 15 shows the resulting floorplans for n100 with various aspect ratios.

6.3 Bus-Driven Floorplanning

We also performed experiments on bus-driven floorplanning. The benchmarks are provided by the authors of [22]; 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 2.8GHz Intel Pentium 4 PC while the work [22] is on a 2.4GHz Xeon PC; the speed difference between the two machines is marginal. The work [22] 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 [6]. 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:

- $R_b = e.x_2 - b.x_1$, where $e.x_2 = \min\{g.x_2 | g.x_2 > b.x_2, g \in M\}$;
- $L_b = d.x_2 - b.x_1$, where $d.x_2 = \max\{g.x_2 | g.x_2 < b.x_2, g \in M\}$;
- $T_b = a.y_2 - b.y_1$, where $a.y_2 = \min\{g.y_2 | g.y_2 > b.y_2, g \in M\}$;
- $B_b = c.y_2 - b.y_1$, 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 simulated annealing:

- Change the width of block b_i to R_i;
- Change the width of block b_i to L_i;
- Change the height of block b_i to T_i;
- Change the height of block b_i to B_i;
- Change the aspect ratio of block b_i to a random value in the range of the given soft aspect ratio constraint.

Figure 16 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 the right boundary of b_3 to R_3, it can generate a more compact floorplan.

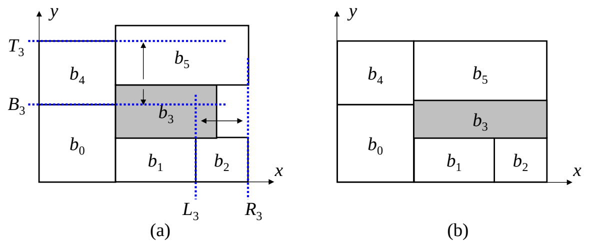


Figure 16: A soft block resizing example. (a) There are four shape candidates for b_3 for the resizing. (b) Stretch the right boundary of b_3 to R_3, and the resulting floorplan becomes more compact.

The block shapes could be changed to obtain a more compact floorplan during simulated annealing. For hard blocks, our average dead space is 4.38% while the work [22] is 5.51%. We only needed 26 seconds on average while [22] required 104 seconds. For soft blocks, since the previous work [22] resizes blocks from existing solutions, 2 seconds are added to the average runtime (106 seconds in total). Our method performs resizing and floorplanning at the same time; so the runtimes are longer than hard block floorplanning alone (but is still much faster than the work [22]). The average runtime is 47 seconds, 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. Figure 17 shows the resulting floorplan for ami49-3.

7. 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 stabler convergence to desired floorplan solutions. The experimental results on the fixed-outline floorplanning

Table 6: Bus-driven floorplanning results. The results of SP is on a 2.4GHz Intel Xeon PC while ours is on a 2.8GHz Intel Pentium 4 PC.

Block type	Circuit	Block	Bus	SP [22]		Ours	
				Time (sec)	Dead space	Time (sec)	Dead space
Hard	apte	9	5	11	4.11%	2	1.59%
	xerox	10	6	12	3.88%	5	3.85%
	hp	11	14	28	5.02%	20	4.47%
	ami33-1	33	8	61	6.02%	19	5.69%
	ami33-2	33	18	81	6.10%	22	3.87%
	ami49-1	49	9	98	5.42%	28	5.34%
	ami49-2	49	12	278	6.09%	43	5.45%
	ami49-3	49	15	265	7.40%	66	4.74%
	Average			104	5.51%	26	4.38%
Soft	apte	9	5	12	0.72%	3	0.02%
	xerox	10	6	13	0.95%	6	0.10%
	hp	11	14	28	0.62%	11	0.03%
	ami33-1	33	8	62	0.94%	30	0.33%
	ami33-2	33	18	86	1.27%	73	0.73%
	ami49-1	49	9	101	0.85%	58	0.51%
	ami49-2	49	12	281	0.84%	112	0.67%
	ami49-3	49	15	268	1.09%	81	0.92%
	Average			106	0.91%	47	0.41%

and the bus-driven floorplanning both have shown the efficiency and effectiveness of our floorplanning algorithms; for those applications, our results outperform the related recent works by large margins.

8. ACKNOWLEDGEMENTS

We would like to thank Professor Igor Markov and the authors of [4], Dr. Hua Xiang and the authors of [22], and Professor De-Sheng Chen and the authors of [13] for providing us with their benchmarks and/or programs for the comparative studies.

9. REFERENCES

[1] E.H.L. Aarts and P.J.M. van Laarhoven, "A new polynomial time cooling schedule," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1985.

[2] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," *Proceedings of IEEE International Conference on Computer Design*, pp. 328-334, 2001.

[3] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-Blocks using Floorplanning and Standard-Cell Placement," *Proceedings of ACM International Symposium on Physical Design*, pp. 12-17, San Diego, 2002.

[4] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. on VLSI Systems*, vol 11(6), pp. 1120-1135, December 2003.

[5] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: a new presentation for non-slicing floorplans," *Proceedings of ACM/IEEE Design Automation Conference*, pp. 458-463, 2000.

[6] J.-C. Chi and M. C. Chi, "A block placement algorithm for VLSI circuits," *Chung Yuan Journal*, Vol. 31, No. 1, March 2003, pp. 69-75.

[7] GSRC Floorplan Benchmark Suite, http://www.cse.ucsc.edu/research/surf/GSRC/GSRCbench.html

[8] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, pp. 510-522, Volume 20, 1985.

[9] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplans and its applications," *Proceedings of ACM/IEEE Design Automation Conference*, pp. 268-273, 1999.

[10] A. B. Kahng, "Classical floorplanning harmful?" *Proceedings of ACM International Symposium on Physical Design*, pp.

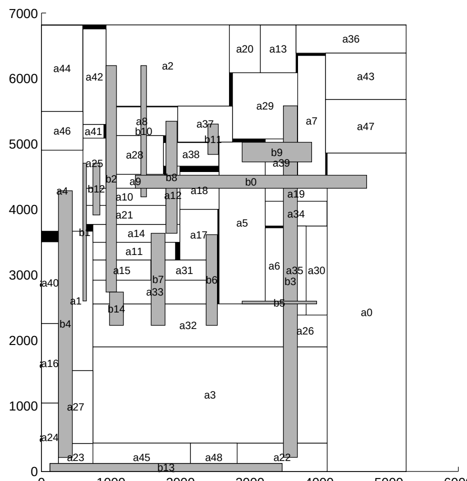


Figure 17: The resulting packing of ami49-3 with soft block adjustment. There are 49 blocks and 15 buses. The buses are {0, 5, 9, 12, 18}, {1, 10, 21, 25}, {2 ,28 ,33}, {3, 19, 22, 26, 29, 34}, {4, 23 ,27}, {5, 35, 30, 6}, {32, 31, 17}, {11, 14, 15, 32, 33}, {12, 8, 14}, {5, 7, 39}, {2, 8, 9, 10}, {37,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

207-213, 2000.

[11] S. Kirpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, pp.671-680, 1983.

[12] H.-C. Lee, Y.-W. Chang, J.-M. Hsu, and H. H. Yang, "Multilevel floorplanning/placement for large-scale modules using B*-trees," *Proceedings of ACM/IEEE Design Automation Conference*, pp. 812-817, 2003.

[13] C.-T. Lin, D.-S. Chen, and Y.-W. Wang, "Robust fixed-outline floorplanning through evolutionary search," *Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 42-44, 2004.

[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 472-479, 1995.

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

[16] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Bus-based integrated floorplanning," *Proceedings of IEEE International Synposium on Circuits and Systems*, pp. 875-878, 2002.

[17] F. Rafiq, M. Chrzanowska-Jeske, H. H. Yang, and N. Sherwani, "Integrated floorplanning with buffer/channel insertion for bus-based microprocessor designs," *Proceedings of ACM International Symposium on Physical Design*, pp. 875-878, 2002.

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

[19] X. Tang and D. F. Wong, "Floorplanning with alignment and performance constraints," *Proceedings of ACM/IEEE Design Automation Conference*, pp. 848-853, 2002.

[20] D.F. Wong, H.W. Leong, and C.L. Liu, *Simulated Annealing for VLSI Design*, Kluwer academic pulishers, Boston, 1988.

[21] M.-C. Wu and Y.-W. Chang, "Placement with alignment and performance constraints using the B*-tree representation," *Proceedings of IEEE International Conference on Computer Design*, 2004.

[22] H. Xiang, X. Tang, and M. D.F. Wong, " Bus-Driven Floorplaning," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 66-73, 2003.