



ELSEVIER

Available online at www.sciencedirect.com

INTEGRATION, the VLSI journal 41 (2008) 306–316

INTEGRATION
 theVLSI journal

www.elsevier.com/locate/vlsi

Multi-bend bus driven floorplanning[☆]

Jill H.Y. Law, Evangeline F.Y. Young*

Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

Received 10 July 2006; received in revised form 10 September 2007; accepted 10 September 2007

Abstract

In this paper, the problem of bus-driven floorplanning is addressed. Given a set of blocks and bus specifications (the width of each bus and the blocks that the bus need to go through), we will generate a floorplan solution such that all the buses go through their blocks, with the area of the floorplan and the total area of the buses minimized. The approach proposed is based on a simulated annealing framework. Using the sequence pair representation, we derived and proved some necessary conditions for feasible buses, for which we allow 0-bend, one-bend, or two-bend. A checking will be performed to identify those buses that cannot be placed simultaneously. Finally, a solution will be generated giving the coordinates of the modules and the buses. Comparing with the results of the most updated work on this problem by Xiang et al. [Bus-driven floorplanning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 2003, pp. 66–73], our algorithm can handle buses going through many blocks and the dead space of the floorplan obtained is also reduced. For example, if the buses have to go through more than 10 blocks, the approach in Xiang et al. [Bus-driven floorplanning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 2003, pp. 66–73] is not able to generate any solution while our algorithm can still give solutions of good quality.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Physical design; CAD; Floorplanning; Bus-driven; Simulated annealing

1. Introduction

Floorplanning is to plan the positions and shapes of a set of modules at the beginning of the design cycle to optimize circuit performance. Interconnect-driven floorplanning is considered to be one of the most important problems in physical design today. As the complexity of chip design increases, the amount of interconnections between different modules on a chip also increases rapidly. Bus is a collection of wires, which can be used to carry signals between different modules. Bus routing has become more and more important as the complexity of chip design increases. An area-compact floorplan is not necessarily bus-routable. In order to ease bus routing and avoid unnecessary iterations of the physical design cycle, it would be

favourable to incorporate this bus routing problem in the early designing phases. Bus-driven floorplanning considers bus placement. Buses are of different widths and need to go through different sets of modules. Therefore, the positions of the modules will affect the placement of the buses. The objective of the problem is to obtain a bus-routable floorplan such that the area of the chip and the total area of the buses are minimized.

The floorplanning problem in general is a well-studied problem. There are three kinds of floorplan: slicing, non-slicing and mosaic. Many new representations were introduced in the past decade to represent different kinds of floorplans, including sequence pair [1], bounded-sliceline grid [2], O-tree [3], B*-tree [4], transitive closure graph [5], corner block list (CBL) [6], Q-sequence [7], twin binary tree [8] and twin binary sequence [9], etc. Some of these floorplanners are extended to handle placement constraints in floorplan design. The floorplanners in [4,10,11] can handle pre-place constraint in which some modules are fixed in position. The paper [12–14] work on boundary constraint in which some modules are constrained to be

[☆]The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 4188/03E).

*Corresponding author.

E-mail address: fyoung@cse.cuhk.edu.hk (E.F.Y. Young).

placed along one of the four sides of the chip for I/O connection. The paper [15] generalizes the approach in [10] to handle range constraint in which some modules are restricted to be placed within some rectangular ranges. In [16], the authors tried to enforce the abutment constraint based on the CBL representation. It is shown that the abutment information of the blocks can be deduced from the CBL representation. However, the blocks on a bus are not necessarily abutted. Thus, their approach cannot be used to solve the bus driven-floorplanning problem. In [17], the authors proposed a unified method to handle simultaneously different kinds of placement constraints, including pre-placed constraint, range constraint, boundary constraint, alignment, abutment and clustering constraint, etc. All these constraints were modeled as a collection of “relative placement constraints” and “absolute placement constraints”, and were enforced by inserting edges in the constraint graphs. However, this approach is not suitable for bus-driven floorplanning as for a bus, the order in which the blocks are placed on a bus is not fixed. Besides, we do not know beforehand the shape of a bus which can be 0-bend, 1-bend, or 2-bend in our problem.

Based on the sequence pair representation, the authors of [18] proposed a method to enforce alignment constraint and some other performance constraints in floorplanning. Although the alignment constraint mentioned in [18] is not applicable for bus-driven floorplanning, their intuition on deducing the approximate positions of the blocks by looking at the sequence pair is very helpful. In [19], the authors have made use of the idea from [18] to design an intact algorithm to solve the bus-driven floorplanning problem. In [19], the authors aimed at solving the bus-driven floorplanning problem, based on a simulated annealing (SA) framework. Each candidate floorplanning solution were checked in an evaluation step to see if the buses are feasible, i.e., the required set of blocks can be passed through by a 0-bend bus. Sequence pair representation was used. One major drawback of this approach is that, only horizontal and vertical buses are considered and the solution quality will deteriorate if the number of blocks involved in each bus is large. Our proposed algorithm has made a significant improvement over [19] by allowing 0-bend, 1-bend, and 2-bend buses.

In this paper, this bus-driven floorplanning problem will be re-visited. Unlike [19], our proposed algorithm allows 0-bend, 1-bend (or 1-via), and 2-bend (or 2-via) buses. Experimental results have shown that our algorithm can generate solutions with high quality especially when the number of blocks in each bus is large. For example, if the buses have to go through more than 10 blocks each, the approach in [19] is not able to generate any solution while our algorithm can still give solutions of good quality.

The rest of this paper is organized as follows. A formal definition of the problem will be given in Section 2. After that, an algorithm is proposed to solve the problem, and details will be discussed in Section 3. Experimental results

will be presented in Section 4. Finally, a conclusion will be drawn in Section 5.

2. Problem formulation

We assume that buses are routed on two layers, one for horizontal buses and the other for vertical buses. The bus-driven floorplanning problem can be formulated as follows.

Given the following:

- (1) A set of n blocks $B = \{b_0, b_1, \dots, b_{n-1}\}$, where each block b_i is associated with a width w_i and a height h_i , where $w_i, h_i \in \mathbf{R}^+$.
- (2) A set of m buses $U = \{u_0, u_1, \dots, u_{m-1}\}$, where each bus u_i has a width t_i where $t_i \in \mathbf{R}^+$, and need to go through a set of blocks $B_i, B_i \subseteq B$.

Our task is to decide the position of each block and the route of each bus such that every bus u_i goes through all its blocks, there is no overlapping between any two blocks and there is no overlapping between the horizontal (vertical) components of the buses (since there are only two layers for bus routing). The goal is to minimize the chip area and the total bus area. We consider buses of 0-bend, 1-bend, or 2-bend only in our problem to minimize the number of vias used.

We will define the meaning of “going through” in the following. For a horizontal component of a bus u_i to go through a set of blocks $\{A, B, C\}$, the vertical overlapping between the blocks has to be greater than or equal to the bus width t_i of u_i . An example is shown in Fig. 1. The condition for a vertical component of a bus to go through a set of blocks can be defined similarly.

3. Methodology

SA will be used as the searching engine. A candidate solution will be evaluated according to (1) the number of buses routed successfully, (2) the total area of the buses, and (3) the total area of the floorplan. There are three main steps to evaluate a solution. The first step is to determine the shapes of the buses by examining the sequence pair. After that, a bus ordering is found such that all feasible buses can be laid out successfully by following this order. Finally, a flooplan is obtained by calculating the coordinates of the blocks and the buses. Bus feasibility is

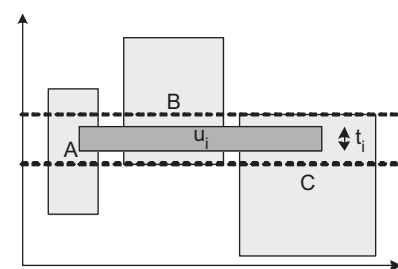


Fig. 1. Bus u_i goes through A , B , and C .

performed on the sequence pair representation of a candidate solution first because some candidate floorplans cannot be realized under the bus constraints, and it will be more efficient if we can check that out by just looking at the sequence pair representation before the floorplan realization step. Details of each step will be given in the following sections.

3.1. Shape validation

We can deduce the shape of a bus by looking at the sequence pair representation of the floorplan. As we allow buses of at most two bends, buses that cannot be realized in two bends will be considered as infeasible and will be excluded from further checking. A penalty will be added to the cost of the annealing process for each infeasible bus.

An example is shown in Fig. 2. Consider a sequence pair $(FGHICDEAB, ABCDEFGHI)$, a bus u_i that need to go through the blocks in $\{D, E, G\}$ can be realized as a 1-bend bus (Fig. 2(a)). Another bus u_j that need to go through the blocks in $\{A, C, D, E, G, H, I\}$ will have at least three bends (Fig. 2(b)) and it will be marked as infeasible. The aim of this step is to find out all the infeasible buses and to determine the shape of each feasible bus.

Given a bus u_i that need to go through $B_i = \{b_1, b_2, \dots, b_k\}$, we will first extract those blocks in B_i from the sequence pair, without altering their relative positions. For example, if we are checking a bus that need to go through a set of blocks $\{A, B, E\}$ in a floorplan represented by the sequence pair $(ADBCE, EBCAD)$, we will first extract $sp_i = (ABE, EBA)$ from the sequence pair and examine sp_i to decide whether u_i can be routed successfully as a 0-bend, 1-bend, or 2-bend bus one after another.

3.1.1. 0-Bend bus shape validation

A 0-bend bus is simply a horizontal or a vertical bus. For a bus u_i to be 0-bend, the orders of the blocks in the two sequences of $sp_i = (\alpha, \beta)$ have to be either the same, i.e., $\alpha = \beta$ (for horizontal bus) or reversed, i.e., $\alpha = \beta^R$ (for vertical bus). For example, given a sequence pair $(DEFABC, ABCDEF)$ and a bus u_0 that need to go through the blocks in $\{A, B, C\}$, the first step is to extract the corresponding blocks from the sequence pair: $sp_0 = (ABC, ABC)$. As the blocks appear in the same order in both sequences, it can be concluded that u_0 can be realized as a horizontal bus. For another bus u_1 that has to go through the blocks in $\{C, F\}$, the extracted sp_1 is

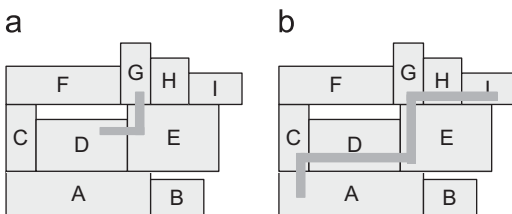


Fig. 2. (a) A 1-bend bus. (b) A 3-bend bus.

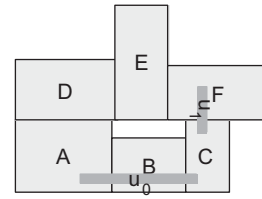


Fig. 3. Two valid 0-bend buses, $\{A, B, C\}$ and $\{C, F\}$.

(FC, CF) . As the blocks appear in reversed order, it can be realized as a vertical bus. This example is illustrated in Fig. 3. The 0-bend condition is stated formally as follows:

Condition 0-bend: A pair of sequences $s_i = (\alpha_i, \beta_i)$ satisfies the 0-bend condition if and only if $\alpha_i = \beta_i$ or $\alpha_i = \beta_i^R$.

3.1.2. 1-Bend bus shape validation

A 1-bend bus is also called an L-shaped bus. For a bus to be 1-bend, a necessary condition is that it consists of one vertical component and one horizontal component. This can be checked easily by identifying the longest common subsequence (LCS) in sp_i first, and then check if the remaining blocks (after removing the blocks in the LCS) in the two sequences are in reversed order. The 1-bend condition is stated formally as follows:

Condition 1-bend: A pair of sequences $s_i = (\alpha_i, \beta_i)$ satisfies the 1-bend condition if and only if $\alpha_i - \gamma = (\beta_i - \gamma)^R$ where γ is the LCS in s_i .

Lemma 1. The 1-bend condition is necessary for a sequence pair s_i that allows a bus passing through all its blocks with only one bend.

Proof. To pass through a set of blocks with a 1-bend bus, some of the blocks must be horizontally related while the remaining is vertically related. Those horizontally related will form one common subsequence in s_i . To check the feasibility, we only need to look at the LCS, instead of any common subsequence. This is because if taking an LCS as the horizontal component fails to form a valid L-shape, taking any other LCS or shorter subsequences will also fail. Let l_1 be the LCS of sp_i we picked and l_2 be another common subsequence such that $|l_2| \leq |l_1|$. There are two different cases according to the number of blocks in l_1 but not in l_2 . The first case is that there are at least two blocks n_1 and n_2 in l_1 but not in l_2 . Then, a valid L-shape cannot be formed with l_2 as the horizontal component because n_1 and n_2 , which are not in l_2 , must also be in a left-right relationship with each other. This implies two separate horizontal components and a valid L-shape cannot be formed.

The second case is that there is only one block n_1 in l_1 but not in l_2 . There are two sub-cases according to the lengths of l_1 and l_2 . In the first sub-case, $|l_2| < |l_1|$, i.e., $|l_2| = |l_1| - 1$, the block n_1 can serve as the joint between the vertical and the horizontal components if a valid L-shape can be formed, and taking either l_1 or l_2 as the horizontal component is the same. In the second sub-case,

$|l_2| = |l_1|$, then there will be another block n_2 in l_2 but not in l_1 . Notice that n_1 and n_2 must be in an upper–lower relationship with each other. If n_2 is not the leftmost nor the rightmost block in l_2 , a valid L-shape cannot be formed by taking l_2 as the horizontal component since the vertical component will appear in the middle part of the horizontal component otherwise. If n_2 is the leftmost or the rightmost block in l_2 (and so as n_1 in l_1), taking either l_1 or l_2 as the horizontal component is the same, as the block n_1 or n_2 can serve as the joint between the vertical and the horizontal components if a valid L-shape can be formed. \square

In the following steps, we will regard the first and the last block of the LCS as in the vertical component and will keep them for checking whether the vertical component is on the left or on the right of the horizontal component. Let us look at an example. Given a sequence pair $(DEFABC, ABCDEF)$ and a bus u_3 that has to go through the blocks $\{A, B, C, D\}$, the first step is to extract the corresponding blocks $sp_3 = (DABC, ABCD)$ from the sequence pair. As it failed the 0-bend checking, the next step is to check if it can be realized as a 1-bend bus. The LCS of sp_3 is ABC , so ABC will be taken as the horizontal component of u_3 and B will be removed from sp_3 . Then we have to check whether the remaining block D can form a vertical component with the block A or C . As the blocks A and D appear in reversed order in sp_3 , AD can form the vertical component of u_3 (Note that C and D also appear in reversed order in sp_3 and we can pick either AC or CD). After checking, u_3 is classified as a valid 1-bend bus. This example is illustrated in Fig. 4. Let us look at another example. Given the same sequence pair $(DEFABC, ABCDEF)$ and another bus u_4 that has to go through the blocks in $\{A, B, E, F\}$, we first extract the corresponding blocks $sp_4 = (EFAB, ABEF)$ from the sequence pair. The LCS is AB or EF . As there exist more than one LCS, we can take any one of them. Suppose that we take AB , the remaining subsequences are not in reversed order, so it is not a valid 1-bend bus and it

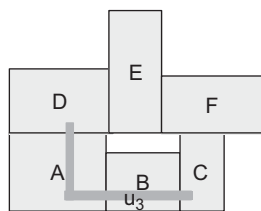


Fig. 4. A valid 1-bend bus $\{A, B, C, D\}$.

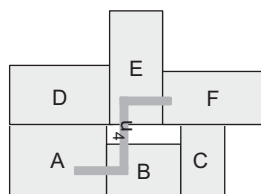


Fig. 5. Bus u_4 cannot be realized as a 1-bend bus.

will proceed to the 2-bend checking. This example is illustrated in Fig. 5.

3.1.3. 2-Bend shape validation

If a bus is found to be neither 0-bend nor 1-bend, we will check whether it is a 2-bend bus. There are several kinds of 2-bend buses, Z-shaped, mirrored Z-shaped, C-shaped, or mirrored C-shaped. There will be two horizontal (vertical) components and one vertical (horizontal) component in the bus, denoted by HVH or VHV, respectively. Assuming the case of HVH in the following discussion, we will first identify the vertical component of the bus. Let the extracted sequence pair sp_i of bus u_i be (α, β) , where α and β are strings of blocks. The vertical component can be found by finding the LCS in (α, β^R) , where β^R denotes the reverse of the string β . Similar to 1-bend checking, the first block and the last block of the LCS will be kept for horizontal component checking. Besides, we have to pick an LCS but not any other shorter subsequence, and if there are more than one LCSs, picking any one of them will be the same. The argument is similar to that in 1-bend shape validation.

After identifying the vertical component, we will classify the remaining blocks of the bus into different relationships with the vertical component. For example, consider a bus u_i with extracted sequence pair $sp_i = (\alpha, \beta) = (ABCFDE, AEDCBF)$. The LCS of α and β^R is $BCDE$. The block A will be classified as in the set *Left*, as A is on the left of all the blocks in the vertical component. On the other hand, block F will be classified as in the set *UpperRight*, as F is on the right-hand side of the blocks B and C and on top of the blocks D and E . We can deduce these relationships easily from the sequence pair. There are totally eight *position sets*:

- (1) A block is in the set *Upper* if it is above all the blocks in the LCS.
- (2) A block is in the set *UpperLeft* if it is above or on the left of all the blocks in the LCS.
- (3) A block is in the set *Left* if it is on the left of all the blocks in the LCS.
- (4) A block is in the set *LowerLeft* if it is below or on the left of the blocks in the LCS.
- (5) A block is in the set *Lower* if it is below the blocks in the LCS.
- (6) A block is in the set *LowerRight* if it is below or on the right of the blocks in the LCS.
- (7) A block is in the set *Right* if it is on the right of the blocks in the LCS.
- (8) A block is in the set *UpperRight* if it is above or on the right of the blocks in the LCS.

There are four valid shapes for the case of HVH: Z-shape, mirrored Z-shape, C-shape, and mirrored C-shape. In order to form a valid shape, some of the position sets have to be emptied. For example, to form a mirrored Z-shape, there should be no blocks in the upper-left and lower-right directions of the vertical component. Thus, the sets

UpperLeft and *LowerRight* have to be emptied. The blocks in the set *Upper*, *UpperRight*, and *Right* will form one horizontal component, and the blocks in the set *Lower*, *LowerLeft*, and *Left* will form another horizontal component. Details are shown in Fig. 6. The last step is to check both horizontal components to ensure that the blocks in each component can indeed align horizontally, i.e., the blocks appear in the same order in both sequences of sp_i . The 2-bend condition is stated formally as follows:

Condition 2-bend: A pair of sequences $s_i = (\alpha_i, \beta_i)$ satisfies the 2-bend condition if (1) the blocks not in the LCS of α_i and β_i are clustered in one quadrant on the left side of the LCS and one quadrant on the right side of the LCS or (2) the blocks not in the LCS of α_i and β_i^R are clustered in one quadrant on the upper side of the LCS and one quadrant on the lower side of the LCS.

Lemma 2. *The 2-bend condition is necessary for a sequence pair s_i that allows a bus passing through all its blocks with only two bends.*

Proof. To pass through a set of blocks with a 2-bend bus that has a horizontal (vertical) trunk, some of the blocks must be horizontally (vertically) related to form the trunk while the remaining will form two vertical (horizontal) components, one on the left (upper) side and one on the right (lower). Those horizontally (vertically) related blocks will form one common subsequence in α_i and β_i (α_i and β_i^R). To check feasibility, we only need to look at the LCS,

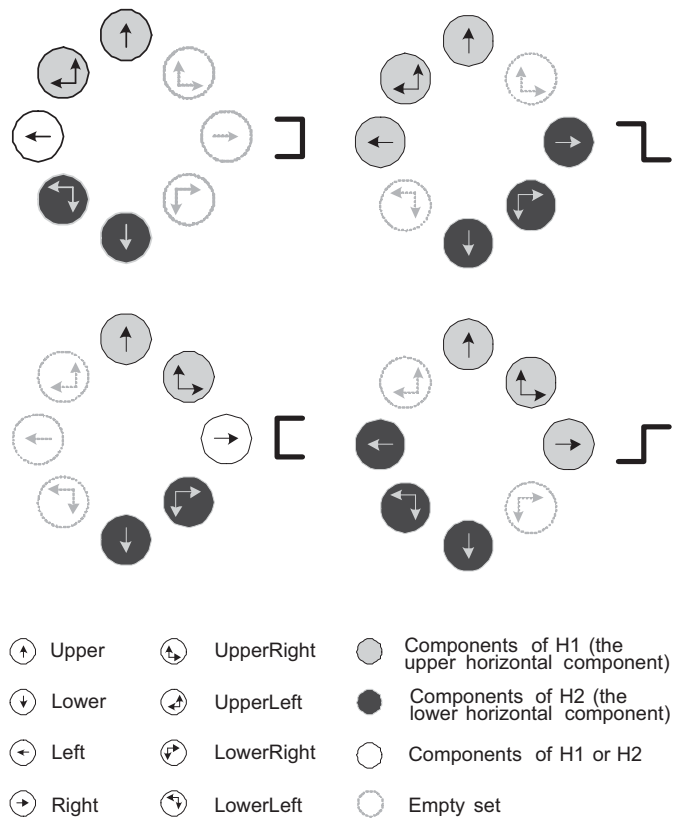


Fig. 6. Necessary conditions on the position sets to form valid 2-bend shape.

instead of any common subsequence and the argument is similar to that for Lemma 1. \square

The shape validation step for 0-bend, 1-bend, and 2-bend buses can be incorporated into one whole process and the overall algorithm is shown in Figs. 7–9.

3.2. Bus ordering

In this step, we aim at determining an ordering between the valid buses, and removing those that have conflicts with others. For example, given a sequence pair $(CADB, ACBD)$, block *C* has to be placed above block *A* according to the orderings in the sequence pair, so any horizontal bus going through block *C* has to be placed above any horizontal bus going through block *A*. This kind of constraint is called the bus ordering constraint.

However, some ordering constraints may be contradictory to each other. An example is shown in Fig. 10. In this example, block *A* is on the left of block *B* according to the sequence pair, so any vertical bus going through *A* has to be placed on the left of any vertical bus going through block *B*. Similarly, block *C* is on the left of block *D* and thus, any vertical bus going through *C* has to be placed on the left of any vertical bus going through *D*. Problem will occur if there are two 2-bend buses u_i and u_j , where a vertical component of u_i has to go through blocks *A* and *D*, and a vertical component of u_j has to go through blocks *B* and *C*. These two vertical components have to be placed on the left-hand side of each other, which is impossible. This step aims at removing the least number of buses such that the remaining buses do not have any conflict with each other. For simplicity, our discussion below is limited to the horizontal components of the buses, where the case for the vertical components can be derived similarly.

Assuming that buses are routed on two layers, one layer for horizontal buses and the other for vertical buses. We can consider the constraints between horizontal components and the constraints between vertical components separately. For 1-bend or 2-bend buses, we will first break them down into two or three 0-bend components, respectively, before checking the ordering constraints (Fig. 11). For horizontal components, we will use a graph $G_h = (V, E)$ to determine whether all the ordering constraints can be satisfied. Each vertex in V represents a 0-bend horizontal component, and $E = \{(v_i, v_j) | \text{component } v_i \text{ has to be placed above component } v_j\}$. In order to check if $(v_a, v_b) \in E$, we will first extract sp_{ab} from the sequence pair, where sp_{ab} contains only the blocks in u_a and u_b . For example, if the sequence pair is $(ABCDEF, DEACBF)$, and u_a has to go through block *A* and *B* and u_b has to go through block *C* and *D*, the extracted sp_{ab} will be $(ABCD, DACB)$. Let m be a block, $s_1[m]$ denotes the position of block m in the first sequence of sp_{ab} , e.g., $s_1[A]$ in the above example is one. Similarly, $s_2[m]$ is the position of block m in the second sequence of sp_{ab} . In the above example, $s_2[A]$ is

```

SHAPE_VALIDATION (int  $i$ )
1    $k \leftarrow$  number of blocks that bus  $u_i$  has to go through
2   Extract  $sp_i$  from the sequence pair
3   Find the longest common subsequence  $lcs_i$  of  $sp_i$ 
4   IF  $|lcs_i| = 1$  OR  $|lcs_i| = k$ 
5       Mark as 0-bend
6        $result \leftarrow$  SUCCESS
7   ELSE
8       Put the remaining blocks into position sets
9        $result \leftarrow$  ONE_BEND_CHECK( $i$ )
10      IF  $result =$  FAIL
11           $result \leftarrow$  TWO_BEND_CHECK_VHV( $i$ )
12          IF  $result =$  FAIL
13              Reverse the first sequence in  $sp_i$ 
14              Find the longest common subsequence of  $sp_i$ 
15              Put the remaining blocks into position sets
16               $result \leftarrow$  TWO_BEND_CHECK_HVH( $i$ )
17          END IF
18      END IF
19  END IF
20  RETURN  $result$ 

```

Fig. 7. Pseudo-code for shape validation.

```

ONE_BEND_CHECK (int  $i$ )
1    $result \leftarrow$  FAIL
2   IF Right =  $\emptyset$  (The vertical component must be on the left)
3       IF UpperRight =  $\emptyset \wedge$  LowerRight =  $\emptyset \wedge$  Lower =  $\emptyset \wedge$  LowerLeft =  $\emptyset$ 
4           IF Upper  $\cup$  UpperLeft can form a vertical component
5               Mark as  $\perp$ -shape and  $result \leftarrow$  SUCCESS
6           END IF
7       ELSE IF UpperLeft =  $\emptyset \wedge$  Upper =  $\emptyset \wedge$  UpperRight =  $\emptyset \wedge$  LowerRight =  $\emptyset$ 
8           IF Lower  $\cup$  LowerLeft can form a vertical component
9               Mark as  $\lrcorner$ -shape and  $result \leftarrow$  SUCCESS
10          END IF
11      END IF
12  ELSE IF Left =  $\emptyset$  (The vertical component must be on the right)
13      IF UpperLeft =  $\emptyset \wedge$  LowerLeft =  $\emptyset \wedge$  Lower =  $\emptyset \wedge$  LowerRight =  $\emptyset$ 
14          IF Upper  $\cup$  UpperRight can form a vertical component
15              Mark as  $\lrcorner$ -shape and  $result \leftarrow$  SUCCESS
16          END IF
17      ELSE IF UpperRight =  $\emptyset \wedge$  Upper =  $\emptyset \wedge$  UpperLeft =  $\emptyset \wedge$  LowerLeft =  $\emptyset$ 
18          IF Lower  $\cup$  LowerRight can form a vertical component
19              Mark as  $\lrcorner$ -shape and  $result \leftarrow$  SUCCESS
20          END IF
21      END IF
22  END IF
23  RETURN  $result$ 

```

Fig. 8. Pseudo-code for 1-bend checking.

two. After computing the $s_1[m]$ and $s_2[m]$ for each related block m , we will check if sp_{ab} falls into one of the following three cases according to [19] (Fig. 12):

- (1) If $\forall x \in B_a, s_1[x] \geq s_2[x]$, and $\exists y \in B_a, s_1[y] > s_2[y]$, then u_a is below u_b . Thus, $(v_b, v_a) \in E$.
- (2) If $\forall x \in B_b, s_1[x] \geq s_2[x]$, and $\exists y \in B_b, s_1[y] > s_2[y]$, then u_b is below u_a . Thus, $(v_a, v_b) \in E$.

- (3) If $\exists x \in B_a, s_1[x] > s_2[x]$, and $\exists y \in B_b, s_1[y] > s_2[y]$, then contradiction occurs, as u_a cannot be above u_b and below u_b at the same time. Thus, $(v_b, v_a) \in E$ and $(v_a, v_b) \in E$.

In the floorplanner, we simply perform the checking for every pair of horizontal (vertical) bus components directly. As some of the buses cannot be placed simultaneously, our

```

TWO_BEND_CHECK_VHV (int i)
1  result ← FAIL
2  IF UpperRight = ∅ AND LowerLeft = ∅
3      IF the blocks in Upper, UpperLeft, Left can be vertical AND
4         the blocks in Lower, LowerRight, Right can be vertical
5         Mark as 2-bend and result ← SUCCESS
6      END IF
7  ELSE IF UpperLeft = ∅ AND LowerRight = ∅
8      IF the blocks in Upper, UpperRight, Right can be vertical AND
9         the blocks in Lower, LowerLeft, Left can be vertical
10        Mark it as 2-bend and result ← SUCCESS
11     END IF
12  ELSE IF LowerLeft = ∅ AND LowerRight = ∅ AND Lower = ∅
13     IF the blocks in Upper, UpperLeft, Left can be vertical AND
14        the blocks in Upper, UpperRight, Right can be vertical
15        Mark it as 2-bend and result ← SUCCESS
16     END IF
17  ELSE IF UpperLeft = ∅ AND UpperRight = ∅ AND Upper = ∅
18     IF the blocks in Lower, LowerLeft, Left can be vertical AND
19        the blocks in Lower, LowerRight, Right can be vertical
20        Mark it as 2-bend and result ← SUCCESS
21     END IF
22  END IF
23  RETURN result
    
```

Fig. 9. Pseudo-code for 2-bend checking.

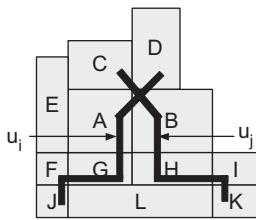


Fig. 10. Bus u_i has to be placed on the left of u_j and bus u_j has to be placed on the left of bus u_i .

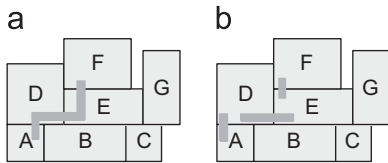


Fig. 11. A 2-bend bus is broken down into three 0-bend components for checking the ordering constraints.

aim in this step is to remove the least number of buses such that all the remaining buses can be placed. Besides, we aim at finding an ordering for the remaining buses such that they can be placed one after another successfully in a bottom-up (left–right) manner. To do so, we have to examine the graph G_h . Contradiction exists if cycle presences. Therefore the first step is to check whether cycles exist in G_h . If there are cycles, we want to remove the least number of nodes (bus components) to make the graph acyclic. However, this Node-Deleting Problem is proved to

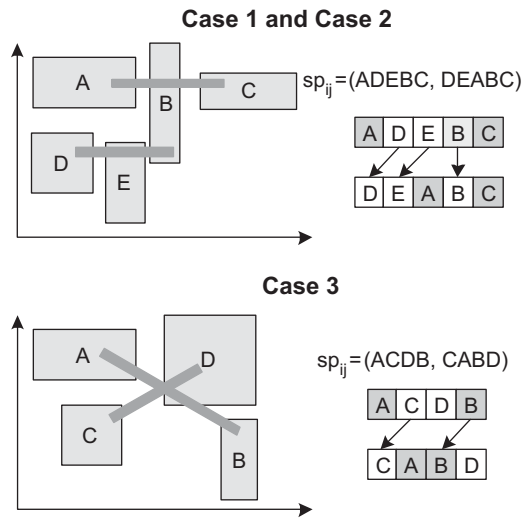


Fig. 12. Different cases of a bus ordering constraint.

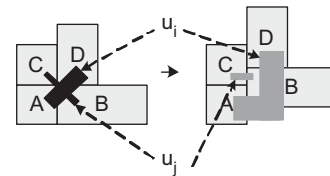


Fig. 13. Adding bend to resolve bus ordering conflict.

be NP-complete [19]. Our heuristic to solve the problem is to keep on removing the node with the highest degree (in-degree plus out-degree), until the graph is acyclic.

Assume that a 2-bend bus u_i is broken into three 0-bend components u_1 , u_2 , and u_3 , where u_1 and u_3 are horizontal and u_2 is vertical. When processing the horizontal components, a graph G_h is built. If u_1 is selected to be removed in order to make G_h acyclic, u_3 in the horizontal graph and u_2 in the vertical graph have to be removed as well. This is obvious since we should not keep partial bus components in the solution, if some other components of the bus are already marked as invalid. In some cases, bending can help to resolve conflicts in the ordering constraint graphs G_v and G_h . An example is shown in Fig. 13. In this example u_i and u_j are horizontal buses that contradict with each other. Changing u_i from 0-bend to 1-bend can resolve the conflict without removing any bus from the graph. However, this technique of adding bends to a bus to resolve conflict can only be used for buses that are 0-bend or 1-bend, so that one more bend can be added to resolve the conflict by the method as illustrated in Fig. 13. This can be done by processing those buses marked as invalid during the cycle removing step of G_h or G_v again to form buses with one more bend by calling the procedures ONE_BEND_CHECK(), TWO_BEND_CHECK_VHV() or TWO_BEND_CHECK_HVH().

After obtaining an acyclic graph, an ordering of the remaining valid components can be obtained from a topological sort of G_h .

3.3. Floorplan realization

The final step to evaluate a candidate solution is to realize the floorplan, i.e., obtaining the coordinates of the blocks and the buses, to determine the chip area and the total bus area. After the previous checkings, all the invalid buses are removed, and a correct bus ordering is found. Based on these information, we can compute the coordinates of all the blocks and the valid buses, and thus the chip area and total bus area. In order to obtain the coordinates of the blocks, we used the algorithm FAST-SP in [20] to construct a floorplan from the sequence pair. Then we use the same approach as in [19] to align blocks on the same bus, which can be described in brief as follows. The following process is repeated $O(m)$ times, where m is the total number of valid buses. Note that all 1-bend and 2-bend buses will have been broken down into 0-bend buses for processing. Let us consider horizontal buses only. In iteration i , bus u_i will be processed.

The coordinates of the blocks that u_i goes through will be computed first. Then, the position of u_i will be calculated by performing some basic alignment steps between the blocks that u_i goes through. These basic alignment steps for horizontal buses are shown in Fig. 14. An example is shown in Fig. 15.

After doing the basic alignment steps, we will check if u_i overlaps with any previously placed bus. If so, u_i will be moved up and the coordinate y_{u_i} will be updated. If u_i is moved up, the positions of all the blocks that u_i goes through may need to be computed again.

3.4. Time complexity

For a bus u_i passing through k blocks, the shape validation step will take $O(k \log k)$ time to find the LCS in sp_i . It then takes $O(k)$ time to put the remaining blocks into the position sets and to perform ONE_BEND_CHECK() and TWO_BEND_CHECK()s. To find the bus ordering, for each pair of horizontal (or vertical) bus components u_a and u_b , we need to scan the extracted sequence pair sp_{ab} once, so the total time will be $O(m^2 K)$ where m is the number of buses and K is the largest number of blocks a bus passes through. For the floorplan realization step, the packing step takes $O(n \log n)$ using a simpler version of the FAST-SP algorithm in [20], while the alignment step will take $O(K)$ for each bus component. Therefore, the total time taken will be $O(mK \log K + m^2 K + n \log n + mK) = O(mK \log K + m^2 K + n \log n)$.

3.5. Simulated annealing

SA is used to search for a good solution. In this section, the set of moves and the cost function used in the SA will be discussed.

BASIC_ALIGNMENT_H (i tni)

```

1    $y_{max} \leftarrow \max\{y_k : u_i \text{ goes through block } k\}$ 
2   FOR all blocks  $j$   $u_i$  goes through
3     IF  $y_{max} + t_i - h_j > y_j$  //  $t_i$  is the thickness of bus  $u_i$ 
4        $y_j \leftarrow y_{max} + t_i - h_j$  //  $h_j$  is the height of block  $j$ 
5     END IF
6   END FOR
```

Fig. 14. Pseudo-code of the basic alignment step for horizontal buses.

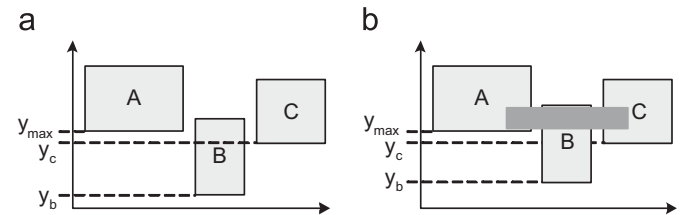


Fig. 15. (a) y_{max} , y_b , and y_c are calculated. (b) y_b has to be moved up to let the bus go through.

3.5.1. Moves

To change from one candidate solution to another, we use two operations, swap and rotate.

- (1) *Swap* is to exchange the positions of two blocks in either the first sequence or the second sequence. This can be done in constant time.
- (2) *Rotate* is to exchange a block’s height with its width. This can be done in constant time.

3.5.2. Cost function

As mentioned above, our objectives are to (1) accommodate all the buses, (2) minimize the total area of the buses, and (3) minimize the area of the floorplan. Thus, the

cost function is defined as follows.

$$Cost = \alpha \cdot A + \beta \cdot B + \gamma \cdot I,$$

where A is the chip area, B is the total bus area, I is the number of invalid bus, and α , β , and γ are parameters that can be specified by the users. In this bus-driven floorplanning problem, we focused on fitting all the buses in a compact floorplan solution. Other aspects like the total wire length and routing congestion can also be considered by including more terms in the cost function.

3.6. Handling soft blocks

In order to compare with the results presented in [19], we have added the feature of “soft block adjustment”. The adjustment is the same as that in [19]. This step makes use of the fact that the width and height of a block can be altered as long as the area is unchanged and the dimension is constrained by an aspect ratio bound. The process is again done by SA. The cost function is the same as before. In each pass, a block lying on a critical path will be selected, and the width or height of it will be changed a little bit. Then, the floorplan realization step is repeated to obtain a new chip area and total bus area. Note that if a valid bus is made invalid by this soft block adjustment step, the new candidate solution will be discarded. Besides, when changing a block width or height, the aspect ratio constraint cannot be violated.

4. Experimental results

The proposed algorithm is implemented using the C++ language and the experiments are conducted using an Intel Xeon (2.2 GHz) machine with 1 G memory. The test cases are derived from the MCNC benchmarks for floorplanning. In order to compare with the results presented in [19], the same test cases (Table 1) are tried using our proposed algorithm and all the experiments (including those of [19])

Table 1
Data set one

File	No. of blocks	No. of buses	Average/max. no. of blocks in a bus
apte	9	5	2.60/3
xerox	10	6	2.50/3
hp	11	14	2.29/3
ami33-1	33	8	4.17/6
ami33-2	33	18	2.39/4
ami49-1	49	9	4.00/6
ami49-2	49	12	3.58/6
ami49-3	49	15	3.53/6

Table 2
Data set two

File	No. of blocks	No. of buses	Average/max. no. of blocks in a bus
ami33-3	33	1	10.00/10
ami33-4	33	3	10.00/10
ami33-5	33	5	10.00/10
ami49-4	49	1	15.00/15
ami49-5	49	3	11.67/15
ami49-6	49	4	11.25/15

Table 3
Results of data set one

	Ref. [19]		Our work		Comparison ^a	
	Time (s)	Dead space (%)	Time (s)	Dead space (%)	Time (%)	Dead space difference (%)
apte	15	0.72	30	0.48	+100	-0.24
xerox	15	0.95	35	0.42	+133.33	-0.53
hp	33	0.62	51	0.29	+54.55	-0.33
ami33-1	11	0.94	93	1.00	+745.45	+0.06
ami33-2	92	1.27	144	1.19	+56.62	-0.08
ami49-1	16	0.85	71	0.56	+343.75	-0.29
ami49-2	302	0.84	713	0.58	+136.09	-0.26
ami49-3	285	1.09	865	0.60	+203.51	-0.49
				Average	+221.65	-31.54

^aCalculated by $((y_1 - y_0)/y_0) * 100\%$, where y_0 and y_1 are the time (or dead space) obtained by [19] and by our algorithm, respectively.

Table 4
Results of data set two

	Ref. [19]		Our work		Comparison	
	Time (s)	Dead space	Time (s)	Dead space (%)	Time	Dead space difference
ami33-3	86	1.81%	32	1.01	−62.79%	−0.80%
ami33-4	> 10 ³	–	92	1.90	–	–
ami33-5	> 10 ³	–	95	3.80	–	–
ami49-4	73	19.34%	88	0.63	+20.55%	−18.71%
ami49-5	> 10 ³	–	261	1.17	–	–
ami49-6	> 10 ³	–	140	2.19	–	–
		Average	118	1.78		

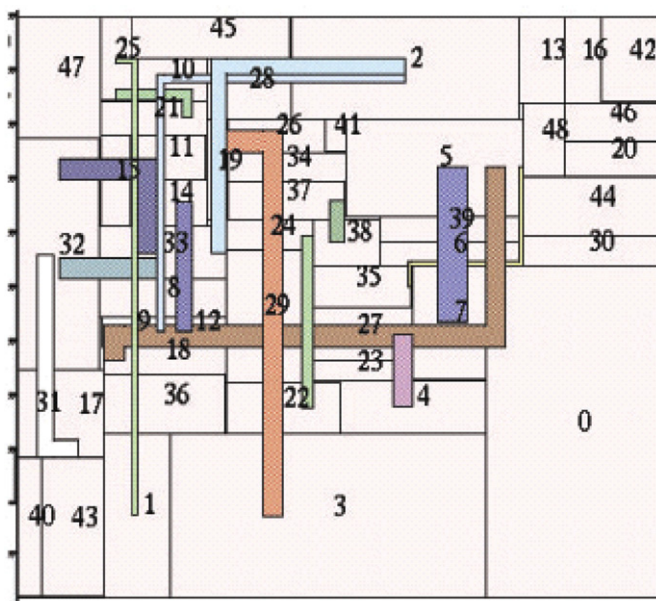


Fig. 16. Result packing of ami49-3.

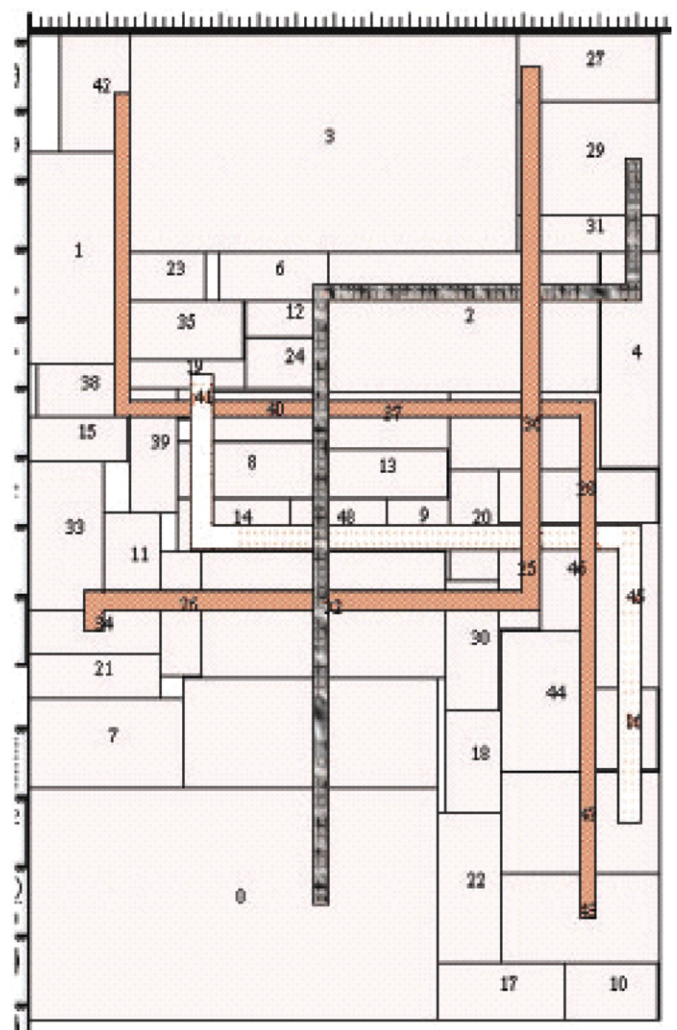


Fig. 17. Result packing of ami49-6.

are run on the same machine. The annealing processes are implemented in such a way that the stopping criteria of both programs are the same. The results are listed in Table 3. Comparing with the results of [19], the dead space of the floorplan obtained by our algorithm is reduced on average. The runtimes of our approach have increased because of the extra steps in processing 2-bend buses. The shape validation step has become more complicated with 2-bend buses in comparison with that in [19] which considers only 0-bend and 1-bend buses.

To demonstrate the importance of having 1-bend and 2-bend buses, we have created another set of test cases (Table 2) based on the ami33 and ami49 benchmarks. In these test cases, each bus will go through at least 10 blocks. The annealing process stops when the deadspace percentage is less than a certain threshold. The results are shown in Table 4. For this data set, the approach in [19] is not able to generate any solution for most of the test cases, while our algorithm can still generate solution of high

quality (with average dead space of 1.8% only). We can see that our algorithm can obtain much better performance. As their approach allows only 0-bend bus, it is very difficult to accommodate several buses that go through many blocks. Some resultant packings are shown in Figs. 16 and 17.

5. Conclusion and future work

In this paper, an algorithm to solve the bus-driven floorplanning problem allowing 0-bend, 1-bend, and 2-bend buses is proposed. Experimental results show that our approach is effective. The presence of 1-bend and 2-bend buses is important especially when the number of blocks that a bus goes through is large. It is difficult to find a solution if only 0-bend bus is allowed in those cases. One feasible extension of this work is to consider buses of other shapes with small number of vias.

Acknowledgement

We would like to thank the authors of [19] who have kindly given us their program and test cases such that we can conduct the experiments.

References

- [1] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, Rectangle-packing-based module placement, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1995, pp. 472–479.
- [2] S. Nakatake, K. Fujiyoshi, H. Murata, Y. Kajitani, Module placement on BSG-structure and IC layout applications, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1996, pp. 484–491.
- [3] P.-N. Guo, C.-K. Cheng, T. Yoshimura, An O-tree representation of non-slicing floorplan and its applications, in: Proceedings of the 36th ACM/IEEE Design Automation Conference, 1999, pp. 268–273.
- [4] Y.C. Chang, Y.W. Chang, G.M. Wu, S.W. Wu, B*-trees: a new representation for non-slicing floorplans, in: Proceedings of the 37th ACM/IEEE Design Automation Conference, 2000.
- [5] J. Lin, Y. Chang, TCG: a transitive closure graph-based representation for non-slicing floorplans, in: IEEE/ACM Proceedings of the Design Automation Conference, 2001, pp. 764–769.
- [6] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, J. Gu, Corner block list: an effective and efficient topological representation of non-slicing floorplan, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2000, pp. 8–12.
- [7] K. Sakanushi, Y. Kajitani, The quarter-state sequence (Q-sequence) to represent the floorplan and applications to layout optimization, in: Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, 2000, pp. 829–832.
- [8] B. Yao, H. Chen, C.K. Cheng, R. Graham, Revisiting floorplan representations, in: Proceedings of International Symposium on Physical Design, Wiley, NY, 2001, pp. 138–143.
- [9] E.F.Y. Young, C.C.N. Chu, Z.C. Shen, Twin binary sequences: a non-redundant representation for general non-slicing floorplan, *IEEE Trans. Very Large Integration (VLSI) Syst.* 22 (4) (2003) 457–469 (ISPD 2002).
- [10] F.Y. Young, D.F. Wong, Slicing floorplans with pre-placed modules, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1998, pp. 252–258.
- [11] H. Murata, K. Fujiyoshi, M. Kaneko, VLSI/PCB placement with obstacles based on sequence-pair, in: International Symposium on Physical Design, 1997, pp. 26–31.
- [12] F.Y. Young, D.F. Wong, H.H. Yang, Slicing floorplans with boundary constraints, *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst.* 18 (9) (1999) 1385–1389 Also appeared in ASP-DAC 1999.
- [13] J. Lai, M.-S. Lin, T.-C. Wong, L.-C. Wang, Module placement with boundary constraints using the sequence-pair representation, in: IEEE Asia and South Pacific Design Automation Conference, 2001, pp. 515–520.
- [14] Y. Ma, S. Dong, X. Hong, Y. Cai, C.-K. Cheng, J. Gu, VLSI floorplanning with boundary constraints based on corner block list, in: IEEE Asia and South Pacific Design Automation Conference, 2001, pp. 509–514.
- [15] F.Y. Young, D.F. Wong, Slicing floorplans with range constraints, in: International Symposium on Physical Design, 1999, pp. 97–102.
- [16] Y. Ma, X. Hong, S. Dong, Y. Cai, C.K. Cheng, J. Gu, Floorplanning with abutment constraints and L-shaped/T-shaped blocks based on corner block list, in: Proceedings of the 38th ACM/IEEE Design Automation Conference, 2001, pp. 770–775.
- [17] E.F. Young, C.C. Chu, M.L. Ho, A unified method to handle different kinds of placement constraints in floorplan design, in: Proceedings of the Seventh Asia and South Pacific Design Automation Conference and 15th International Conference on VLSI Design, 2002, pp. 661–667.
- [18] X. Tang, D.F. Wong, Floorplanning with alignment and performance constraints, in: Proceedings of the 39th ACM/IEEE Design Automation Conference, 2002, pp. 848–853.
- [19] H. Xiang, X. Tang, D. Wong, Bus-driven floorplanning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 2003, pp. 66–73.
- [20] X. Tang, D. Wong, FAST-SP: a fast algorithm for block placement based on sequence pair, in: IEEE Asia and South Pacific Design Automation Conference, 2001, pp. 521–526.

Jill H.Y. Law received her B.Sc. degree and M.Phil. degree in Computer Science and Engineering from The Chinese University of Hong Kong (CUHK) in 2003 and 2005, respectively.

Evangeline F.Y. Young received her B.Sc. degree and M.Phil. degree in Computer Science from The Chinese University of Hong Kong (CUHK). She received her Ph.D. degree from The University of Texas at Austin in 1999. Currently, she is an associate professor in the Department of Computer Science and Engineering in CUHK. Her research interests include algorithms and CAD of VLSI circuits. She is now working actively on floorplanning, placement and combinatorial optimization.