Multi-Bend Bus Driven Floorplanning *

Jill H.Y.Law
Department of Computer Science and Engineering
The Chinese University of Hong Kong
hylaw@cse.cuhk.edu.hk

Evangeline F.Y.Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT

In this paper, the problem of bus-driven floorplanning is addressed. Given a set of blocks and the bus specification (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 its 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 the necessary conditions for feasible buses, for which we allow 0-bend, one-bend, or two-bend. Then, we will check whether there are buses that cannot be placed at the same time. Finally, a solution will be generated giving the coordinates of the modules and the buses. Comparing with the results of the algorithm by Xiang et al. [5], the dead space of the floorplan obtained is reduced. Besides, our algorithm can handle buses going through many blocks. For example, if the buses have to go through more than 10 blocks, [5] is not able to generate any solution while our algorithm can still generate solutions of good quality.

Categories and Subject Descriptors: B.7.2 [Integrated Circuit]: Design Aids – Placement and Routing

General Terms: Algorithms, Design

Keywords: Floorplanning, VLSI CAD, Bus planning

1. INTRODUCTION

The floorplanning problem is to plan the positions and shapes of the modules at the beginning of the design cycle to optimize the circuit performance. Interconnect-driven floorplanning is considered to be one of the most important problems in physical design todays. As the complexity of chip design increases, the amount of interconnections between different modules on a chip becomes huge. Bus is a collection of wires, which occupies more spaces than a single wire. Bus routing has become more and more important as the complexity of chip design increases. 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 the placement of buses. 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.

In [2], the authors tried to enforce the abutment constraint based on the Corner Block List (CBL) representation. They showed that the abutment information of the blocks can be deduced from the CBL representation. By applying this abutment constraint to the sub-modules of a L(T)-shaped block, rectilinear blocks can be placed. However, the blocks on a bus are not necessarily abutted. Thus, their approach cannot be used to solve the bus driven-floorplanning problem.

In [1], the authors proposed a unified method to handle different kinds of placement constraints, including preplaced constraint, range constraint, boundary constraint, alignment, abutment and clustering constraint, etc. All these constraints were modelled as a collection of "relative placement constraint" and "absolute placement constraint", 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 is not fixed. Besides, we do not know beforehand whether a bus will be 0-bend, 1-bend, or 2-bend.

Basing on the sequence pair representation, the authors of [4] proposed a method to enforce the alignment constraint and some other performance constraints. Although the alignment constraint mentioned in this paper is not applicable in the bus-driven floorplanning problem, the intuitive idea of deducing the approximate positions of the blocks by looking at the sequence pair is very helpful. In [5], the authors have made use of this to design an intact algorithm to solve the bus-driven floorplanning problem.

In [5], the authors aimed at solving the bus-driven floorplanning problem, based on a simulated annealing framework. Each candidate floorplanning solution would be 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

*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. CUHK4188/03E).

ISPD'05, April 3–6, 2005, San Francisco, California, USA.

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 [5] by allowing 0-bend, 1-bend, and 2-bend buses.

In this paper, this bus-driven floorplanning problem will be re-visited. Unlike [5], our proposed algorithm allows 0-bend, 1-bend, and 2-bend buses. To have a 1-bend bus, one via is used and thus, it can be considered as a 1-via bus. Experimental results have proven that our algorithm can generate solutions with higher 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, [5] is not able to generate any solution while our algorithm can still generate solutions of good quality.

The rest of the 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 the details will be discussed in Section 3. Experimental results will be presented in Section 4. Finally, a conclusion will be given in Section 5.

2. PROBLEM FORMULATION

We assume that there are two layers reserved for bus-routing, one for placing horizontal buses and the other for vertical buses. The bus-driven floorplanning problem can be defined as follows.

Given the following:

1. A set of n blocks $B = \{b_0, b_1, \cdots, b_{n-1}\}$, 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 $U = \{u_0, u_1, \cdots, u_{m-1}\}$, where each bus u_i has a width t_i, $t_i \in \mathbf{R}^+$, and goes 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 all the buses are 0-bend, 1-bend, or 2-bend and each bus u_i can go through all its blocks. There should be no overlapping between any two blocks. Besides, as there are only two layers for bus routing, we have to ensure that there is no overlapping between the horizontal (vertical) components of the buses. The goal is to minimize the chip area and the total bus area.

We will define the meaning of "going through" here. 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 Figure 1. The condition for a vertical component of a bus to go through a set of blocks can be defined similarly.

3. METHODOLOGY

Simulated annealing (SA) will be used to derive a solution. The candidate solution will be evaluated according to 1)the number of buses it can accommodate, 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 shape of the buses by examining the sequence pair. After that, a bus ordering is found. Finally, the flooplan is realized by calculating the coordinates of the blocks and the buses. Details of each step will be presented in the following sections.

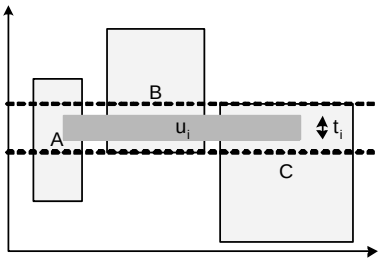

Figure 1: Bus u_i goes through block A, block B, and block C.

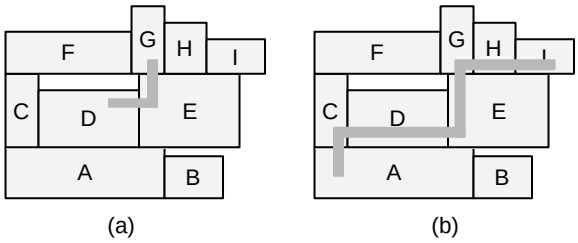

Figure 2: (a) A 1-bend bus. (b) A 3-bend bus.

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 for each infeasible bus.

For example, given a sequence pair ($FGHICDEAB$, $ABCDEFGHI$), a bus u_i that need to go through blocks $\{D, E, G\}$ can be realized as a 1-bend bus. Another bus u_j that need to go through blocks $\{A, C, D, E, G, H, I\}$ has to be realized as a bus of at least three bends (Figure 2), and that 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 $B_i = \{b_1, b_2, \cdots, b_k\}$, 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 goes through blocks $\{A, B, E\}$ from the sequence pair ($ADBCE$, $EBCAD$), we will first extract $sp_i = (ABE, EBA)$ from the sequence pair. Then, we will work on sp_i to check whether u_i can be realized as a 0-bend, 1-bend, or 2-bend bus one after another.

3.1.1 0-Bend Bus Checking

A 0-bend bus is actually a horizontal bus 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 have to be either the same (horizontal bus) or reversed (vertical bus). Let (α, β) be the sequence pair of sp_i, α and β are in reversed order if $\alpha = \beta^R$, where X^R is the reverse of string X. For example, given a sequence pair ($DEFABC$, $ABCDEF$) and a bus u_0 that has to go through the blocks $\{A, B, C\}$, the first step is to extract the corresponding blocks from the sequence pair: $sp_0 = (ABC,$

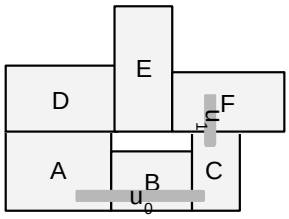

Figure 3: Two valid 0-bend buses, $\{A, B, C\}$ and $\{C, F\}$.

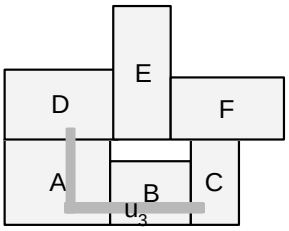

Figure 4: A valid 1-bend bus $\{A, B, C, D\}$

ABC). As the blocks appear in the same order in both sequences, it can be concluded that u_0 can be realized using a 0-bend horizontal bus. For another bus u_1 that has to go through the blocks $\{C, F\}$, the extracted sp_1 is (FC, CF). As the blocks appear in reversed order in the two sequences, it can be realized using a 0-bend vertical bus. This example is illustrated in Figure 3.

3.1.2 1-Bend Bus Checking

1-bend bus is also called 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.

We have to identify the longest common subsequence to form the horizontal component of an L-shaped bus. It must be the LCS but not any common subsequence in sp_i because if taking the longest common subsequence as the horizontal component fails to form a valid L-shape, taking any other shorter subsequences will also fail. Let l_1 be the longest common subsequence of sp_i and l_2 be another common subsequence of shorter length. We can analyze the situation by looking at the two different cases. The first case is that l_2 is not a substring of l_1. Then, a valid L-shape can never be formed with l_2 as the horizontal component because there exist at least two blocks n_1 and n_2 which are in l_1 but not in l_2, and these two blocks must be in left-right relationship with each other. This implies two separate horizontal components and thus, a valid L-shape cannot be formed. Another case is that l_2 is a substring of l_1. Choosing l_2 as the horizontal component may prevent a valid L-shape to be formed as those blocks in l_1 must be in left-right relationship with each other. Therefore, we will pick the longest common subsequence as the horizontal component.

If there exist more than one longest common subsequence l_1 and l_3, picking either one of them will be the same. Let's consider the three different cases according to the number of blocks in l_1 but not in l_3. The first case is that there exist more than one blocks in l_1 but not in l_3 (i.e., there exist more than one blocks in l_3 but not in l_1). Then, the blocks in l_1 but not in l_3 will form a horizontal component, and the blocks in l_3 but not in l_1 will also form another horizontal component. Thus, a valid L-shape cannot be formed no matter which one we pick. The second case is that there is only one block x that is in l_1 but not in l_3 (i.e., there is also another block y that is in l_3 but not in l_1), and that block appears in the middle of l_1, i.e., x is neither the first nor the last block in l_1. Note that the position of x in l_1 must be the same as that of y in l_3. In this case, a T-shape, which is invalid, will be formed if we take l_1 as the horizontal component, as y will be in upper-lower relationship with x. Notice that we cannot take l_3 as the horizontal component neither in this second case for a similar reason. The last case is that there is only one block x that is in l_1 but not in l_3, and that x is the first block or the last block of l_1. A valid 1-bend shape may be formed as x can participate in the vertical component and act as a 'joint' of the two components. In the last case, picking either l_1 and l_3 will be the same. In the following steps, we will regard the first and the last block of the longest common subsequence 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.

Note that even if a bus is consisted of one vertical component and one horizontal component only, there are still several possibilities. The blocks may be in T-shape or +-shape which we consider as invalid. Let $\{a_0, a_1, \cdots, a_x\}$ be the set of blocks that form the vertical component, and $\{b_0, b_1, \cdots, b_y\}$ be the set of blocks that form the horizontal component. If there exists a block b_i that has to be on the left of a_j for some $j \in \{0, 1, \cdots, x\}$, and a block b_k that has to be on the right of a_l for some $l \in \{0, 1, \cdots, x\}$, this bus is in T-shape ($\perp$-shape or +-shape) and is invalid. If there exists a block a_i that has to be on top of b_j for some $j \in \{0, 1, \cdots, y\}$, and a block a_k that has to be at the bottom of b_l for some $l \in \{0, 1, \cdots, y\}$, this bus is in $\vdash$-shape ($\dashv$-shape or +-shape) and is also invalid.

Let's 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 from the sequence pair, which is ($DABC$, $ABCD$). 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, ABC will then 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 is regarded as the vertical component of u_3 (Note that C and D also appear in reversed order and we can choose either A and D or C and D). After checking, u_3 is classified as a valid 1-bend bus. This example is illustrated in Figure 4.

Let's look at another example. given the same sequence pair ($DEFABC$, $ABCDEF$) and another bus u_4 that has to go through the blocks $\{A, B, E, F\}$, we first extract the corresponding blocks sp_4 from the sequence pair, which is ($EFAB$, $ABEF$). The LCS is AB or EF. As there exist more than one longest common subsequence and there are more than one different symbols between them, it is not a

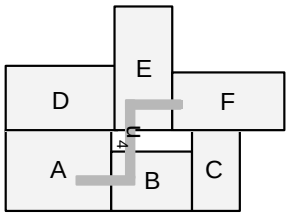

Figure 5: Bus u_4 cannot be realized as a 1-bend bus.

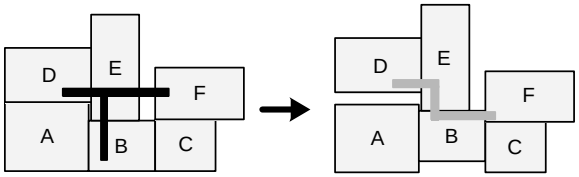

Figure 6: In some cases, a T-shaped bus can be changed into a valid 2-bend bus.

valid 1-bend bus and will proceed to the 2-bend checking. This example is illustrated in Figure 5.

In this 2-bend checking, some buses may be identified to be T-shaped but we will not mark it as infeasible yet since it may form a valid 2-bend bus by adjusting the positions of some blocks. An example is illustrated in Figure 6.

3.1.3 2-Bend Bus Checking

If the 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-shape, mirrored Z-shape, C-shape, or mirrored C-shape. There will be two horizontal (vertical) components and one vertical (horizontal) component in the bus, denoted by HVH and VHV respectively. Assuming the case of HVH, 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 longest common subsequence 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 longest common subsequence will be kept for horizontal component checking. Besides, we have to pick the longest common subsequence but not any other subsequence, and if there are more than one longest common subsequences, picking any one of them will do. The argument is similar to that in 1-bend checking.

After identifying the vertical component, we will classify the remaining blocks of the bus into different relationships with the vertical component. For example, block A from the bus $(ABCDEF, FEDABC)$ will be classified as in the set *Upper*, as A is on top of all the blocks in the vertical component. On the other hand, block F will be classified as in the set *Lower*, as F is below all the blocks in the vertical component. We can deduce these relationships easily from the sequence pair. There are totally eight *position sets* that are defined in a similar fashion, 1) *Upper*, 2) *UpperLeft*, 3) *Left*, 4) *LowerLeft*, 5) *Lower*, 6) *LowerRight*, 7) *Right*, and 8) *UpperRight*.

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 block on the upper-left and lower-right 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 Figure 7. 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.

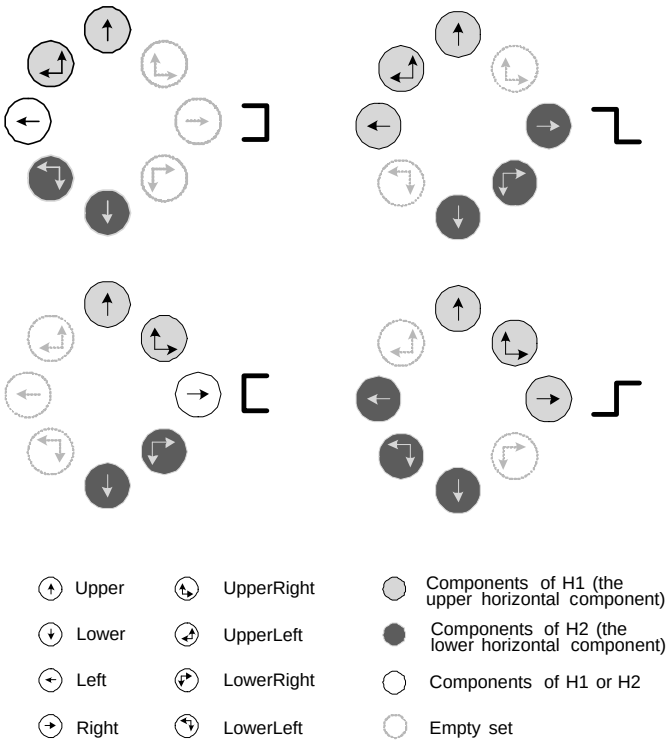

Figure 7: The necessary conditions of the position sets to form a valid 2-bend shape.

The shape validation step for 0-bend, 1-bend, and 2-bend buses can be incorporated into one whole process. The overall algorithm is shown in Figure 8.

3.2 Bus Ordering

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

However, some ordering constraints may be contradictory to each other. An example is shown in Figure 11. In this example, block A is on the left of block B according to the sequence pair, so any vertical bus that goes through A has to be placed on the left of any vertical bus that goes through block B. Block C is also on the left of block D and thus, any vertical bus that goes through C has to be placed on the

```
SHAPE_VALIDATION (int i)
1    k ← 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 ← SUCCESS
7    ELSE
8        Put the remaining blocks into position sets
9        result ← ONE_BEND_CHECK(i)
10       IF result = FAIL
11           result ← 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 ← TWO_BEND_CHECK_HVH(i)
17           END IF
18       END IF
19   END IF
20   RETURN result
```

Figure 8: The pseudo code of shape validation.

```
ONE_BEND_CHECK (int i)
1    result ← FAIL
2    IF |Right| = 1
3        IF |UpperRight|=0∧|LowerRight|=0∧|Lower|=0∧|LowerLeft|=0
4            IF Upper ∪ UpperLeft can form a vertical component
5                Mark as ⌞-shape and result ← SUCCESS
6            END IF
7        ELSE IF |UpperLeft|=0∧|Upper|=0∧|UpperRight|=0∧|LowerRight|=0
8            IF Lower ∪ LowerLeft can form a vertical component
9                Mark as ⌜-shape and result ← SUCCESS
10           END IF
11       END IF
12   ELSE IF |Left| = 1
13       IF |UpperLeft|=0∧|LowerLeft|=0∧|Lower|=0∧|LowerRight|=0
14           IF Upper ∪ UpperRight can form a vertical component
15               Mark as ⌟-shape and result ← SUCCESS
16           END IF
17       ELSE IF |UpperRight|=0∧|Upper|=0∧|UpperLeft|=0∧|LowerLeft|=0
18           IF Lower ∪ LowerRight can form a vertical component
19               Mark as ⌝-shape and result ← SUCCESS
20           END IF
21       END IF
22   END IF
23   RETURN result
```

Figure 9: The pseudo code of 1-bend checking.

```
TWO_BEND_CHECK_HVH (int i)
1    result ← FAIL
2    IF |UpperLeft| = 0 AND |Left| = 0 AND |LowerLeft| = 0
3        IF the blocks in Upper, UpperRight, Right can be horizontal AND
4        the blocks in Lower, LowerRight, Right can be horizontal
5            Mark as C-shape and result ← SUCCESS
6        END IF
7    ELSE IF |UpperRight| = 0 AND |Right| = 0 AND |LowerRight| = 0
8        IF the blocks in Upper, UpperLeft, Left can be horizontal AND
9        the blocks in Lower, LowerLeft, Left can be horizontal
10           Mark as mirrored C-shape and result ← SUCCESS
11       END IF
12   ELSE IF |LowerLeft| = 0 AND |UpperRight| = 0
13       IF the blocks in Upper, UpperLeft, Left can be horizontal AND
14       the blocks in Lower, LowerRight, Right can be horizontal
15           Mark as Z-shape and result ← SUCCESS
16       END IF
17   ELSE IF |UpperLeft| = 0 AND |LowerRight| = 0
18       IF the blocks in Upper, UpperRight, Right can be horizontal AND
19       the blocks in Lower, LowerLeft, Left can be horizontal
20           Mark as mirrored Z-shape and result ← SUCCESS
21       END IF
22   END IF
23   RETURN result
```

Figure 10: The pseudo code of 2-bend checking.

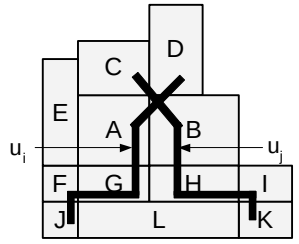

Figure 11: Bus u_i has to be placed on the left of u_j and bus u_j has to be placed on the left of bus u_i.

left of any vertical bus that goes through block D. Problem will occur if there exists two 2-bend buses u_i and u_j, where a vertical component of u_i has to go through block A and block D, and a vertical component of u_j has to go through block B and block C. These two vertical components have to be placed on the left hand side of each other, which is contradictory. 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 is limited to the horizontal components of the buses, where the case for the vertical components can be derived similarly.

Assuming that there are two layers reserved for bus routing, one layer is used to place horizontal buses while the other for vertical buses. We only need to consider constraints between horizontal components and constraints between vertical components separately. For 1-bend or 2-bend buses, we will first break them down into two and three 0-bend components respectively before checking the ordering constraints (Figure 12).

For horizontal buses, we use a graph $G = (V, E)$ to determine whether all the ordering constraints can be satisfied. Each vertex in V represents a 0-bend component, and

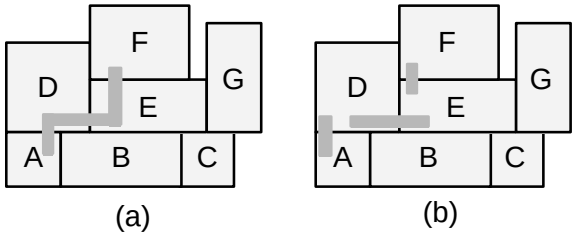

Figure 12: A 2-bend bus is broken down into three 0-bend components before checking the ordering constraint.

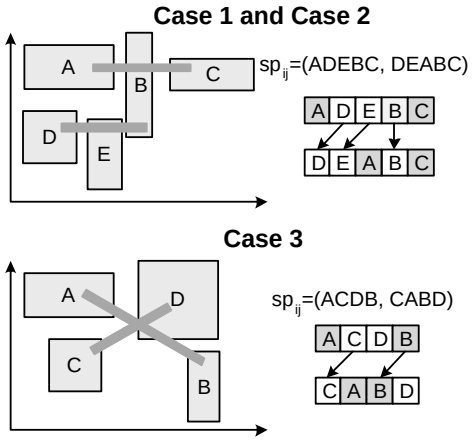

Figure 13: Different cases of the bus ordering constraint.

$E = \{(v_i, v_j)|$ component v_i 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 block B and u_b has to go through block C and block 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 1. 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 2. Let $B_a(B_b)$ be the set of blocks that $u_a(u_b)$ has to go through.

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 (Figure 13):

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, $(v_a, v_b) \in E$.

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, $(v_b, v_a) \in E$.

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 $(v_a, v_b) \in E$.

As some of the buses cannot be placed at the same time, our 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 buses, in a bottom-up (left-right) fashion such that all the remaining buses can be placed one after another successfully according to that order. To do so, we have to examine the graph G_h.

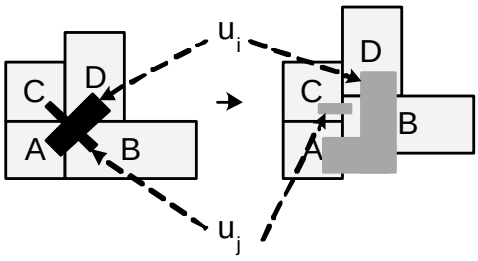

Figure 14: Using a L-shaped bus to resolve the bus ordering conflict.

If contradiction exists, cycle presences. So 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 (buses) to make the graph acyclic. However, this Node-Deleting Problem is proven to be NP-complete [5]. 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 buses, 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. It is obvious that we should not keep partial bus components in the solution, if some parts of the bus are already marked as invalid.

In some cases, bending can help resolving conflicts in the ordering constraint graph. An example is shown in Figure 14. In the example u_i and u_j are horizontal buses that contradict to 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 can only be used for buses that are 0-bend or 1-bend originally, so that one more bend can be added to resolve the conflict by the method illustrated in Figure 14.

After obtaining an acyclic graph, an ordering of the buses can be obtained from a topological order 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 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 those information, we can obtain the coordinates of all the blocks and 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 [3] to construct a floorplan from the sequence pair.

We use the same approach as in [5], which can be described in brief as follows. The following process repeats $O(m)$ 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's consider horizontal buses only. In iteration i, bus u_i will be processed. The coordinates of the blocks that u_i goes through

```
BASIC_ALIGNMENT_H (int i)
1     y_max ← max{y_k : u_i goes through block k}
2     FOR all blocks j u_i goes through
3         IF y_max + t_i - h_j > y_j
4             y_j ← y_max + t_i - h_j
5         END IF
6     END FOR
```

Figure 15: The pseudo code of the basic alignment step for the horizontal buses.

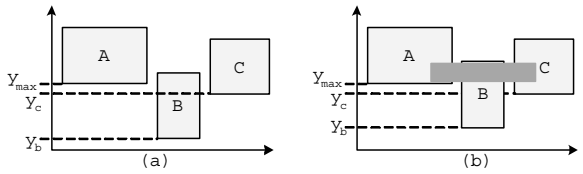

Figure 16: (a) y_{max}, y_b, and y_c are calculated correspondingly. (b) y_b has to be adjusted to let the bus go through.

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 Figure 15. An example is shown in Figure 16.

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, we need to check all its blocks again to ensure that u_i goes through them. We may need to move some of them up in some cases.

3.4 Simulated Annealing

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

3.4.1 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 the block height with the block width. This can be done in constant time.

3.4.2 Cost Function

As mentioned before, the aim of the problem is to 1)accommodate all the buses, 2)minimize the total area of the buses, and 3)minimize the area of the floorplan. Bus area is included in the cost function as bus is actually a collection of wires, and it will be favorable to have the total bus area (interconnect resources) as small as possible. 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, and I is the number of invalid bus, and α, β, and γ are parameters that can be specified by the users.

Table 1: Data Set One.

File	No. of Blocks	No. of Buses	Average/Max. No. of Blocks a Bus Goes Through
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 a Bus Goes Through
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

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.

3.5 Soft Block Adjustment

In order to compare with the results presented in [5], we have added the feature of 'soft block adjustment'. The adjustment is the same as that in [5]. 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 simulated annealing. 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 an originally valid bus is made invalid, the candidate solution will be discarded. Besides, when changing a block width or height, the aspect ratio constraint has to be obeyed.

4. EXPERIMENTAL RESULTS

The proposed algorithm was implemented using the C++ language and the experiments were conducted using an Intel Xeon (2.2 GHz) machine with 1G memory. The test cases are derived from the MCNC benchmarks for floorplanning. In order to compare with the results presented in [5], the same test cases are tried using our proposed algorithm and all the experiments (including those of [5]) are run on the same machine. The results are listed in Table 3. Comparing with the results of [5], the dead space of the floorplan obtained by our algorithm can be reduced on average.

To demonstrate the importance of having 1-bend and 2-bend buses, we have created another set of test cases based on the ami33 and ami49 benchmarks. In these test cases, each bus will go through at least ten blocks. The results are shown in Table 4. For this data set, [5] is not able to generate any solution for most of the test cases, while our algorithm

Table 3: Results of Data Set One.

	[5]		Our Work		Comparison*	
	Time (s)	Dead Space	Time (s)	Dead Space	Time	Dead Space
apte	15	0.72%	30	0.48%	+100%	-33.33%
xerox	15	0.95%	35	0.42%	+133.33%	-55.79%
hp	33	0.62%	51	0.29%	+54.55%	-53.23%
ami33-1	11	0.94%	93	1.00%	+745.45%	+6.38%
ami33-2	92	1.27%	144	1.19%	+56.62%	-6.30%
ami49-1	16	0.85%	71	0.56%	+343.75%	-34.12%
ami49-2	302	0.84%	713	0.58%	+136.09%	-30.95%
ami49-3	285	1.09%	865	0.60%	+203.51%	-44.95%
				Average:	+221.65%	-31.54%

*It is calculated by $[(y_1 - y_0)/y_0] * 100\%$, where y_0 and y_1 are the time (dead space) obtained by [5] and our algorithm respectively.

Table 4: Results of Data Set Two.

	[5]		Our Work		Comparison	
	Time (s)	Dead Space	Time (s)	Dead Space	Time	Dead Space
ami33-3	86	1.81%	32	1.01%	-62.79%	-44.20%
ami33-4	$>10^3$	–	92	1.90%	–	–
ami33-5	$>10^3$	–	95	3.80%	–	–
ami49-4	73	19.34%	88	0.63%	+20.55%	-96.74%
ami49-5	$>10^3$	–	261	1.17%	–	–
ami49-6	$>10^3$	–	140	2.19%	–	–
		Average:	118	1.78%		

can still generate solution with high quality (with average dead space of 1.8% only). We can see that our algorithm can obtain much better results than [5]. As their approach allows only 0-bend bus, it is very difficult to accommodate several buses that go through many blocks.

5. SUMMARY

In this paper, an algorithm to solve the bus-driven floorplanning problem is proposed, allowing 0-bend, 1-bend, and 2-bend buses. Experimental results show that our approach is very 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.

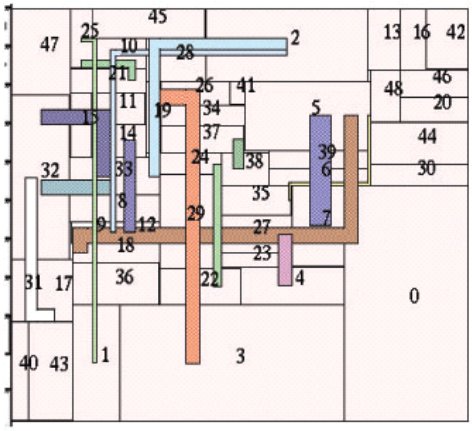

Figure 17: Result packing of ami49-3.

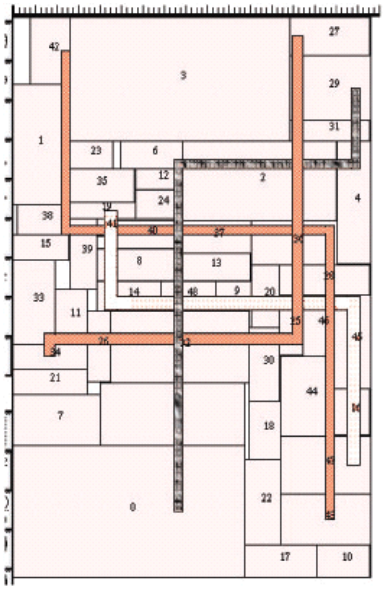

Figure 18: Result packing of ami49-6.

6. ACKNOWLEDGEMENT

We would like to thank the authors of [5] who have kindly given us their program and test cases such that we can conduct the experiments.

7. REFERENCES

[1] Evangeline F.Y.Young, Chris C.N.Chu and M.L.Ho. "A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design," In *with EDA Technofair Design Automation Conference Asia and South Pacific*, 2002.

[2] Yuchun Ma, Xianlong Hong, Sheqin dong, Yici Cai, Chung-Kuan Cheng and Jun Gu. "Floorplanning with Abutment Constraints and L-Shaped/T-Shaped Blocks based on Corner Block List," In *Annual ACM IEEE Design Automation Conference*, 2001.

[3] Xiaoping Tang and D.F.Wong. "FAST-SP: a fast algorithm for block placement based on sequence pair," In *Conference on Asia South pacific Design Automation*, 2001.

[4] Xiaoping Tang and D.F.Wong. "Floorplanning with Alignment and Performance Constraint," In *Annual ACM IEEE Design Automation Conference*, 2002.

[5] Hua Xiang, Xiaoping Tang and Martin D.F.Wong. "Bus-Driven Floorplanning," In *International Conference on Computer Aided Design*, 2003.