Twin Binary Sequences: A Non-redundant Representation for General Non-slicing Floorplan

Evangeline F.Y. Young, Chris C.N. Chu and Zion Cien Shen

Abstract

The efficiency and effectiveness of many floorplanning methods depend very much on the representation of the geometrical relationship between the modules. A good representation can shorten the searching process so that more accurate estimations on area and interconnect costs can be performed. Non-slicing floorplan is the most general kind of floorplan that is commonly used. Unfortunately, there is not yet any complete and non-redundant topological representation for non-slicing structure.

In this paper, we will propose the first representation of this kind. Like some previous work [14], we have also made used of mosaic floorplan as an intermediate step. However, instead of including a more than sufficient number of extra dummy blocks in the set of modules (that will increase the size of the solution space significantly), our representation allows us to insert an *exact* number of *irreducible empty rooms* to a mosaic floorplan such that *every* non-slicing floorplan can be obtained uniquely from *one and only one* mosaic floorplan. The size of the solution space is only $O(n!2^{3n}/n^{1.5})$, since mosaic floorplan is used as an intermediate step, but every non-slicing floorplan can be generated uniquely and efficiently in linear time without any redundant representation.

1 Introduction

Floorplan design is a major step in the physical design cycle of VLSI circuits to plan the positions and shapes of a set of modules on a chip in order to optimize the circuit performance. As technology moves into the deep-submicron era, circuit sizes and complexities are growing rapidly, and floorplanning has become ever more important than before. Area minimization used to be the most important objective in floorplan design, but today, interconnect issues like delay, total wirelength, congestion and routability have instead become the major goal for optimization. Unfortunately, floorplanning problems are NP-complete. Many floorplanners employ methods of perturbations with random searches and heuristics. The efficiency and effectiveness of these kinds of methods depend very much on the representation of the geometrical relationship between the modules. A good representation can shorten the searching process and allows fast realization of the floorplan so that more accurate estimations on area and interconnect costs can be performed.

1.1 Previous Works

The problem of floorplan representation has been studied extensively. There are three types of floorplan: slicing, mosaic and non-slicing. A slicing floorplan is a floorplan that can be obtained by recursively cutting a rectangle into two by using a vertical line or a horizontal line. Normalized Polish expression [12] is the most popular method to represent slicing floorplan. This representation can describe any slicing structure with no redundancy. An upper bound on its solution space is $O(n!2^{3n-3}/n^{1.5})$. For general floorplan

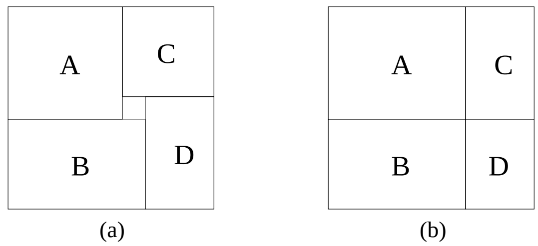

Figure 1: Structures that cannot be represented in a mosaic floorplan.

that is not necessarily slicing, there was no efficient representation other than the constraint graphs until the sequence pair (SP) [7] and the bounded-sliceline grid (BSG) [8] appeared in the mid 90's. The SP representation has been widely used because of its simplicity. Unfortunately, there are a lot of redundancies in these representations. The size of the solution space of SP is $(n!)^2$ and that of BSG is $n!C(n^2, n)$. This drawback has restricted the applicability of these methods in large scale problems. O-tree [4] and B*-tree [1] are later proposed to represent compacted (admissible) non-slicing floorplan. They have very small solution space of $O(n!2^{2n-2}/n^{1.5})$ and can give a floorplan in linear time. However, they describe only partial topological information and module dimensions are needed to give a floorplan exactly. The representation is not unique, and a single O-tree or B*-tree representation, depending on the module dimensions, can lead to more than one floorplan with modules of different topological relationships with each other.

The paper [5] proposes a new type of floorplan called mosaic floorplan. A mosaic floorplan is similar to a general non-slicing floorplan except that it does not have any unoccupied room (Figure 1(a)) and there is no *crossing cut* in the floorplan (Figure 1(b)). A representation called Corner Block List (CBL) is proposed to represent mosaic floorplan. This representation has a relatively small solution space of $O(n!2^{3n})$ [1] and the time complexity to realize a floorplan from its representation is linear. However, some corner block lists do not correspond to any floorplan. As a remedy to the weakness that some non-slicing structures cannot be represented (e.g., Figure 1(a)), CBL is extended by including dummy blocks of zero area in the set of modules. In order to represent all non-slicing structure, $O(n^2)$ of such dummy blocks are used and this has increased the size of the solution space significantly [14]. In the paper [10], a new representation called Q-sequence is proposed to represent mosaic floorplan, which is later enhanced in the paper [15] by including empty rooms. It is also proved in [15] that the number of empty rooms required is upper bounded by $n - \lfloor\sqrt{4n-1}\rfloor$ where n is the number of modules.

1.2 Our Contributions

Although the problem of floorplan representation has been studied extensively, and numerous floorplan representations have been proposed in recent years, it is still practically useful and theoretically interesting to find a complete (i.e., every non-slicing floorplan can be represented) and non-redundant topological representation for general non-slicing structure. In this paper, we will present such a representation, the Twin Binary Sequences. This will mark the first of this kind. Like some previous work [14], we have made

[1] In [5], the paper claims without proof that the size of solution space for Corner Block List is $O(n!2^{3n}/n^{1.5})$. However, we believe that the correct size of CBL solution space should be $\Theta(n!2^{3n})$. In the CBL algorithm, the corner block list (S, L, T) are perturbed randomly and independently in the simulated annealing process. There are $n!$ combinations for S, 2^{n-1} combinations for L, and 2^{2n-3} combinations for T. So the total number of combinations is $\Theta(n!2^{3n})$.

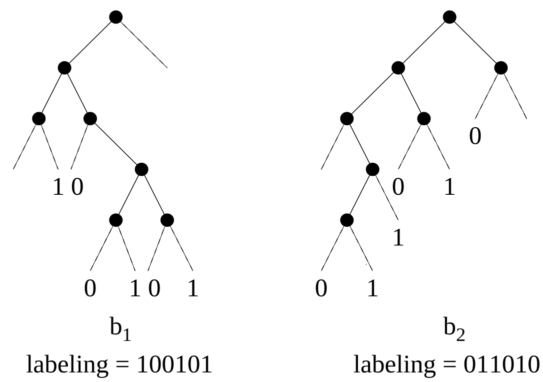

Figure 2: An example of a twin binary trees.

use of mosaic floorplan as an intermediate step to represent a non-slicing structure. However, instead of including an extra number of dummy blocks in the set of modules, the representation allows us to insert an exact number of *irreducible empty rooms* to a mosaic floorplan such that every non-slicing structure can be generated uniquely and non-redundantly. Besides, the representation can give a floorplan efficiently in linear time. We have studied the relationship between mosaic and non-slicing floorplan and have proved that the number of empty rooms needed to be inserted into a mosaic floorplan to obtain a non-slicing structure is tightly bounded by $\Theta(n)$ where n is the number of modules. [2]

In the following section, we will define twin binary sequences, and show how a floorplan can be constructed from this representation in linear time. In section 3, we will show how this representation can be used to describe non-slicing structure with the help of a fast empty room insertion process. We will also present some interesting results on the relationship between mosaic and general floorplan. In section 4 and 5, we will discuss our floorplanner based on simulated annealing and the experimental results will be shown.

2 Twin Binary Sequences (TBS) Representation

In the paper [13], Yao, et al. first suggest that the Twin Binary Trees (TBT) can be used to represent mosaic floorplan. They have shown a one-to-one mapping between mosaic floorplan and TBT. We have made use of TBT in our representation. Recall that the definition of twin binary trees comes originally from the paper [3] as follows:

Definition 1 *The set of twin binary trees with n nodes $\mathrm{TBT}_n \subset \mathrm{Tree}_n \times \mathrm{Tree}_n$ is the set:*

$$\mathrm{TBT}_n = \{(b_1, b_2) | b_1, b_2 \in \mathrm{Tree}_n \text{ and } \Theta(b_1) = \Theta^c(b_2)\}$$

where Tree_n is the set of binary trees with n nodes, and $\Theta(b)$ is the labeling of a binary tree b as follows. Starting with an empty sequence, we perform an inorder traversal of the tree b. When a node with no left child is reached, we will add a bit 0 to the sequence, and when a node with no right child is reached, we will add a bit 1 to the sequence. The first 0 and the last 1 will be omitted. Θ^c is the complement of Θ obtained by interchanging all the 0's and 1's in Θ. An example of a twin binary trees is shown in Figure 2.

[2] Together with the upper bound result in [15], the tight bound can be further improved to $\Theta(n - 2\sqrt{n})$.

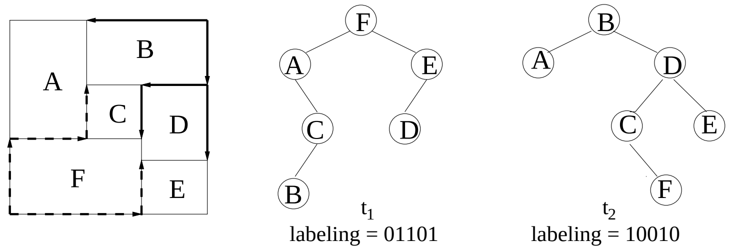

Figure 3: Building a twin binary trees from a mosaic packing.

Instead of using an arbitrary pair of trees (which may not be twin binary to each other) directly, we used a 4-tuple $s = (\pi, \alpha, \beta, \beta')$ called a twin binary sequences to represent a mosaic floorplan with n modules where π is a permutation of the module names, α is sequence of $n-1$ bits, and β and β' are sequences of n bits. The properties of these bit sequences will be described in details in section 2.2. This 4-tuple can be one-to-one mapped to a pair of binary trees t_1 and t_2 such that t_1 and t_2 must be twin binary to each other and they together represent a mosaic floorplan uniquely. Most importantly, we are then able to insert empty rooms to t_1 and t_2 at the right places to give a non-slicing floorplan. We proved that every non-slicing structure can be obtained by this method from one and only one mosaic floorplan. In order to motivate the idea of our new representation, we will first show how a twin binary trees can be obtained from a mosaic floorplan in the following subsection.

2.1 From Floorplan to Twin Binary Trees

Given a mosaic floorplan F, we can obtain a pair of twin binary trees t_1 and t_2 by traveling along the slicelines of F. An example is shown in Figure 3. To construct t_1, we start from the module at the lower left corner and travel upward (left subtree) and to the right (right subtree). Whenever the lower left corner of another module x is reached, a node labeled x will be inserted into the tree and the process will be repeated starting from module x until all the modules in the floorplan are visited. The tree t_2 can be built similarly by starting from the module at the upper right corner and travel downward (right subtree) and to the left (left subtree). Similarly, whenever the upper right corner of another module y is reached, a node labeled y will be inserted into the tree and the process will be repeated starting from y until all the modules are visited. The paper [13] has shown that the pair of trees built in this way must be twin binary to each other, and there is a one-to-one mapping between mosaic floorplan and twin binary trees. We observed that the inorder traversal of the two binary trees constructed by the above method must be the same. Let us look at the example in Figure 3. We can see that the inorder traversals of both t_1 and t_2 are $ABCFDE$. We have proved the following observation that helps in defining the Twin Binary Sequences representation:

Observation 1 A pair of binary trees t_1 and t_2 can be constructed from a mosaic floorplan by the above method if and only if (1) they are twin binary to each other, i.e., $\Theta(t_1) = \Theta^c(t_2)$, and (2) their inorder traversals are the same.

Proof: (*if part*) This part can be proved by induction on the number of modules in the floorplan. The base case occurs when there is only one module in the floorplan and conditions (1) and (2) follow trivially. Assume that these conditions are true when there are not more than $k \geq 1$ modules in the floorplan. Consider a floorplan F with $k+1$ modules. Let the pair of binary trees constructed from F by the above method be

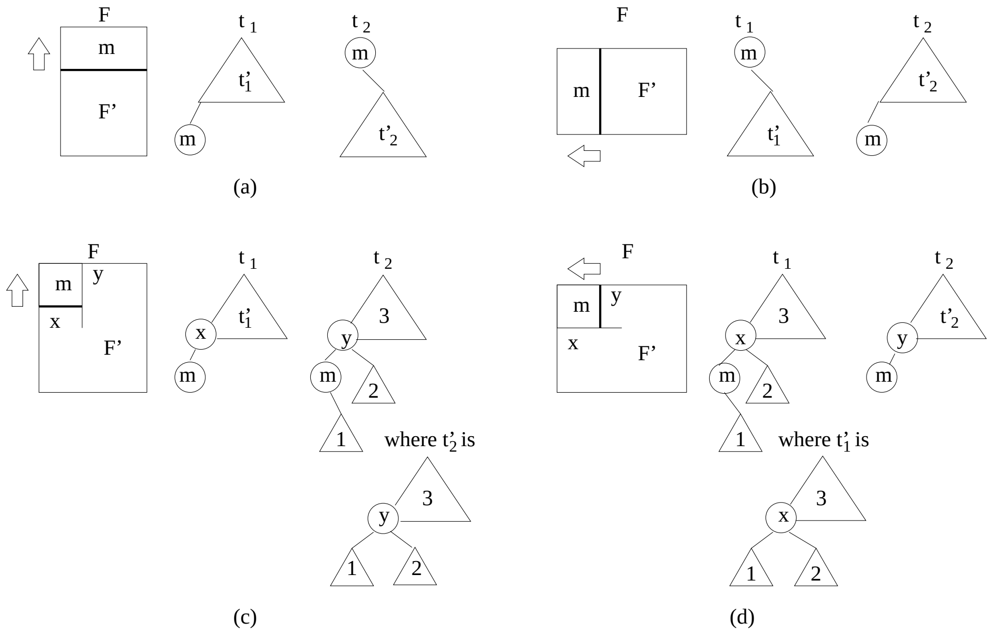

Figure 4: Proof of Observation 1 (if part).

t_1 and t_2. Consider the module m at the upper left corner of F. There are only four possible configurations for the position of m in F as shown in Figure 4. In each case, let F' be the floorplan obtained by *sliding* module m out of F by moving the thickened sliceline in the direction shown. Let t'_1 and t'_2 be the pair of binary trees constructed from F' by the above method. Since floorplan F' has only k modules, t'_1 and t'_2 satisfy conditions (1) and (2) according to the hypothesis, i.e., $\Theta(t'_1) = \Theta^c(t'_2)$, and their inorder traversals are the same. From Figure 4, we can see that in case (a) and (c), $\Theta(t_1) = 1\Theta(t'_1)$, $\Theta(t_2) = 0\Theta(t'_2)$ and the inorder traversal of t_1 (t_2) is the same as that obtained by appending m in front of the inorder traversal of t'_1 (t'_2). Similarly, in case (b) and (d), $\Theta(t_1) = 0\Theta(t'_1)$, $\Theta(t_2) = 1\Theta(t'_2)$ and the inorder traversal of t_1 (t_2) is the same as that obtained by appending m in front of the inorder traversal of t'_1 (t'_2). Therefore t_1 and t_2 also satisfy conditions (1) and (2).

(*only if part*) Again, this part is proved by induction. The base case occurs when there is only one node in the pair of binary trees. If both conditions (1) and (2) are true (note that condition (1) must be true since there is only one node in the trees and their labelings are both empty), their nodes are labeled the same and they correspond to a packing with only one module. Assume that this statement is true for any pair of trees with $k \geq 1$ nodes, i.e., inorder traversal of length k and labeling of length $k-1$. Consider a pair of trees (t_1 and t_2) with inorder traversal $m_1 m_2 \ldots m_{k+1}$, and labelings $b_1 b_2 \ldots b_k$ and $\bar{b}_1, \bar{b}_2, \ldots \bar{b}_k$. There are two cases as shown in Figure 5 according to the value of the bit b_1. In both cases, the inorder traversal $m_2 m_3 \ldots m_{k+1}$, and the bit sequences $b_2 b_3 \ldots b_k$ and $\bar{b}_2, \bar{b}_3, \ldots \bar{b}_k$ will correspond to a floorplan F' according to the hypothesis. We can obtain a floorplan F from F' by putting the module m_1 on the right (case (a)) or at the top (case (b)). F will correspond to a pair of trees with inorder traversal $m_1 m_2 \ldots m_{k+1}$, and labelings $b_1 b_2 \ldots b_k$ and $\bar{b}_1 \bar{b}_2 \ldots \bar{b}_k$. We can choose between case (a) and (b) depending on the value of b_1. Therefore this only if statement is also true when there are $k+1$ nodes in the pair of trees. $\square$

If we extend a tree t by adding a left child of bit 0 to every node (except the leftmost node) that has no

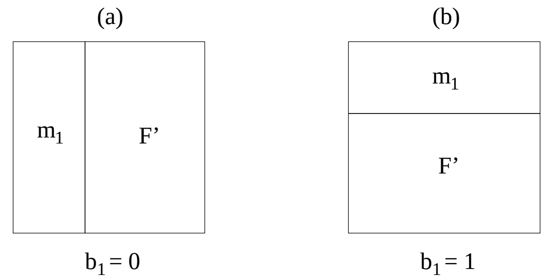

Figure 5: Proof of Observation 1 (only if part).

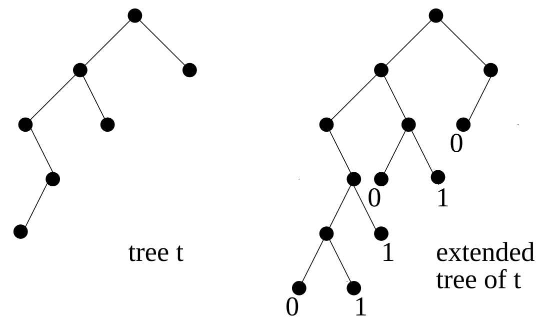

Figure 6: An example of an extended tree.

left child and by adding a right child of bit 1 to every node (except the rightmost node) that has no right child, the tree obtained is called an *extended* tree of t. An example of an extended tree is shown in Figure 6. Notice that the inorder traversal of the extended tree of t will be $m_1\alpha_1 m_2\alpha_2 \dots \alpha_{n-1} m_n$ where $m_1 m_2 \dots m_n$ are the inorder traversal of t and $\alpha_1\alpha_2 \dots \alpha_{n-1}$ is the labeling of t. Observation 1 can be restated as follows:

Observation 2 A pair of binary trees t_1 and t_2 can be constructed from a mosaic floorplan by the above method if and only if the inorder traversal of their extended trees are the same except that all the bits are complemented.

2.2 Definition of Twin Binary Sequences

From observation 1, we know that a pair of binary trees t_1 and t_2 are *valid* (i.e., corresponding to a packing) if and only if their labelings are complement of each other and their inorder traversals are the same. However, the labeling and the inorder traversal are not sufficient to identify a unique pair of t_1 and t_2. Given a permutation of module names π and a labeling α, there can be more than one valid pairs of t_1 and t_2 such that their inorder traversals are π and $\Theta(t_1) = \Theta^c(t_2) = \alpha$. In order to identify a pair of trees uniquely, we need two additional bit sequences β and β' for t_1 and t_2 respectively such that the i^{th} bit in β and β' tells whether the i^{th} module in π is the left child (when the bit is 0) or the right child (when the bit is 1) of its parent in t_1 and t_2 respectively. These bits are called the *directional bits*. If module k is the root of a tree, its directional bit will be assigned to zero.

For a binary tree t, its labeling sequence $\alpha = \alpha_1\alpha_2 \dots \alpha_{n-1}$ and its directional bit sequence $\beta = \beta_1\beta_2 \dots \beta_n$ must satisfy the following conditions:

(1) In the bit sequence $\beta_1\alpha_1\beta_2\ldots\alpha_{n-1}\beta_n$, the number of 0's is one more than the number of 1's.

(2) For any prefix of the bit sequence $\beta_1\alpha_1\beta_2\ldots\alpha_{n-1}\beta_n$, the number of 0's is more than or equal to the number of 1's.

We proved the following lemmas which show that conditions (1) and (2) are necessary and sufficient for a pair of labeling sequence α and directional bit sequence β to correspond to a binary tree.

Lemma 1 *For any binary tree, its labeling sequence α and directional bit sequence β must satisfy conditions (1) and (2).*

Proof: Given a binary tree t, the bit sequence $\beta_1\alpha_1\beta_2\ldots\alpha_{n-1}\beta_n$ is the inorder traversal of the extended tree t' of t (with the internal nodes labeled by their directional bits). To verify condition (1), notice that each internal node of t' has two children, one is labeled by 0 and the other one is labeled by 1. We assume that the root is labeled by 0. Therefore condition (1) must be satisfied. To verify condition (2), notice that for any two children having the same parent, the child labeled 0 is always visited first in the inorder traversal. Therefore condition (2) must be satisfied. $\square$

Lemma 2 *For any binary sequences α of $n-1$ bits and β of n bits satisfying conditions (1) and (2), there exists a unique binary tree t such that the labeling sequence of t is α and the directional bit sequence of t is β.*

Proof: The uniqueness can be proved by induction on the number of modules. The claim is trivially true when there is only one module, i.e., when $n = 1$. Assume that the claim holds when the number of modules is at most k, i.e., when $n \leq k$. Consider the case when $n = k+1$. Given a pair of binary sequences $\alpha = \alpha_1\alpha_2\ldots\alpha_k$ and $\beta = \beta_1\beta_2\ldots\beta_{k+1}$, we can reduce the problem to the case with k or less modules as follows. First of all, we append a bit $\alpha_0 = 0$ in front of α and a bit $\alpha_{k+1} = 1$ at the end of α. Then there exists at least one i such that $\alpha_{i-1} = 0$ and $\alpha_i = 1$. This is a place for a leaf node where the leaf is either a left (when $\beta_i = 0$) or a right (when $\beta_i = 1$) child of its parent. We use S to denote the set of all such locations, i.e., $S = \{i | (1 \leq i \leq k+1) \cap (\alpha_{i-1} = 0) \cap (\alpha_i = 1)\}$. Let α' be the binary sequence obtained from α by replacing $\alpha_{i-1}\alpha_i$ by β_i for all $i \in S$, and β' be the binary sequence obtained from β by deleting β_i for all $i \in S$. Notice that the first bit of α' must be 0 and the last bit must be 1, i.e., we can write α' as $0\alpha''1$. According to the induction hypothesis, there exists a unique binary tree t' such that the labeling sequence of t' is α'' and the directional bit sequence of t' is β'. The tree t for the original pair of binary sequences $\alpha = \alpha_1\alpha_2\ldots\alpha_{k-1}$ and $\beta = \beta_1\beta_2\ldots\beta_k$ can be constructed uniquely from t' by inserting a leaf node corresponding to the module π_i to the position of bit β_i for all $i \in S$. Therefore the uniqueness still holds when $n = k+1$. $\square$

Now, we can define a twin binary sequences representation. A twin binary sequences s for n modules is a 4-tuple:

$$s = (\pi, \alpha, \beta, \beta')$$

where π is a permutation of the n modules, both α and β, and α^c (the complement of α) and β' satisfy conditions (1) and (2). We have proved the following two theorems that show the one-to-one mapping between twin binary trees and mosaic floorplan.

Theorem 1 *The mapping between twin binary sequences and twin binary trees is one-to-one.*

Proof: Given a pair of twin binary trees, we can construct one unique twin binary sequences according to the definition in section 2.1. On the other hand, if we are given a twin binary sequences $s = (\pi, \alpha, \beta, \beta')$, according to Lemma 2, there exists a unique binary tree t (t') such that the labeling sequence of t (t') is α (α^c) and the directional bit sequence of t (t') is β (β'). Since $\Theta(t) = \Theta^c(t')$, t and t' are twin binary. We can then label their nodes according to the inorder traversal π. This is the unique pair of twin binary trees t and t' corresponding to s. Therefore the mapping between twin binary sequences and twin binary trees is one-to-one. □

Theorem 2 *The mapping between twin binary sequences and mosaic floorplan is one-to-one.*

Proof: The one-to-one mapping between twin binary sequences and mosaic floorplan follows from Theorem 1 and the proof in paper [13] that the mapping between twin binary trees and mosaic floorplan is one-to-one. □

2.3 From TBS to Floorplan

2.3.1 Algorithm for Floorplan Realization

In order to realize a floorplan from its TBS representation efficiently, we devised an algorithm that only needs to scan the sequences once from right to left to construct the packing. We will construct the floorplan by inserting the modules one after another following the π sequence in the reversed order. A simple example illustrating the steps of the algorithm is given in Figure 7. At the beginning, we will put the last module of the π sequence, i.e., module D, into the packing P. We will then insert the other modules one after another. The next module to be considered after D is $\pi_4 = E$. Since $\alpha_4 = 0$, we will look at the sequence β and find the closest bit "1" on the right of β_4, i.e., β_5. We will then add module E into P from the left pushing D (since $\alpha_5 = D$) to the right as shown in Figure 7(b) and delete bit β_5 from β. The next module to be considered after E is $\pi_3 = C$. Since $\alpha_3 = 1$, we will look at the sequence β' and find the closest bit "1" on the right of β'_3, i.e., β'_4. We will then add module C into P from above pushing E (since $\alpha_4 = E$) down as shown in Figure 7(c) and delete bit β'_4 from β'. These steps repeat until the whole sequence π is processed and a complete floorplan is obtained.

Algorithm TBStoFloorplan
Input: *A TBS* $s = (\pi, \alpha, \beta, \beta')$
Output: *Packing* P *corresponding to* s

Begin
1. *Append* α *with bit '1', i.e.,* $\alpha_n = 1$.
2. *Initially, we have only module* π_n *in* P.
3. *For* $i = n - 1$ *down to* $i = 1$*:*
4. *If* ($\alpha_i = 0$)*:*
5. *Find the smallest* k *s.t.* $i < k \leq n$ *and* $\beta_k = 1$.
6. *Note that the set* S *of modules* $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_k$ *(those with corresponding* β *bit*

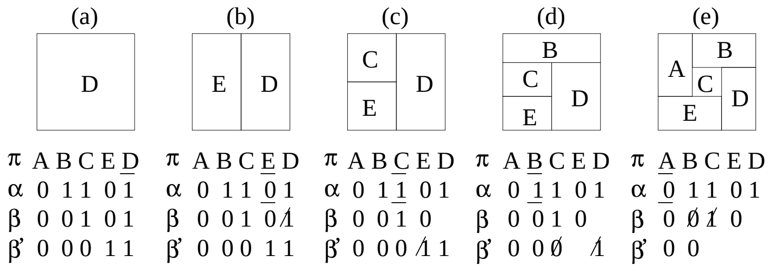

Figure 7: A simple example of constructing a floorplan from its TBS.

not deleted yet) will be lying on the left boundary of P. Add module π_i to P from the left, pushing those modules in S to the right.

7. Delete $\beta_{i+1}, \beta_{i+2}, \ldots, \beta_k$ from β.

8. If $(\alpha_i = 1)$:

9. Find the smallest k s.t. $i < k \leq n$ and $\beta'_k = 1$

10. Note that the set S of modules $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_k$ (those with corresponding β' bit not deleted yet) will be lying on the top boundary of P. Add module π_i to P from above, pushing those modules in S down.

11. Delete $\beta'_{i+1}, \beta'_{i+2}, \ldots, \beta'_k$ from β'.

End

2.3.2 Proof of Correctness

The correctness of the above algorithm on floorplan realization can be proved by the following lemma and theorem:

Lemma 3 *In the for-loop of the above algorithm, when we scan to a point i where $1 \leq i \leq n-1$ and $\alpha_i = 0$ ($\alpha_i = 1$), the corresponding node π_i in t_1 (t_2) has a right child π_j and all the nodes in t, where t is the subtree of t_1 (t_2) rooted at π_j, have been scanned immediately before π_i. In addition, any node $\pi_k \in t$ where $k \neq j$ and $\beta_k = 1$ ($\beta'_k = 1$) will have its β (β') bit deleted.*

Proof: W.l.o.g., we only prove the case when $\alpha_i = 0$. The case when $\alpha_i = 1$ can be proved similarly. The proof can be done by induction on i. The base case is when $i = n-1$. If $\alpha_{n-1} = 0$, π_{n-1} must have a right child in t_1 according to the definition of TBS. Let t be the right subtree of π_{n-1} in t_1. Since we are performing the inorder traversal in the reversed order, the nodes in t must have been scanned immediately before π_{n-1}. In this base case, there is only one node (π_n) in t which is the right child of π_{n-1} and $\beta_n = 1$. Therefore the statement is true for this base case.

Assume that the statement is true when $i \geq p$ for some $1 < p \leq n-1$. Consider the case when $i = p-1$. If $\alpha_{p-1} = 0$, similarly, π_{p-1} must have a right child π_j in t_1 according to the definition of TBS. Let t be the subtree of t_1 rooted at π_j. Since we are performing the inorder traversal in the reversed order, the nodes in t must have been scanned immediately before π_{p-1}. Let them be $\pi_p, \pi_{p+1}, \ldots, \pi_{p+m-1}$ where m is the size of t. (Note that $p \leq j \leq p+m-1$.) If there is any node π_k in t where $k \neq j$ and $\beta_k = 1$, β_k must have been deleted when the scan reaches π_{p-1}. This is because if $\beta_k = 1$, π_k is the right child of its parent π_l in t_1 and π_l must also be in t. According to the inductive hypothesis, when we scan

Figure 8: Proof of Theorem 3.

to π_l, we will find that $\alpha_l = 0$ (since π_l has a right child π_k in t_1) and π_k will be the only node in the right subtree of π_l in t_1 such that $\beta_k = 1$ at that moment. Since the nodes in the right subtree of π_l will be lying immediately in front of π_l in the reversed inorder traversal, we will delete all the β bits up to and including β_k. Therefore, when we scan to π_{p-1}, any node $\pi_k \in t$ where $k \neq j$ and $\beta_k = 1$ will have its β bit deleted. □

Theorem 3 *The algorithm TBStoFloorplan can convert a TBS to its corresponding floorplan correctly.*

Proof: Again, the proof can be done by induction on the number of modules. The base case occurs when there are only two modules in the packing. There can only be two different mosaic packings with two modules, one with the two modules lying side by side and the other one with the two modules piling up vertically. It is easy to show that the algorithm is correct in both situations.

Assume that the algorithm is correct when there are n modules in the floorplan for some $n \geq 2$. Consider the case when there are $n+1$ modules. W.l.o.g., we assume that $\alpha_1 = 0$. The case when $\alpha_1 = 1$ can be proved similarly. Since $\alpha_1 = 0$, the upper left module $A = \pi_1$ has a right child $B = \pi_j$ in t_1 and A should be packed in one of the two ways shown in Figure 8 in the floorplan F . Assume that the TBS of F is $s = (\pi, \alpha, \beta, \beta')$ where $\pi = \pi_1, \pi_2, \ldots, \pi_{n+1}$, $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$, $\beta = \beta_1, \beta_2, \ldots, \beta_{n+1}$ and $\beta' = \beta'_1, \beta'_2, \ldots, \beta'_{n+1}$. Consider sliding module A out of the floorplan F (Figure 9) in the direction shown to obtain a floorplan F_1 with n modules. Note that the TBS s_1 for F_1 can be obtained from s by changing β_j from 1 to 0 and removing π_1, α_1, β_1 and β'_1 from π, α, β and β' respectively. Since F_1 has only n modules and the algorithm can construct the floorplan F_1 correctly from s_1 according to the inductive hypothesis.

Consider the sequence of operations of the algorithm on s. The first $n-1$ steps of the for-loop will be the same as that for s_1. The two sequences of operations are the same although β_j is changed from 1 to 0 because all the modules lying between B and A in the inorder sequence π are in the left subtree of B in t_1. After scanning pass B, if there is an $\alpha_k = 0$ where $1 < k < j$, we will only delete those β bits up to and including β_l where π_l is the right child of π_k according to Lemma 3. Thus, the value of β_j will not affect the first $n-1$ steps of the for loop. That means, when we reach A, the intermediate floorplan obtained is the same as F_1. At A, since $\alpha_1 = 0$, according to the above lemma, $B = \pi_j$ will be the only module in the left subtree of A in t_1 such that $\beta_j = 1$. Therefore we will delete all the β bits up to and including β_j and insert module A to F_1 from the left, pushing to the right all the modules from the upper left corner of F_1 down to and including module B. We will get back the correct packing F. Therefore the statement is also true when there are $n+1$ modules in the packing. □

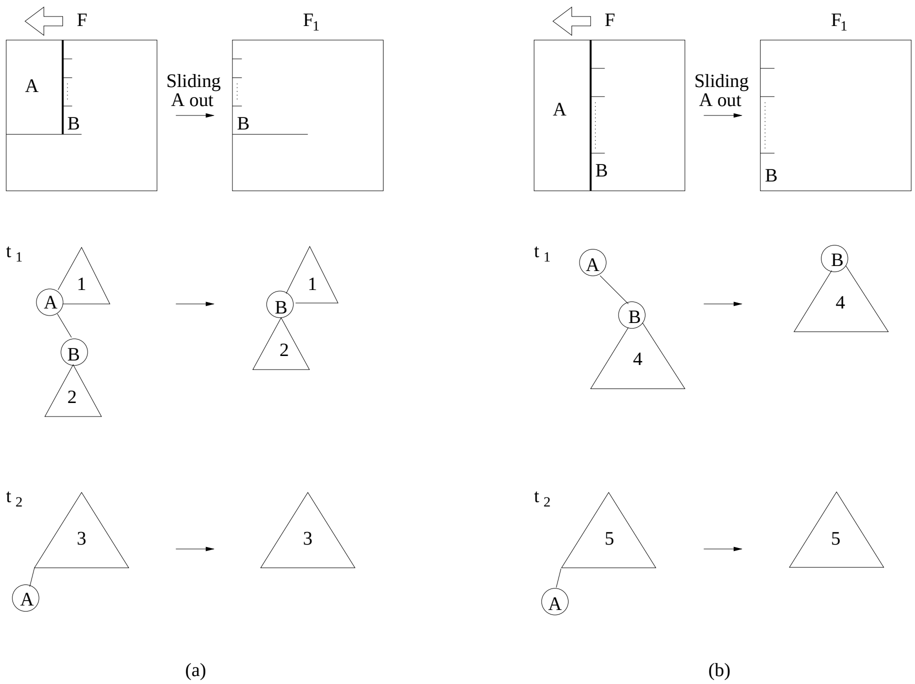

Figure 9: Proof of Theorem 3.

2.4 Size of Solution Space

The TBS representation is a complete and non-redundant representation for mosaic floorplan. Thus the number of different TBS configurations should give the Baxter number [13]. Baxter number can be written analytically as a complicated summation (equation 3.1 in [13]). However, there is no known simple closed form expression for the Baxter number. In the following, an upper bound on the number of different TBS configurations (i.e., on the Baxter number) is presented.

Consider a TBS $(\pi, \alpha, \beta_1, \beta_2)$ for n modules. α and β_1 uniquely specify a rooted ordered binary tree. Thus the number of combinations of α and β_1 is given by the Catalan number. Since the number of combinations for π is $n!$, the number of combinations for β_2 is upper-bounded by $O(2^n)$, and the Catalan number is upper-bounded by $O(2^{2n}/n^{1.5})$, the number of different TBS configurations is bounded by $O(n!2^{3n}/n^{1.5})$.

3 Extension to General Floorplan

3.1 Empty Room in Mosaic Floorplan

A twin binary sequences s represent a mosaic floorplan F. Now we want to insert an exact number of empty rooms at the right places in F to obtain a corresponding non-slicing floorplan F' such that every non-slicing floorplan can be generated by this method from one mosaic floorplan non-redundantly. There are two kinds of empty rooms. One is resulted because a big room is assigned to a small module. This kind of empty room is called *reducible empty room*. An example is shown in Figure 10(a). Another kind of empty room is called *irreducible empty room* and is defined as follows:

Definition 2 *An irreducible empty room is an empty room that cannot be removed by merging with another* *room in the packing.*

An example of an irreducible empty room is shown in Figure 10(b). We observed that an irreducible empty room must be of *wheel* shape and its four *adjacent rooms* (the rooms that share a T-junction at one of its corners) must not be irreducible empty rooms themselves:

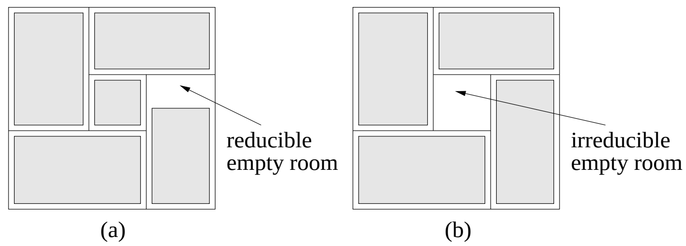

Figure 10: Examples of reducible and irreducible empty rooms.

Lemma 4 *The T-junctions at the four corners of an irreducible empty room must form a wheel structure (Figure 11).*

Proof: If an empty room X does not form a wheel structure, there is at least one slicing cut (Figure 12) on one of its four sides. By removing this slicing cut, we can merge X with the room on the other side of the slicing cut (room A in Figure 12) and X can be removed. $\square$

Lemma 5 *The adjacent rooms at the four T-junctions of an irreducible empty room must not be irreducible empty rooms themselves.*

Proof: W.l.o.g., we consider an irreducible empty room X of clockwise wheel shape, and assume that its adjacent room A sharing with X the T-junction at its upper left corner is also an irreducible empty room (Figure 13). Then A must be an anti-clockwise wheel. There are two cases: (1) If $width(A) \geq width(S)$, X can be merged with A_1 (Figure 13(a)) to form a new empty room. This empty room $X + A_1$ is reducible, and can be removed by combining with the modules on the right hand side (labeled B). (2) If $width(A) \leq width(S)$, A can be merged with X_1 (Figure 13(b)) to form a new empty room and similar argument follows. In both cases, we are able to reduce the number of irreducible empty rooms by one. By

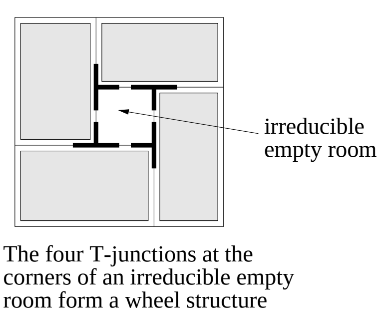

Figure 11: A wheel structure.

repeating the above process, we will either end up with only one irreducible empty room that must satisfy the condition, or the situation that every remaining irreducible empty room does not share a T-junction with each other. □

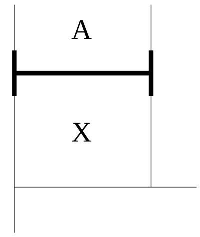

Figure 12: Proof of Lemma 4.

3.2 Mapping Between Mosaic Floorplan and General Non-slicing Floorplan

In this section, we will show how a non-slicing floorplan F' can be constructed from a mosaic floorplan F by inserting some irreducible empty rooms at the right places in F. For simplicity, we will make use of twin binary trees for explanation. That means, given a mosaic floorplan F represented by a twin binary trees t_1 and t_2, we want to insert the minimal number of empty rooms (represented by X) to the trees appropriately so that they will correspond to a valid non-slicing floorplan F', and the method should be such that every non-slicing floorplan can be constructed by this method uniquely from one and only one mosaic floorplan. To construct a non-slicing floorplan from a mosaic floorplan, we only need to consider those irreducible empty rooms, because all reducible empty rooms can be removed by merging with some neighboring rooms. From Lemma 4, we know that an irreducible empty room must be of the shape of a wheel, so its structure in the twin binary trees must be of the form as shown in Figure 14. In our approach, we will use the following mapping M_x to create irreducible empty rooms from a sliceline structure.

Definition 3 *The mapping M_x will map a vertical (horizontal) sliceline with one T-junction on each side to an irreducible empty room of anti-clockwise (clockwise) wheel shape (Figure 15).*

It is not difficult to prove the uniqueness of this mapping as stated in the next Lemma:

Lemma 6 *Every non-slicing floorplan can be mapped by M_x from one and only one mosaic floorplan.*

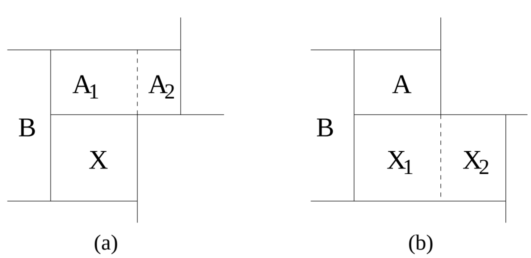

Figure 13: Proof of Lemma 5.

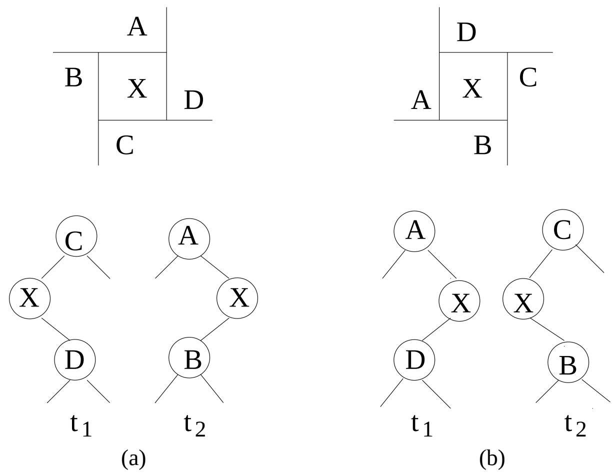

Figure 14: Tree structure of an irreducible empty room.

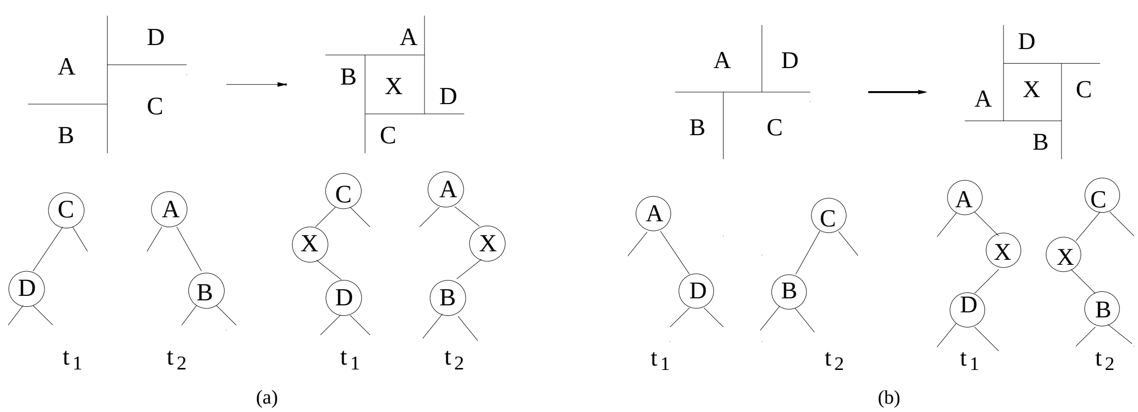

Figure 15: Mapping between mosaic floorplan and non-slicing floorplan.

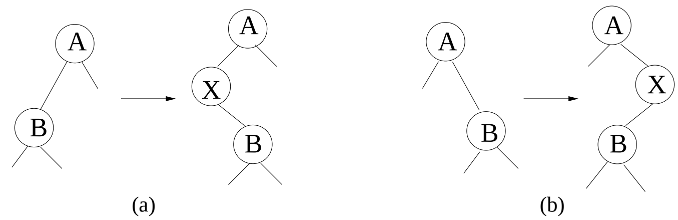

Figure 16: The only two ways to insert X into a tree.

Proof: Given a non-slicing floorplan F, each of it's irreducible empty rooms must form a wheel structure, sharing its four corners with four different modules. Each of them can only be created from one slicing structure as described in the mapping M_x. It is thus obvious that the floorplan F can only be mapped from one unique mosaic structure. $\square$

From Lemma 5, we know that the adjacent rooms of an irreducible empty room must be occupied. Therefore if we want to insert X's into the twin binary trees t_1 and t_2 of a mosaic floorplan, the X's must be inserted between some module nodes as shown in Figure 16. Given this observation, we will first insert as many X's as possible (i.e., $n-1$) into t_1 and t_2 to obtain another pair of trees t'_1 and t'_2. An example is shown in Figure 17(b). Now, the most difficult task is to select those X's that are inserted correctly. According to Observation 2, a pair of twin binary trees are valid (correspond to a packing) if and only if the inorder traversal of their extended trees are equivalent except that all the bits are reversed. Therefore, in order to find out those valid X's, we will write down the inorder traversal of the extended trees of t'_1 and t'_2 and try to match the X's. The matching is not difficult since there must be an equal number of X's between any two neighboring module names (Figure 17(c)). We may need to make a choice when there are more than one X's between two modules. For example, in Figure 17(c), there is one X between C and D in the first sequence and there are two X's in the second sequence. In this case, we can match one pair of X's. There are two choices from the second sequence, and they will correspond to different non-slicing structures as shown in Figure 17(c). Every matching will correspond to a valid floorplan, and each non-slicing floorplan can be constructed uniquely by this method from one and only one mosaic floorplan.

3.3 Inserting Empty Rooms directly on TBS

In our implementation, we do not need to build the trees explicitly to insert empty rooms. We can scan the twin binary sequences $s = (\pi, \alpha, \beta, \beta')$ once to find out all the positions of the X's in the inorder traversals of t_1 and t_2 after insertion. This is possible because of the following observation. Consider an X inserted at a node position A in a tree. If A has a left subtree B (Figure 16 (a)), this inserted X will appear just before the left subtree of A in the inorder traversal of t'. Similarly, if A has a right child B (Figure 16 (b)), this inserted X will appear just after the right subtree of A in the inorder traversal of t'. A simple algorithm can be used to break down the subtree structure of a tree and find out all the positions of the X's in the sequences after insertion in linear time. The details of the algorithm as follows.

We scan the TBS from left to right and assume that $\alpha_n = 1$. If $\alpha_i = 0$, module π_i has a right subtree in t_1 according to the definition of TBS. By the observation above, we only need to find the position of the last module (π_k) in the right subtree of π_i in t_1 from the TBS, and then insert one X just after π_k in the inorder

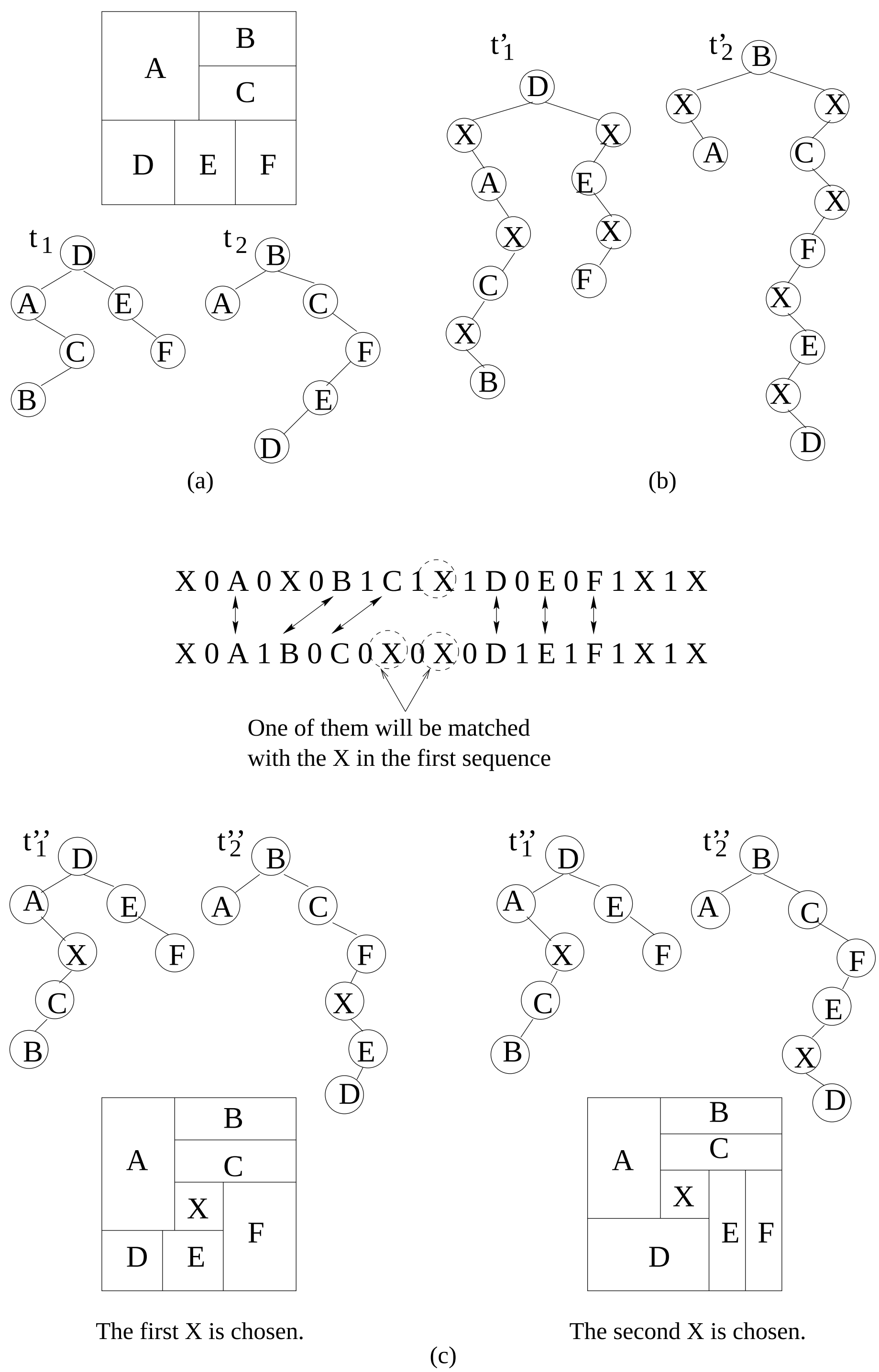

Figure 17: A simple example of constructing a non-slicing floorplan from a mosaic floorplan.

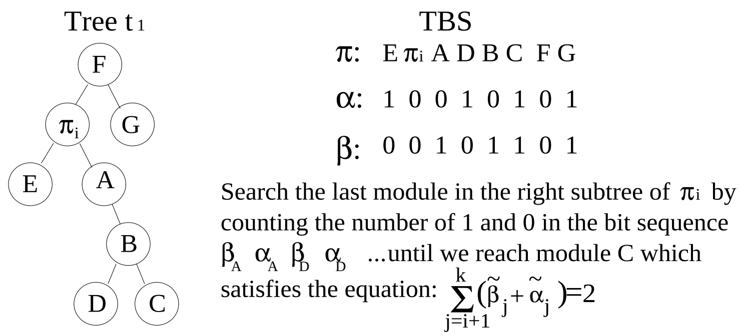

Figure 18: An example of searching the last module in the right subtree of π_i.

traversal of t_1. In addition, we will assign 1 as the labeling bit of the inserted X. Note that the right subtree of π_i can be taken as a binary tree except that the directional bit of the root is 1, not 0 as usual. In addition, $\alpha_k = 1$. Thus, we obtain the modified conditions for the right subtree of π_i as follows:

(a) In the bit sequence $\beta_{i+1}\alpha_{i+1}\beta_{i+2}\ldots\alpha_{k-1}\beta_k\alpha_k$, the number of 1's is two more than the number of 0's.

(b) For any proper prefix of the bit sequence $\beta_{i+1}\alpha_{i+1}\beta_{i+2}\ldots\alpha_{k-1}\beta_k\alpha_k$, the number of 1's is less than or equal to the number of 0's plus 1.

Based on the above conditions, we can count the number of 0 and 1 from β_{i+1} and α_{i+1} until we reach the module π_k. It is not difficult to find π_k by the following mathematical form:

$$\sum_{j=i+1}^{k}(\tilde{\beta}_j+\tilde{\alpha}_j)=2 \tag{1}$$

where we define

$$\tilde{x}=\begin{cases}1 & \text{if } x=1\\ -1 & \text{if } x=0\end{cases}$$

A simple example is shown in Figure 18. After we insert X for module π_i, the inorder traversal of the extended t_1' becomes $E1\pi_i0A0D1B0C1X1F0G$. Note that the inserted X for module π_i just appears after the last module (i.e., module C) of the right subtree of π_i in t_1. The labeling bit for the inserted X is 1.

If $\alpha_i = 1$, module π_i has a right subtree in t_2 according to the definition of TBS. Similarly, we can insert X for π_i directly into the inorder traversal of the extended t_2' by searching the last module of the right subtree of π_i in t_2. The algorithm is exactly the same as above.

Now we consider the case that π_i has a left subtree in the twin binary trees. If $\alpha_{i-1} = 1$, π_i has a left subtree in t_1. According to the observation above, we only need to find the position of the first module (π_k) in the left subtree of π_i in t_1 from the TBS, and insert one X just before π_k in the inorder traversal of t_1. In addition, we assign 0 as the labeling bit of the inserted X. Note that the left subtree of π_i in t_1 is exactly a general binary tree. In addition, $\alpha_{k-1} = 0$. We thus obtain the modified conditions for the left subtree of π_i as follows:

(a) In the bit sequence $\beta_{i-1}\alpha_{i-2}\beta_{i-2}\alpha_{i-3}\ldots\alpha_k\beta_k\alpha_{k-1}$, the number of 0's is two more than the number of 1's.

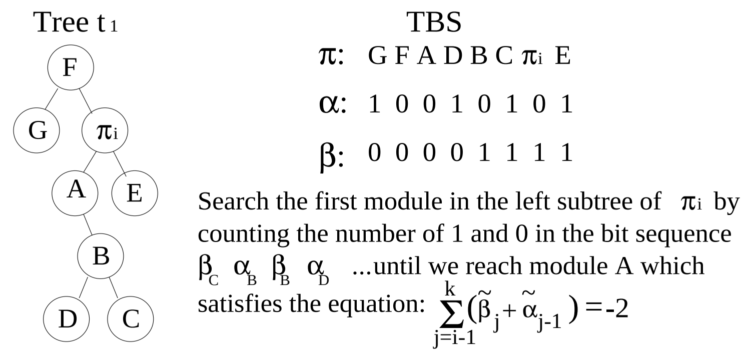

Figure 19: An example of searching the first module in the left subtree of π_i.

(b) For any proper prefix of the bit sequence $\beta_{i-1}\alpha_{i-2}\beta_{i-2}\alpha_{i-3}\ldots\alpha_k\beta_k\alpha_{k-1}$, the number of 0's is less than or equal to the number of 1's plus 1.

Based on the above conditions, we can count the number of 0 and 1 from β_{i-1} and α_{i-2} until we reach the module π_k. It is not difficult to find π_k by the following mathematical form:

$$\sum_{j=i-1}^{k}(\tilde{\beta}_j+\tilde{\alpha}_{j-1})=-2 \tag{2}$$

Another simple example is shown in Figure 19. After we insert X for module π_i, the inorder traversal of the extended t_1' becomes $G1F0X0A0D1B0C1\pi_i0E$. Note that the inserted X for module π_i appears just before the first module (i.e., module A) of the left subtree of π_i in t_1. The labeling bit for the inserted X is 0.

If $\alpha_{i-1}=0$, module π_i has left subtree in t_2. Similarly, we can insert X for π_i directly into the inorder traversal of t_2' by searching the first module in the left subtree of π_i in t_2. The algorithm is exactly the same as above.

After we inserted all the possible X's, we obtain the inorder traversals of the trees t_1' and t_2'. Matching can then be done as described in the previous subsection.

3.4 Tight Bound on the Number of Irreducible Empty Rooms

In order to describe non-slicing structure by a mosaic floorplan representation, some previous works [14, 15] include dummy blocks of zero area in the set of modules. The method described in section 2.3 is very efficient but it is applicable to the TBS representation only. In general, we only need to have $n-1$ extra dummy blocks in order to represent all non-slicing structures by a mosaic floorplan representation. We have proved an upper bound of $n-1$ and a lower bound of $n-2\sqrt{n}+1$ on the number of irreducible empty rooms in a general non-slicing floorplan. (An example with 49 modules and 36 irreducible empty rooms is shown in Figure 20.) It means that $n-1$ dummy blocks are needed and we cannot use much less.

Theorem 4 *In a non-slicing floorplan P, there can be at most $n-1$ irreducible empty rooms.*

Proof: According to Lemma 3, the adjacent rooms of an irreducible empty room in P must be occupied. Therefore, each irreducible empty room will take up four corners of some occupied rooms. Since there are

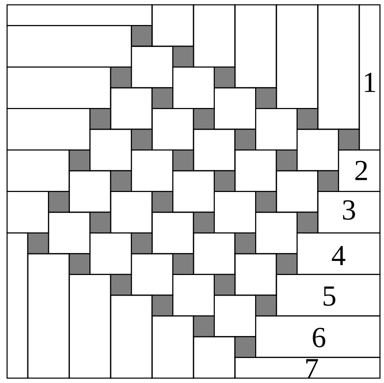

Figure 20: Floorplan example with many irreducible empty rooms.

only n occupied rooms in total and the four corners of the chip cannot be used, there are only $4n-4$ corners to be used. Therefore, there are at most $n-1$ irreducible empty rooms. □

Theorem 5 *There exists a non-slicing floorplan P of n modules and $n-2\sqrt{n}+1$ irreducible empty rooms.*

Proof: A floorplan with $n-2\sqrt{n}+1$ irreducible empty rooms can be constructed similarly to the example in Figure 20. Let k be the number of modules along each edge (for the example in Figure 20, $k = 7$), number of modules $n = k^2$ and number of empty rooms $= (k-1)^2 = (\sqrt{n}-1)^2 = n-2\sqrt{n}+1$. □

4 Floorplan Optimization by Simulated Annealing

Simulated annealing is used to search for a good TBS. The temperature is set to 1.5×10^6 initially and is lowered at a constant rate of 0.95 to 0.97 until it is below 1×10^{-10}. The number of iterations at one temperature step is 30. In every iteration of the annealing process, we will modify the TBS by one of the following four kinds of moves:

- **M1:** Swap two modules in π.
- **M2:** Change the width and height of a module.
- **M3:** Rotation based on t_1.
- **M4:** Rotation based on t_2.

We design the moves such that all TBS's are reachable. In Lemma 7, we prove that starting from any TBS, we can generate any other TBS with the same π sequence by applying one or more moves from the set $\{M3, M4\}$. Since we can swap any two modules in the π sequence by move M1 and M2 changes the dimensions of a module, all TBS's are reachable by applying moves from the set $\{M1, M2, M3, M4\}$. In addition, we will make sure that the sequences obtained after each move is a valid TBS (i.e., satisfying conditions (1)-(2)).

For move M1, we only exchange the module names in two randomly selected rooms. For move M2, we change the width and height of a module within the given limits of its aspect ratio. Obviously, both move

M1 and M2 takes $O(1)$ time. For move M3 and M4, we borrow and modify the idea of rotations in red-black tree [2]. A red-black tree is a binary search tree. The rotation in a red-black tree is an operation that changes the tree structure locally while preserving the inorder traversal of the tree. Two kinds of rotations, Right-Rotate and Left-Rotate, are defined originally in [2] (Figure 21). A and B represent two nodes. C, D and E represent arbitrary subtrees. Right-Rotate(T, A) transforms the left tree structure to the right tree structure, while keeping the inorder traversal of the tree unchanged (e.g., the inorder traversal of the tree before and after rotation are both equal to $CBDAE$ in Figure 21). The operation of left rotation is similar. Both Left-Rotate and Right-Rotate run in $O(1)$ time. When we apply red-black tree rotations on our twin binary trees, the subtree D in Figure 21 should not be 1 or 0. In the case that subtree D is 1 or 0, we modify the red-black rotations as shown in Figure 22, where D is designated to 0 or 1 after Right-Rotate(T, A) or Left-Rotate(T, B).

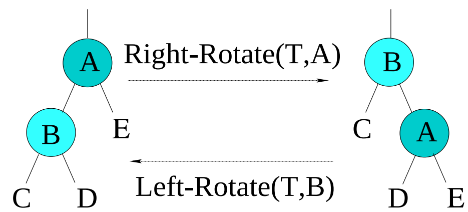

Figure 21: Right-Rotate and Left-Rotate for a binary search tree

.

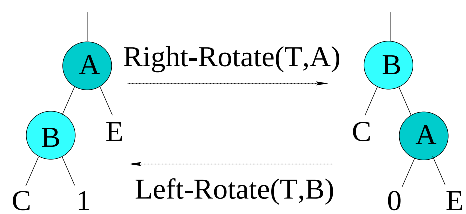

Figure 22: Modified red-black rotations when subtree D is 0 or 1

.

For the moves M3 and M4, we randomly pick one module π_i from π, and check α_i. If $\alpha_i = 0$, π_i has a right child in t_1 and π_{i+1} has a left child in t_2. We can then use move M3 to apply Left-Rotate(T, π_i) on t_1 or use move M4 to apply Right-Rotate(T, π_{i+1}) on t_2. They are similar to each other and one of them will be randomly picked and applied. W.l.o.g., we present the details of Left-Rotate(T, π_i) on t_1 according to the following four cases shown in Figure 23(a), (b), (c) and (d). For simplicity, we use letter B, C and D to represent the root of each subtree.

Case 1: $\beta_i = 0$ and the right child of π_i has a left child.
Case 2: $\beta_i = 1$ and the right child of π_i has a left child.
Case 3: $\beta_i = 0$ and the right child of π_i has no left child.
Case 4: $\beta_i = 1$ and the right child of π_i has no left child.

For Case 1, after left rotation of module π_i, the only change in t_1 is the directional bit of module A and C, so we only need to flip β_A and β_C. Because the labeling sequence α does not change, we do not need to update t_2. Thus, we keep β' the same as before. Case 2 is similar to Case 1. For Case 3, both α_i and the

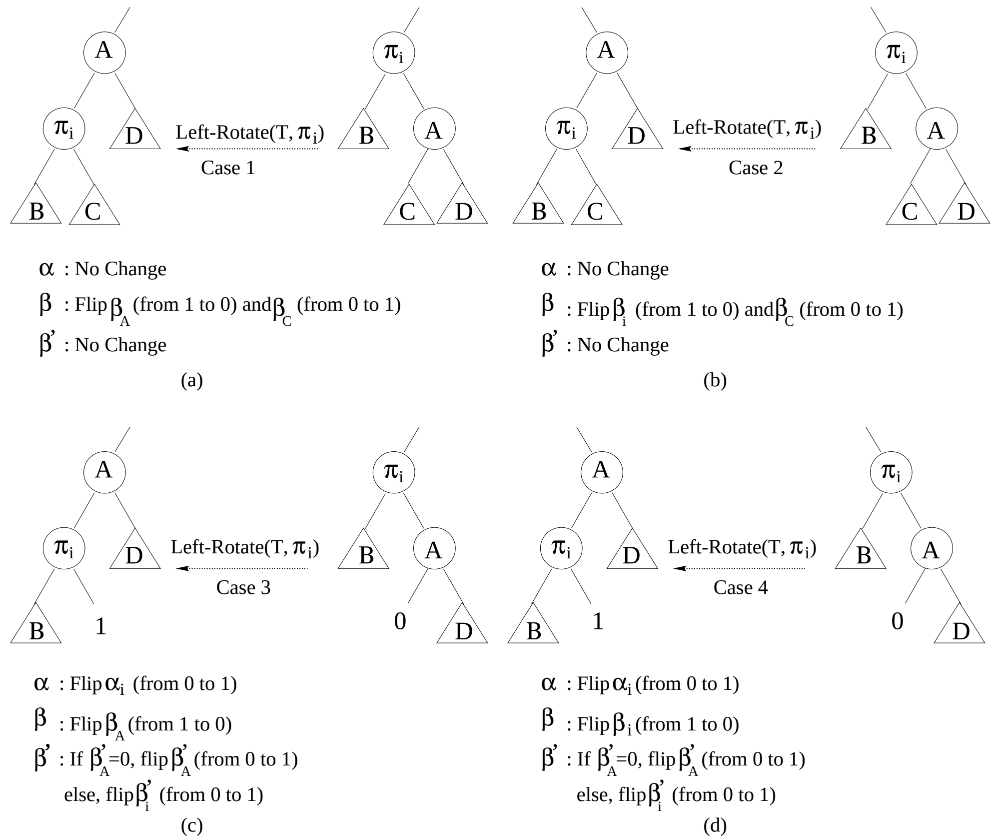

Figure 23: Four cases of Left-Rotate(T, π_i) on t_1

.

directional bit of module A are flipped after left rotation of module π_i. In order to maintain conditions (1) and (2), we need to update t_2 by flipping one directional bit of β' from 0 to 1. Note that π_i is the left child of A in t_2. Thus, if β'_A is 0, we will flip β'_A from 0 to 1. Otherwise, we will flip β'_i from 0 to 1. Case 4 is similar to Case 3. Actually, updating t_2 in case 3 and 4 is exactly the Right-Rotate(T, A) on t_2 in case 3 and 4.

If $\alpha_i = 1$, π_i has a right child in t_2 and π_{i+1} has left child in t_1. We can thus use move M4 to apply Left-Rotate(T, π_i) on t_2 or use move M3 to apply Right-Rotate(T, π_{i+1}) on t_1. One of them will be randomly picked and applied. The algorithm of right rotation is similar to that of left rotation.

In move M3 and M4, if α does not change, we only need to update one tree and each move takes $O(1)$ time. If α changes, we need to update both trees (i.e., apply two rotations). Therefore, both move M3 and M4 take $O(1)$ time in practice.

Lemma 7 *Starting from any TBS, we can generate any other TBS with the same π sequence by applying one or more moves from the set $\{$M3, M4$\}$*

Proof: We observe that at most $n - 1$ Left Rotations suffice to transform any arbitrary n-node binary tree into a left-going chain [2]. Given a TBT, w.l.o.g., we can apply at most $n - 1$ Left Rotations by move M3. The binary tree t_1 will become a left-going chain (Figure 24(a)). Since move M3 always results in a

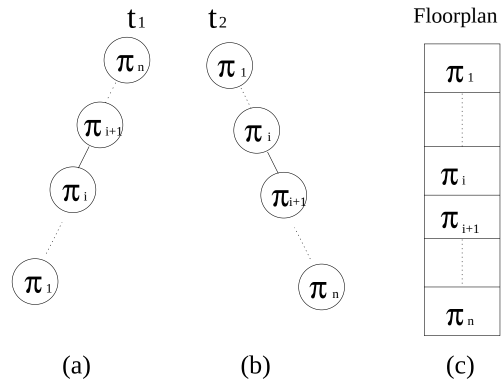

Figure 24: Proof of Lemma 7.

TBT, the binary tree t_2 must also be transformed into a right-going chain (Figure 24(b)). The corresponding floorplan is shown in Figure 24(c).

Noticing that any Left Rotation in move M3 has its reversed rotation which is the Right Rotation, a n-node TBT where t_1 is a left-going chain and t_2 is a right-going chain can thus be transformed into any other arbitrary TBT by applying at most $n - 1$ Right Rotations by move M3. Therefore, at most $2n - 2$ moves are sufficient to convert a TBS to any other arbitrary TBS with the same π sequence. We design move M4 as a symmetric move to M3. □

5 Experimental Results

All experiments are carried out on a PC with 1400MHz Intel Xeon Processor and 256Mb Memory. Simulated annealing as stated in section 4 is used to search for a good TBS.

We test our algorithm using TBS with empty room insertion on six MCNC benchmarks. Besides, we also run the algorithm with empty room insertion disabled. In other words, only mosaic floorplan can be generated. For each case, two objective functions are considered. The first is to minimize area only. The second is to minimize a weighted sum of area and wirelength. The weights are set such that the costs of area and wirelength are approximately equal. Because of the stochastic nature of simulated annealing, for each experiment, ten runs are performed and the result of the best run is reported. The results for area minimization is listed in Table 1. The results for area and wirelength minimization is listed in Table 2.

As the results show, our floorplanner can produce high-quality floorplans in a very short runtime. We also notice that empty room insertion is very effective in reducing the floorplan area. If empty room insertion is disabled, the deadspace is worse for all but two cases. The deadspace is 32.84% more on average. However, with empty room insertion, the floorplanner is about 40.8% slower.

In Table 3, we compare our results with ECBL [14] and the enhanced Q-sequences [15]. Notice that ECBL is run on Sun Sparc20 (248MHz) while Enhanced Q-seq is run on Sun Ultra60 (360MHz). We found that the scaling factors for the speeds of the three machines are 1:1.68:5.03. The runtimes reported in brackets in Table 3 are the scaled runtimes. We can see that the run time of TBS is much faster, although the performance of all three of them in area optimization are similar. We also compared TBS with those representations designed intrinsically for slicing structure. The performance of Fast-SP [11], Enhanced O-

MCNC benchmark	TBS (with X)		TBS (no X)	
	% Dead-space	Run-time (s)	% Dead-space	Run- (s) time (s)
apte	1.89	0.86	1.30	0.73
xerox	2.17	1.30	2.46	1.20
hp	2.10	0.76	2.22	0.63
ami33a	3.05	1.26	4.05	0.98
ami49a	4.05	2.55	4.38	2.08
playout	6.20	2.58	7.60	1.09

Table 1: Area minimization.

MCNC benchmark	TBS (with X)			TBS (no X)		
	% Dead-space	Wire-length	Run-time (s)	% Dead-space	Wire-length	Run-time (s)
apte	1.79	12652	0.89	3.45	13267	0.62
xerox	2.64	14937	1.36	4.41	14738	1.22
hp	1.32	4246	0.73	3.43	4292	0.61
ami33a	8.41	6078	1.30	7.25	6488	1.02
ami49a	9.40	29668	2.60	10.82	30256	2.14
playout	5.19	2.373	2.50	6.32	2.265	1.08

Table 2: Area and wirelength minimization.

tree [9], B*-tree [1] and TCG [6] are shown in Table 4. Notice that Fast-SP and B*-tree are run on Sun Ultra1 (166MHz) while Enhanced O-tree and TCG are run on Sun Sparc20 (248MHz), and the scaling factors for their speeds are 0.613:1. Again, the runtimes reported in brackets in Table 4 are the scaled runtimes. We can see that TBS has again out-performed the other representations in terms of runtimes, while the packing quality in terms of area is similar. TBS is thus a more desirable representation since its fast computation allows us to handle very large circuits and to embed more interconnect optimization issues in the floorplanning process.

References

[1] Y.C. Chang, Y.W. Chang, G.M. Wu and S.W. Wu. B*-Trees: A New Representation for Non-Slicing Floorplans. *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pages 458–463, 2000.

[2] T. H. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to Algorithms. *The MIT Press*, pages 265–266. 1990.

[3] S. Dulucq and O. Guibert. Baxter Permutations. *Discrete Mathematics*, 180:143–156, 1998.

MCNC benchmark	Total area	ECBL [14][1]		Enhanced Q-seq [15][2]		TBS	
		Area	Runtime (s)	Area	Runtime (s)	Area	Runtime (s)
apte	46.56	45.93 [3]	3 (3)	46.92	0.35 (0.59)	47.44	0.86 (4.33)
xerox	19.35	19.91	3 (3)	19.93	3.6 (6.05)	19.78	1.3 (6.54)
hp	8.30	8.918	11 (11)	9.03	3.5 (5.88)	8.48	0.76 (3.82)
ami33	1.16	1.192	73 (73)	1.194	40 (67.2)	1.196	1.26 (6.34)
ami49	35.4	36.70	117 (117)	36.75	57 (95.76)	36.89	2.55 (12.83)

[1] Using Sun Sparc20 machine [2] Using Sun Ultra60 workstation [3] Negative deadspace

Table 3: Comparisons with ECBL and enhanced Q-sequences.

MCNC benchmark	Fast-SP [11][1]		Enhanced O-tree [9][2]		B*-tree [1][1]		TCG [6][2]	
	Area	Runtime (s)	Area	Runtime (s)	Area	Runtime (s)	Area	Runtime (s)
apte	46.92	1 (0.61)	46.92	11 (11)	46.92	7 (4.29)	46.92	1 (1)
xerox	19.80	14 (8.58)	20.21	38 (38)	19.83	25 (15.33)	19.83	18 (18)
hp	8.947	6 (3.68)	9.16	19 (19)	8.947	55 (33.72)	8.947	20 (20)
ami33	1.205	20 (12.26)	1.242	119 (119)	1.27	3417 (2095)	1.20	306 (306)
ami49	36.5	31 (19.00)	37.73	406 (406)	36.8	4752 (2913)	36.77	434 (434)

[1] Using Sun Ultra1 machine [2] Using Sun Ultra60 machine

Table 4: Comparisons with other representations for slicing floorplan.

[4] Pei-Ning Guo, Chung-Kuan Cheng and Takeshi Yoshimura. An O-Tree Representation of Non-Slicing Floorplan and Its Applications. *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[5] Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, Chung-Kuan Cheng and Jun Gu. Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 8–12, 2000.

[6] Jai-Ming Lin and Yao-Wen Chang. TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans. *Proceedings of the 38th ACM/IEEE Design Automation Conference*, pages 764–769, 2001.

[7] H. Murata, K. Fujiyoushi, S. Nakatake and Y. Kajitani. Rectangle-Packing-Based Module Placement. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 472–479, 1995.

[8] S. Nakatake, K. Fujiyoushi, H. Murata and Y. Kajitani. Module Placement on BSG-Structure and IC Layout Applications. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 484–491, 1996.

[9] Yingxin Pang, Chung-Kuan Cheng and Takeshi Yoshimura. An Enhanced Perturbing Algorithm for Floorplan Design Using the O-tree Representation. *Proceedings of International Symposium on Physical Design*, pages 168–173, 2000.

[10] K. Sakanushi and Y. Kajitani. The Quarter-State Sequence (Q-Sequence) to Represent the Floorplan and Applications to Layout Optimization. *Proceedings of IEEE Asia Pacific Conference Circuits and Systems*, pages 829–832, 2000.

[11] Xiaoping Tang and D.F. Wong. FAST-SP: A Fast Algorithm for Block Placement based on Sequence Pair. *Proceedings of IEEE Asia South Pacific Design Automation Conference*, pages 521–526, 2001.

[12] D.F. Wong and C.L. Liu. A New Algorithm for Floorplan Design. *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[13] B. Yao, H. Chen, C.K. Cheng and R. Graham. Revisiting Floorplan Representations. *Proceedings of International Symposium on Physical Design*, pages 138–143, 2001.

[14] S. Zhou, S. Dong, X. Hong, Y. Cai and C.-K. Cheng. ECBL: An Extended Corner Block List with Solution Space Including Optimum Placement. *Proceedings of International Symposium on Physical Design*, pages 156–161, 2001.

[15] C. Zhuang, K. Sakanushi, L. Jin and Y. Kajitani. An Enhanced Q-Sequence Augmented with Empty-Room-Insertion and Parenthesis Trees. *Proceedings of Design, Automation and Test in Europe*, pages 61–68, 2002.